# Wise Timetable — Importing Companion

## A catalogue of every import and data operation in the desktop application


Wise Technologies · reference for the csvimport module · August 2026

# Table of Contents

## About this companion

Wise Timetable can read data from outside. Over the years this has grown into three distinct groups of routines, all living in one place in the program:

- **Standard CSV imports** on the `Data ▸ CSV import` menu — rooms, lecturers, programs, courses, reservations, students, exams. These are the documented, general-purpose imports every installation can use.
- **Custom import codes** — reached through `Data ▸ Custom import`, which asks for a short code before it asks for a file. Each code is a purpose-built reader for one particular file layout. There are 67 of them.
- **Custom operations** — reached through `Data ▸ Custom operation`, which asks for a code and then either converts something in the open schedule or writes an export file.

This companion lists all of them: what each one does, how it is started, the exact columns it expects, a short sample file, and — where it matters — the traps, the variants, and the order in which several imports have to be run.

The descriptions are derived from the source code, not from memory. Where the code does something surprising, that is written down as a note rather than quietly smoothed over: an import that silently skips a line is more dangerous than one that refuses it, and knowing which is which is most of the value of a document like this.

**About this revision.** The catalogue is written from the source code of the current build, and writing it prompted a full review of the import module. The notes below describe how each import behaves now. Where an unchanged file produces a different result from an earlier build, the note says so and is marked *Changed in this revision*. If you are running an older build, read those notes the other way round.

### How to read a column table

Column numbers are **1-based** and count the fields as they appear in the file, separated by the separator named in the *File format* paragraph. A column marked *optional* may be left empty; a column marked *(ignored)* is read past but never used — it must still be present so that the columns after it land in the right place.

# Shared mechanics

## The custom import prompt

`Data ▸ Custom import` first shows a prompt asking for the import code (maximum 16 characters), then a file dialog filtered on `*.csv`, `*.xlsx` and `*.xls`.

- **The code is upper-cased**, so `kp` and `GRIDLEGEND` are the same thing.
- **The prompt remembers the last code used.** Confirming an empty prompt repeats it. This is convenient and it is also the easiest way to run the wrong import on the right file.
- **`.xlsx` / `.xls` files are converted to CSV first**, so a workbook may be selected directly.
- **Files are opened through the UTF-8 reader.** A legacy ANSI export (CP1250 and similar) is transcoded on the way in, so files with diacritics do not have to be converted first.

## The -d and -f suffixes

Two suffixes may be appended to the code at the prompt, separated by a space:

| Suffix | Meaning |
|---|---|
| `-d` | Raises the debug logging level for this run |
| `-f` | Sets the *delete before import, without update* flag |

They are stripped from the code before it is upper-cased, so **they must be typed in lower case**: `MODCRSEVENTS -d`, not `MODCRSEVENTS -D`. Most handlers ignore both; the ones that read them say so in their own Notes. The `-f` flag is **not reset between runs** in the same session — once used, it stays set until the application is restarted.

## Two passes, and passes around the file

A custom import may run more than one pass over the data:

1. **A first parsing pass** over the file, used by `GRIDLEGEND` (and declared but inert for `ENRLC`), typically to collect a legend or dictionary before the main data is read.
2. **A preprocessing step** that touches no lines at all — `DBBOOKINGS`, `DBBOOKINGSDEL` and `ENRLSTD` use it to prepare or clear the database.
3. **The main parsing pass**, which is where the column tables in this document apply.
4. **A postprocessing step** after the file is closed — `CURRICULUM`, `PARTTIME` and `DBBOOKINGS`/`DBBOOKINGSDEL` use it to link up loose ends or to show one collected report.

Where this happens it is stated in the entry.

## Warnings, logs and what “0 warnings” really means

At the end of a custom import a popup reports how many lines were read and how many produced warnings. **That count is not a quality check.** Many handlers skip a line by returning a value the dispatcher treats as success, so a file that matched nothing at all can finish with zero warnings.

The reliable record is `wtt_event.log`. After any import that matters, open the log and look for `Could not parse line no …` and the handler’s own messages.

## Saving

Almost every import changes the schedule **in memory only** and marks it as needing to be saved. Nothing is written until the schedule is saved. The exception is `DBBOOKINGS`/`DBBOOKINGSDEL`, which writes reservations into the database directly with SQL, and `LASTWEEK`, which writes one web setting.

# Part 1 — Standard CSV imports

These are on the `Data ▸ CSV import` menu. They share one handler, which means they also share two global settings:

- **Alternative import** (`MISC_ALTERNATIVE_IMPORT`) changes the expected column layout of several of them, and for *Courses* it makes the **file name** decide which import step runs: a path containing `1.csv` is step 1, `2.csv` step 2, and so on up to `6.csv`. Anything else defaults to step 1.
- **FIS courses format** (`L_ADV_FIS_COURSES_FORMAT`) runs the chosen file through a converter before the import sees it.

Both are described in full under *Courses*.

## Data ▸ CSV import ▸ Rooms

**What it does.** Reads a room list and either creates the rooms that do not exist yet or updates the ones that do (matched by exact room name). It sets the number of seats and attaches room properties (equipment), creating any property that is not yet defined. Nothing is ever deleted: the import is purely additive/updating.

**How to start it.** `Data ▸ CSV import ▸ Rooms`

**File format.** Plain CSV, one line per room, no header line is expected (a line shorter than 2 characters or starting with `;` is skipped). The field separator is the character configured in the application settings, typically `;` or `,`. Fields may be double-quoted; `\"` inside a quoted field yields a literal `"`. A field whose text is exactly the configured placeholder string is treated as empty. Read through the UTF-8 aware open helper, so legacy ANSI exports are transcoded on the way in.

Standard layout (alternative import switched off):

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Room name | Key. If no room with this name exists a new one is created. | `A-001` |
| 2 | Number of seats | Integer, non-numeric text becomes 0. | `60` |
| 3..n | Room property | Optional, repeatable to the end of the line. Each non-empty field is added as a room property; unknown property names are created. Empty fields are skipped. | `computers` |

Layout when the alternative import setting is on:

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Room code | Read but not stored; the room is identified by the name in column 2. | `RM-01` |
| 2 | Room name | Key. If no room with this name exists a new one is created. | `A-001` |
| 3 | Number of seats | Integer. | `60` |
| 4 | Room property | Optional, single value only. It is wrapped in round brackets before lookup, so the property stored is `(computers)`. | `computers` |

**Sample**

```
A-001;60;computers;projector
B-102;180;lecture hall
LAB-3;24;computers
```

**Notes.**

- **Changed in this revision.** The alternative layout used to read column 1 twice, so a `code;name;seats;property` file was read as `name;seats;property`: the room was named after the code column and its seat count came out as 0. The columns now advance correctly, which means files written for the old behaviour (name in column 1) have to gain a leading code column.
- In the standard layout an unlimited number of property columns is supported; in the alternative layout only one, and it is bracketed.
- Rooms must be imported before Reservations, Exams and Courses, because those imports look rooms up by name and, for exams, fail the line when the room does not exist.

## Data ▸ CSV import ▸ Tutors

**What it does.** Imports lecturers/tutors. A tutor is looked up by first name plus surname and created if not found, then code, role, e-mail, password and note are filled in from the remaining columns. Existing tutors are updated in place; no tutor is ever deleted.

**How to start it.** `Data ▸ CSV import ▸ Tutors`

**File format.** Plain CSV, one line per tutor, no header line is expected (lines shorter than 2 characters are skipped). Separator, quoting and placeholder handling are as for Rooms.

Standard layout:

| # | Column | Description | Example |
|---|---|---|---|
| 1 | First name | Key part 1. | `James` |
| 2 | Surname | Key part 2. Tutor is created if the pair is unknown. | `McClusky` |
| 3 | Code | Optional. Special case: if the value contains `@` it is stored as the e-mail instead, the *whole rest of the line* is stored as the note, and columns 4..7 are not parsed. | `T0123` |
| 4 | Role | Optional. Empty leaves the role at its previous value; if the tutor still has no role, `100` is set. | `100` |
| 5 | E-mail | Optional. | `james.mcclusky@example.edu` |
| 6 | Password | Optional, stored as given. | `initialPwd` |
| 7 | Note | Optional. | `part-time` |

Layout when the alternative import setting is on:

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Code | Stored as the tutor code. | `T0123` |
| 2 | Full name | `Surname Firstname` or `Surname, Firstname`. Split at the **last** space: everything before it is the surname (a trailing comma is stripped), everything after it is the first name. A value with no space is rejected and the line fails. | `McClusky, James` |
| 3 | E-mail | Stored as the tutor e-mail. | `james.mcclusky@example.` |

| # | Column | Description | Example |
|---|---|---|---|
| | | | `edu` |
| 4 | Note | Optional. | `part-time` |

**Sample**

```
James;McClusky;T0123;100;james.mcclusky@example.edu;;part-time
Anna;Bergstrom;T0124;;anna.bergstrom@example.edu
```

**Notes.**

- Because of the `@` short-cut in column 3, a file that puts the e-mail in the code position silently loses role and password and turns the tail of the line into the note.
- Multi-word first names are not supported in the standard layout (they must be a single field), while the alternative layout supports multi-word surnames because it splits on the last space only.
- Tutors must exist before Courses, Courses2, Exams and Exam data are imported — those imports resolve people by name or by code.

## Data ▸ CSV import ▸ Tutors LDAP

**What it does.** Fills the LDAP user name of already existing tutors. It does not create tutors and it does not change any other tutor field. After every processed line the user table in the database is updated.

**How to start it.** `Data ▸ CSV import ▸ Tutors LDAP`

**File format.** Plain CSV, one line per tutor, no header line is expected (lines shorter than 2 characters are skipped). Only two columns are read; surrounding whitespace is removed from both.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Matching key | Three forms, auto-detected: contains `@` — matched against the tutor e-mail; contains `/` — read as `FirstName/Surname` and matched against name plus surname; neither — matched against the tutor code. | `james.mcclusky@example.edu` |
| 2 | LDAP user name | Value written to the tutor's LDAP name. | `jmcclusky` |

**Sample**

```
james.mcclusky@example.edu;jmcclusky
James/McClusky;jmcclusky
T0124;abergstrom
```

**Notes.**

- The line processor always reports success, even when no tutor matched the key. Unmatched lines are therefore *not* counted as warnings and the import still finishes with a success message — check the log if the LDAP names do not appear.
- Only the first matching tutor is updated; the search stops at the first hit.
- Run this after the Tutors import, never before.

## Data ▸ CSV import ▸ Programs

**What it does.** Creates or updates study programs. In the standard layout one line is one program (name, code, number of years). In the alternative layout one line describes a branch together with its parent department, and branches are created for every year from 1 to the given number of years.

**How to start it.** `Data ▸ CSV import ▸ Programs`

**File format.** Plain CSV, one line per program (standard) or per branch (alternative). No header line is expected; lines shorter than 4 characters are skipped. Separator and quoting as for Rooms; surrounding whitespace is trimmed from every field in the alternative layout.

Standard layout:

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Program name | Key; program is created if unknown. | `Undergraduate` |
| 2 | Program code | Optional (only applied when the column is present). | `BAC` |
| 3 | Number of years | Optional. Set as-is, including 0 if the text is not numeric. | `3` |

Layout when the alternative import setting is on:

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Branch code | Written to the code of every branch created by this line. | `BCIT` |
| 2 | Branch name | Branch name, one branch per year. | `Business Computing and IT` |
| 3 | Number of years | Values of 0 or less are forced to 1. Also raises the program's year count. | `3` |
| 4 | Department name | Used as the **program** name (key; program is created if unknown). | `Faculty of Computing` |
| 5 | Department code | Written as the program code. | `FC` |
| 6 | Number of students | Per branch. A value of 0 (or non-numeric) is forced to 2. | `120` |
| 7 | Faculty id | Integer, stored on each branch. | `10` |

**Sample**

```
Undergraduate;BAC;3
Postgraduate;MSC;2
```

Alternative layout:

```
BCIT;Business Computing and IT;3;Faculty of Computing;FC;120;10
```

**Notes.**

- Standard layout creates programs only; branches come from the Courses import or from the alternative layout here.
- Alternative layout is the first step of a chained import: it must run before the alternative Courses steps, because those resolve branches by code and year.
- Forcing an empty student count to 2 is marked as questionable in the source; supply the real number.

## Data ▸ CSV import ▸ Constraints

**What it does.** Applies scheduling constraints to every turn of every course part of the courses matched by the first column. It can allow or forbid the sixth day of the week, pin a start hour, say whether that hour is an exact start / earliest / latest, or hand a free-form advanced timing string to the timing parser.

**How to start it.** `Data ▸ CSV import ▸ Constraints`

**File format.** Plain CSV, one line per course, no header line is expected; lines shorter than 4 characters are skipped. Surrounding whitespace is stripped from every field. Column 2 decides which of two shapes the line has: if it contains `:` or `,` the line is an advanced timing line and columns 3 and 4 are not read.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Course code or name | Matched first against the course code, then against the course name. All matching courses are affected; courses without course parts are skipped. | `DB1` |
| 2 | Saturday flag **or** advanced timing string | Plain number: 1 = allow the sixth day for all turns, 0 = forbid it. If the value contains `:` or `,` it is instead passed verbatim to the advanced timing parser and columns 3-4 are ignored. | `1` |
| 3 | Start hour index | 0-based time-slot index. `-1` = no fixed start hour. `-2` = leave the hour settings untouched (only the Saturday flag is applied). Values above the last slot index reject the line. | `5` |
| 4 | Before/after flag | `0` = start exactly at that slot, `1` = at or after, `-1` = at or before. Any other value rejects the line. | `0` |

**Sample**

```
DB1;1;5;0
MATH101;0;-1;0
PHYS200;1:8,3:10
```

**Notes.**

- Courses must already exist (import Courses first). Lines whose course is not found are silently ignored — no warning is produced.
- **Changed in this revision.** The caller used to treat the processor's success value as a failure, so every good line was logged as unparsable and counted as a warning — and a file of 2047 lines or more aborted with the I/O error popup. Only a real parse error is a warning now, so the completion popup finally means something for this import.
- The hour index is a time-slot index, not a clock hour.

## Data ▸ CSV import ▸ Courses

**What it does.** The main teaching-load import. Each line describes one turn (and optionally its scheduled terms) of one course part, together with the program, year, branch, course, execution type, week range, weekly hours, tutor and student groups. Programs, branches, courses, course parts, turns, turn parts and groups are auto-created as needed, and fixed terms given in the line are turned into schedule entries.

**How to start it.** `Data ▸ CSV import ▸ Courses`

**File format.** Plain CSV, one line per turn. No header line is expected; a line whose first column starts with `#` is skipped, empty lines are skipped. Separator, quoting and placeholder handling as for Rooms

— except the trailing timing columns, which are read with a helper that accepts **both** `,` and `;` as separators regardless of the configured one.

**Line types.** The file contains **one line type only** — the flat course line documented below. The source also carries an older indentation-based line-type scheme, in which the number of leading empty columns identified the line type. That machinery has no caller anywhere in the file and is dead code today. For reference, the types it defines are:

| Type | Name | Leading empty columns | Handler present |
|---|---|---|---|
| 0 | Program line | none | no dedicated handler |
| 1 | Year line | 1 | yes (year number only; needs a current program) |
| 2 | Branch line | 2 | yes (branch name, optional branch code) |
| 3 | Course line | 3 | no dedicated handler |
| 4 | Course part line | 4 | no dedicated handler |
| 5 | Turn line | 5 | yes (list of tutor names `Surname,FirstName`) |

Columns of the live course line:

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Program name | Key; program is created if unknown. A leading `#` marks the whole line as a comment. | `Undergraduate` |
| 2 | Program code | Optional (applied only when non-empty). | `BAC` |
| 3 | Year | Must be 1 or more and within the maximum; otherwise the line fails. Raises the program's year count when larger. | `1` |
| 4 | Branch name | Present only in the full edition; omitted in the WTT Basic build, where name and code are derived automatically. | `Business Computing and IT` |
| 5 | Branch code | Optional (applied only when non-empty). Basic build: absent. | `BCIT` |
| 6 | Course name | Key together with the branch; course is created if unknown. | `Databases` |
| 7 | Course name translation | Optional, may be empty. Stored as the course translation when non-empty. | |
| 8 | Course code | Optional (applied only when non-empty). Also used when looking the course up. | `DB1` |
| 9 | Execution type | Course part type (lecture, tutorial, lab, seminar…). A course part of that type is created if missing. | `tutorial` |
| 10 | Begin week | Must be 1 or more. | `1` |
| 11 | End week | Must be 52 or less and not before the begin week. | `7` |
| 12 | Hours per week | Either a plain list of per-week durations (`4`, `2+2`) or a period list `week[-week]:hours[,week[-week]:hours…]`. | `1:6,3-5:2+2` |
| 13 | Tutor first name | Tutor is created if the pair is unknown; the line fails if no tutor id results. | `James` |
| 14 | Tutor surname | | `McClusky` |

| # | Column | Description | Example |
|---|---|---|---|
| 15 | Tutor code | Optional, may be empty. Warning: if it contains `@` the tutor reader switches to its e-mail short-cut and the rest of the line is swallowed as the tutor note. | |
| 16 | Group info | Optional. Comma-separated group specifications, each `name` or `name(students[,parent[,class[,email]]])`. Groups are created when unknown. | `GroupBAC` |
| 17 | Turn index | 1-based index of the turn inside the course part (offset by the number of turns that existed before this import). Empty or out of range creates a new turn. | `1` |
| 18 | Turn part index | 1-based index of the turn part. May carry a name in round brackets, e.g. `1(Lab A)`. | `1` |
| 19..n | Fixed term | Optional, repeatable. Format `DayHH[:MM]@Room`, where Day is the first 3 letters of the localized full day name, and Room is either a room name or `#id`. Each entry creates one schedule entry. | `Mon7@A-001` |

Group specification inside column 16:

| Part | Description | Example |
|---|---|---|
| 1 | Group name | `G1` |
| 2 | Number of students (first value in brackets) | `(30)` |
| 3 | Parent group name (second value in brackets) | `(30,YEAR1)` |
| 4 | Class number (third value in brackets; defaults to 1 when only a parent is given) | `(30,YEAR1,1)` |
| 5 | Group e-mail (fourth value in brackets) | `(30,YEAR1,1,g1@example.edu)` |

**Sample**

```
Undergraduate;BAC;1;Business Computing and IT;BCIT;Databases;;DB1;tutorial;1;7;1:6,3-5:2+2;James;McClusky;;GroupBAC;1;1;Mon7@A-001;Wed10@A-002;Fri9:30@A-001
Undergraduate;BAC;1;Business Computing and IT;BCIT;Databases;;DB1;lecture;1;15;2;Anna;Bergstrom;;G1(30,YEAR1,1,g1@example.edu),G2(28,YEAR1,1);1;1
```

**Notes.**

- **Alternative import.** When the *alternative import* setting is on, the file *name* selects the import step before the file is opened: a path containing `1.csv` sets step 1, `2.csv` step 2, and so on up to `6.csv`; anything else defaults to step 1. Only the Courses import consumes the step. Steps 5 and 6 do nothing. The steps must be run in order 1 » 2 » 3 » 4, and after step 2 the import automatically creates exactly one group per branch, named `g1_<branchCode>(<year>)` with the branch student count.
  - **Step 1 (`1.csv`)**, one course per line: 1 course code (key, course created), 2 course name, 3 required equipment (looked up as `(value)` among room properties), 4 number of students (0 becomes 2), 5 faculty id.
  - **Step 2 (`2.csv`)**, one course/branch link per line: 1 branch code, 2 year (0 becomes 1), 3 course code (course must exist), 4 period code (`S1` = 0, `AN` = 1, anything else = 0). Links the course to the branch, creating the branch for that year from the year-1 branch if needed, and creates a `Lecture` course part when the course has none.

  - **Step 3 (`3.csv`)**, one enrolment count per line: 1 branch code, 2 course code, 3 year (0 or less becomes 1), 4 number of students (0 becomes 1). Course and branch must exist, otherwise the line is skipped silently.
  - **Step 4 (`4.csv`)**, one turn per line: 1 course code, 2 total hours (a `.0` suffix is cut), 3 number of weeks, 4 tutor code. The tutor must already exist and be findable by code — otherwise the line is skipped and a message asks for a dummy lecturer with that code. Week range comes from the period code set in step 2 (period 0 = weeks 1–14, period 1 = weeks 1–36); group allocation and student counts are computed from the data collected in steps 2 and 3.
- **FIS courses format and the FIS conversion.** When that advanced flag is set, the chosen file is first run through a converter before any import step. The converter skips the first line and reads 19 columns per line: program name, program code, year, branch name, branch code, course name, course code, semester, collaborator type, first week, last week, total hours, lecture hours, tutorial hours, lab hours, seminar hours, tutor surname, tutor first name, tutor code. It then writes one output line per non-zero hour bucket (types `predavanje`, `vaje`, `laboratorijske vaje`, `seminar`) into `tempFISfile.csv` in the working directory and redirects the import to that file. An empty branch name is replaced by the program name. **Two traps:** the converted line has 14 columns and omits the course-name-translation column, so it no longer lines up with column 7 onwards of the current course line; and it writes surname before first name while the course line expects first name first. A line shorter than 10 characters makes the converter stop and report success, silently truncating the rest of the file; parse errors also end up reported as success.
- The import does not depend on the Program/Branch/Year selected in the workspace — every line carries its own program, year and branch.
- **Deleting of previously imported data:** before reading, the import records how many turns each course part already had; after reading, every pre-existing turn whose week period overlaps a newly imported turn of the same course part is deleted together with its schedules. Turns that do not overlap survive.
- **Strictness:** any invalid year, week range, hours-per-week string, unknown room in a fixed term, missing `@` in a fixed term, or a duplicate group name aborts the whole import at that line with the I/O error popup — unlike the row-tolerant imports (Rooms, Tutors, Exams), which only count warnings.
- Duplicate schedules are suppressed: if an identical turn part / day / start / duration / week range / room entry already exists, the rest of the line's terms are skipped.
- Rooms and (ideally) Tutors should be imported first; tutors given here are created on the fly, but rooms referenced in fixed terms must already exist.

## Data ▸ CSV import ▸ Reservations

**What it does.** Imports room bookings as reservations. The standard behaviour reads an external booking-system export (booking title, area, room, start/end timestamps) and creates one reservation per line, auto-creating the room. The alternative behaviour reads the application's own reservation layout, with groups, tutors, course, block type and an optional GUID that lets an existing reservation be updated instead of duplicated.

**How to start it.** `Data ▸ CSV import ▸ Reservations`

**File format.** Plain CSV, one line per booking. Lines shorter than 4 characters, lines starting with `;;` and lines starting with `#` are skipped. Separator and quoting as for Rooms.

Standard layout (alternative import switched off) — fixed 11-column booking export:

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Booking title | Mandatory; an empty value skips the line. A line containing `Booking title (short text)` is treated as the header and skipped. | `Board meeting` |
| 2 | Area | Only used to classify `Meeting Rooms`, `Video Conferencing`, `Event Spaces`; in all cases the room name is taken from column 3. | `Meeting Rooms` |
| 3 | Room | Room name. Created if unknown, with 40 seats and “show on web” enabled. | `MR-1` |
| 4 | Start time | Format `HH:MM:SS - DayName DD MonthName YYYY`; month names are English. Outside the school year the line is skipped. | `08:00:00 - Friday 01 September 2017` |
| 5 | End time | Same format. | `17:00:00 - Friday 01 September 2017` |
| 6 | Duration | Read but not used. | `9 hours` |
| 7 | Booking full description | Appended to the title as `title - description` (or used alone when identical). A `\` becomes a line break. | `Quarterly board meeting` |
| 8 | Type | Read but not used. | `Internal` |
| 9 | Created by | Read but not used. | `J. Smith` |
| 10 | Confirmation status | Read but not used. | `Confirmed` |
| 11 | Last updated | Read but not used. | `01/09/2017` |

Layout when the alternative import setting is on (column 1 is optional, so the numbering shifts by one when it is present):

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Day name | **Optional.** Only recognized when the value exactly equals one of the localized full day names. A line whose first field equals the localized “Day” header text is skipped. | `Friday` |
| 2 | Date | Parsed with the configured date format. May carry a `@weeks[/days]` suffix, e.g. `01.09.2017@4/2` (4 weeks, 2 days). | `01.09.2017` |
| 3 | Number of weeks | Present only when the date has no `@` suffix. | `1` |
| 4 | Number of days | Present only when the date has no `@` suffix. | `1` |
| 5 | Time range | `HH:MM-HH:MM`; `.`, `=`, `,` and `"` are normalised to `:` or space first. Duration is derived from the range and the configured segment length. | `08:00-10:00` |
| 6 | Groups | Comma-separated group names; anything in round brackets is blanked out before matching. Unknown groups are ignored. | `G1, G2` |
| 7 | Block type | **Optional.** Recognized only when it equals one of the six configured block colour names; otherwise this field is taken to be the course name and the type defaults to 1. | `Exams` |
| 8 | Course name | Matched against existing course names; no match leaves the reservation without a course. | `Databases` |

| # | Column | Description | Example |
|---|---|---|---|
| 9 | Rooms | Comma-separated room names; unknown rooms are ignored. | `A-001, A-002` |
| 10 | Tutors | Comma-separated, each `Surname FirstName` (exactly that order and one space). | `McClusky James` |
| 11 | Comment | `\` is converted to a line break. | `Guest lecture` |
| 12 | Unique id | **Optional.** A 36-character GUID. If it matches an existing reservation, that reservation is updated in place (rooms, groups, tutors and course are cleared first); otherwise a new GUID is generated. | `2b1f0e4e-…` |

**Sample**

```
Board meeting;Meeting Rooms;MR-1;08:00:00 - Friday 01 September 2017;17:00:00 - Friday 01 September 2017;9 hours;Quarterly board meeting;Internal;J. Smith;Confirmed;01/09/2017
```

Alternative layout:

```
01.09.2017@2/1;08:00-10:00;G1, G2;Exams;Databases;A-001;McClusky James;Written exam;2b1f0e4e-6b0f-4a2f-9c33-5d1a2b3c4d5e
```

**Notes.**

- In the standard layout the room is always created with 40 seats and “show on web” on, and the reservation type is fixed to 4; the start slot is derived from the clock time on a fixed half-hour grid offset by 7 slots, which assumes a timetable starting at 07:00.
- Bookings outside the configured school year (either endpoint) are skipped without a warning.
- In the alternative layout only the GUID column makes the import idempotent; without it, re-importing the same file duplicates every reservation.
- Groups, rooms, courses and tutors are **not** auto-created in the alternative layout — unresolved names are simply dropped, so import Rooms, Tutors, Courses and the group-producing imports first.

## Data ▸ CSV import ▸ Students

**What it does.** Imports students into the branch that is currently selected in the workspace. The menu handler passes the file handle, the selected branch and a flag saying the students should *not* be added to the on-screen table.

**How to start it.** `Data ▸ CSV import ▸ Students`

**File format.** The reader itself lives outside this source file, so its column layout cannot be stated from the import module. What the module does establish: the file is chosen with a `*.csv` dialog and opened read-only through the UTF-8 aware helper, like every other import on this menu.

**Notes.**

- **Depends on the workspace selection.** The branch passed to the import is whatever branch is selected in the workspace when the menu item is chosen; select the correct program, branch and year first.
- The students are not appended to the visible table during the import — the workspace is refreshed once afterwards.
- The alternative-import file-name step (`1.csv` … `6.csv`) is computed before this import runs but is not used by it.

- This is the older single-branch student import; the ini-driven one is `Data ▸ CSV import ▸ Students2`.

## Data ▸ CSV import ▸ Exams

**What it does.** Imports exam terms as reservations of type "exam". Each line is one exam: date, time span, course, room and up to five examiners. The course is resolved by name (with several fall-backs), matching groups can be attached automatically, and the examiners are added to the reservation and written into its comment.

**How to start it.** `Data ▸ CSV import ▸ Exams`

**File format.** Plain CSV, one line per exam. No header line is expected, but lines shorter than 4 characters and lines starting with `;`, `#` or `*` are skipped by the reader. Separator and quoting as for Rooms in the standard layout; the alternative layout hard-codes `;`.

Standard layout:

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Date | Parsed with the configured date format. Fall-back: `d.m` or `m/d` without a year, in which case the year is derived from the school-year start month. | `01.09.2017` |
| 2 | Start time | `HH:MM`. The nearest time slot is used as the start. | `09:00` |
| 3 | End time | `HH:MM`. Duration is the slot difference; if it is not positive the configured default exam reservation length is used. | `11:00` |
| 4 | Course name | Resolved against course names, with fall-backs that strip a trailing `(…)`, a trailing `EN`, or everything after a comma. If nothing matches, the reservation is created without a course. | `Databases` |
| 5 | Room name | Matched by the first part of the room name. **Must exist** — an unknown room fails the line. | `A-001` |
| 6 | Examiner 1 | Mandatory. Added to the reservation and to its comment. | `McClusky James` |
| 7 | Examiner 2 | Optional. | `Bergstrom Anna` |
| 8 | Examiner 3 | Optional. | |
| 9 | Examiner 4 | Optional. | |
| 10 | Examiner 5 | Optional. Special case: if this field starts with `#`, the text after the `#` replaces the entire generated comment and the field is not treated as a person. | |

Layout when the alternative import setting is on (separator is always `;`, at most 10 fields are read):

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Comment / exam title | Stored as the reservation comment. | `Databases - written exam` |
| 2 | Date | `d.m.y`. | `1.9.2017` |
| 3 | Start time | `HH:MM`; nearest time slot is used. Duration is fixed at 4 slots. | `09:00` |

| # | Column | Description | Example |
|---|---|---|---|
| 4 | Rooms | Comma-separated room names, created if unknown. If empty, a single room named `brez ucilnice` is used and created if missing. | `A-001, A-002` |
| 5 | Examiner | `FirstName Surname`; created if unknown. A value without a space is taken as the first name only. | `James McClusky` |
| 6 | Program name | Program is created if unknown. | `Undergraduate` |
| 7 | Branch name | Mandatory — an empty value skips the line. | `Business Computing and IT` |
| 8 | (ignored) | Read past, not interpreted. | |
| 9 | Year | 0 or non-numeric becomes 1; raises the program's year count. | `1` |
| 10 | (stop) | Parsing stops after 10 fields. | |

**Sample**

```
01.09.2017;09:00;11:00;Databases;A-001;McClusky James;Bergstrom Anna
15.09.2017;14:00;16:00;Mathematics I;B-102;Novak Peter
```

Alternative layout:

```
Databases - written exam;1.9.2017;09:00;A-001, A-002;James McClusky;Undergraduate;Business Computing and IT;;1
```

**Notes.**

- Header lines are recognized only in the alternative layout, where a line beginning `Predmet;dt` or `Course;dt` is skipped, as are lines starting with `;`, `#` or `'`.
- Advanced settings change the behaviour: the default exam reservation length, the default block type used for exam reservations (matched against the six configured block colour names), whether reservations in the same room/slot are merged instead of duplicated, whether matching groups are attached at all, and whether group attachment is filtered by a name prefix derived from the course title.
- When merging is enabled and a reservation already exists in that room and slot for the same course, the new exam is appended to the existing comment as an extra line instead of creating a second reservation.
- Each created reservation also stores an internal comment containing the source line number and the raw line, which makes tracing bad input easy.
- The alternative layout auto-creates a group named `<branch name>-<year>` with one student and attaches it to the reservation.
- Rooms must be imported before exams in the standard layout (unknown rooms fail the line); in the alternative layout rooms are created on the fly.
- Warnings are collected per line and the failing line numbers are listed in an error popup at the end; the import stops after 2047 warnings.

## Data ▸ CSV import ▸ Courses2

**What it does.** A configurable variant of the courses import: the column positions, the delimiter and the placeholder for "empty" are not fixed in the code but read from an ini file in the program data directory. Each line creates or updates program, branch, course, course part, turn and turn part, sets the week range, weekly duration, preferred room, no-pause flag, day/hour constraint and the

student count. It also records which execution types occur in which program and writes that relation to a temporary file used later by Groups2 and Students2.

**How to start it.** `Data ▸ CSV import ▸ Courses2`

**File format.** Plain CSV. The **first line is skipped as a header**. Lines starting with `;;;` are skipped, as are lines whose first field starts with `//`. The delimiter is *not* the global one: it is taken from the ini file for the duration of the line and restored afterwards. Exactly *number of columns* fields are read from every line, and any field equal to the configured placeholder is treated as empty.

The controlling ini file (`MISC_IMPORT_COURSES_FILE` in the program data directory) is read line by line, in this fixed order:

| Ini line | Meaning |
|---|---|
| 1 | title, skipped |
| 2 | number of columns per CSV line |
| 3 | delimiter character (first character of the line) |
| 4 | placeholder text meaning “empty” (cut at the first `;`) |
| 5 | column number of the program |
| 6 | column number of the year |
| 7 | column number of the first week |
| 8 | column number of the last week |
| 9 | column number of the fixed term |
| 10 | column number of the course name |
| 11 | column number of the execution type |
| 12 | column number of tutor 1 surname |
| 13 | column number of tutor 1 first name |
| 14 | column number of tutor 2 surname |
| 15 | column number of tutor 2 first name |
| 16 | column number of tutor 3 surname |
| 17 | column number of tutor 3 first name |
| 18 | column number of the weekly duration |
| 19 | column number of the room |
| 20 | column number of the number of students |
| 21 | column number of the “no pauses” flag |

All column numbers in the ini file are **1-based**. The resulting CSV fields are:

| Field | Description | Example |
|---|---|---|
| Program | Program name; created if unknown. Also used as the branch name. | `Undergraduate` |
| Year | 1 up to the configured maximum, otherwise the import aborts. | `1` |
| First week | 1–52, otherwise the import aborts. | `1` |
| Last week | 1–52, otherwise the import aborts. | `15` |
| Fixed term | Optional, `day:starttime`. The leading number is the day (1–7), the text after the `:` must match a time-slot start time exactly. When present, the turn is restricted to that single day. | `2:08:00` |

| Field | Description | Example |
|---|---|---|
| Course name | Mandatory; empty aborts the import. Course is created for the branch if unknown. | `Databases` |
| Execution type | Mandatory; empty aborts the import. Created as a new execution type when unknown. | `Lecture` |
| Tutor 1 surname / first name | Tutor created if unknown. A tutor is only added when the surname field is non-empty. | `McClusky` / `James` |
| Tutor 2 surname / first name | Optional. | |
| Tutor 3 surname / first name | Optional. | |
| Weekly duration | Duration string inserted for the whole first-week..last-week period. | `2` |
| Room | Optional. Set as the preferred room; created if unknown. | `A-001` |
| Number of students | Applied to the turn part only when the turn had no turn parts yet; marks the count as custom. | `30` |
| No pauses | Integer flag on the turn. | `0` |

**Sample** (assuming an ini that declares 8 columns with `;` as delimiter, in the order program, year, first week, last week, course, type, tutor surname, tutor first name)

```
Program;Year;FirstWeek;LastWeek;Course;Type;Surname;FirstName
Undergraduate;1;1;15;Databases;Lecture;McClusky;James
Undergraduate;1;1;15;Databases;Tutorial;Bergstrom;Anna
```

**Notes.**

- **Ordering requirement:** Courses2 must run **before** Groups2 and Students2. It writes the program-to-execution-type relation into the temporary file (`MISC_IMPORT_TEMP_FILE` in the program data directory), and both of those imports refuse to run if that file is missing.
- The ini file must exist; if it cannot be opened the import stops immediately with the I/O error popup.
- Turns are reused when the same set of tutors is found on an existing turn of that course part; otherwise a new turn is created. A turn part is only created when the turn has none, so re-importing does not multiply turn parts.
- **Changed in this revision.** The day-restriction code cleared the day flags with a logical not instead of a bitwise complement, which wiped the turn's whole flags word instead of one day bit. A fixed term now clears only the day it is meant to clear.
- **Alternative import.** When the *alternative import* setting is on, the ini file is *not* read and a completely different, grid-shaped file is parsed instead, with these line types: a line containing `;;;;` together with `Teden`/`teden` sets the default week range from its leading number and the number after `-`; a line whose first field contains a day name (`PONEDELJEK`, `TOREK`, `SREDA`, `ČETRTEK`/`CETRTEK`, `PETEK`, `SOBOTA`, `NEDELJA`) sets the current day; a line whose first field is not numeric is ignored. Data lines have 5 columns: 1 time range `H:MM-H:MM` (dots are read as colons; slots are quarter-hours counted from 07:00), 2 course name (trailing `*` and spaces stripped), 3 comma-separated tutors as `FirstName Surname` (an empty field becomes `N N`), 4 room name (created with 32 seats if unknown), 5 group name — the group **must already exist**, otherwise the line is skipped and a message is printed. The default week range

can also be pre-seeded from a file named `defaultweeks.txt` (`first-last`) in the working directory, defaulting to 1–16. In this mode nothing is written to the temporary relation file.

## Data ▸ CSV import ▸ Groups2

**What it does.** Assigns groups to the turn parts of a course. Each line names a group, its size, its program, its year and up to two alternative course names; the group is created if unknown, bound to the matching branch, and then allocated to the turn part with the fewest groups in every course part of that course whose execution type is registered for that program.

**How to start it.** `Data ▸ CSV import ▸ Groups2`

**File format.** Plain CSV. The **first line is skipped as a header**; empty lines are skipped, and a line whose first field starts with `//` is treated as a comment. Delimiter, column positions and the placeholder come from an ini file, exactly as for Courses2 — the global separator is temporarily replaced by the one from the ini file. Exactly *number of columns* fields are read per line.

Controlling ini file (`MISC_IMPORT_GROUPS_FILE` in the program data directory):

| Ini line | Meaning |
|---|---|
| 1 | title, skipped |
| 2 | number of columns per CSV line |
| 3 | delimiter character |
| 4 | placeholder text meaning “empty” (cut at the first `;`) |
| 5 | column number of the group name |
| 6 | column number of the number of students |
| 7 | column number of the program |
| 8 | column number of the year |
| 9 | column number of the course name |
| 10 | column number of the alternative course name |
| 11 | ignore list — course names in double quotes; a line whose course matches is skipped |
| 12 | skipped |
| 13.. | programs dictionary: `sourceName"targetName"` entries, terminated by a line containing `Courses dictionary:` |
| after | courses dictionary: `sourceName,"targetName"` entries |

Column numbers in the ini file are **1-based**. CSV fields:

| Field | Description | Example |
|---|---|---|
| Group name | Key; group is created if unknown, otherwise re-bound to the resolved branch. | `G1` |
| Number of students | Written to the group. | `30` |
| Program | Translated through the programs dictionary if listed. The program **must already exist**, otherwise the line is skipped. | `Undergraduate` |
| Year | 1–9. `0` or non-numeric skips the line silently; greater than 9 aborts the import. | `1` |
| Course name | Translated through the courses dictionary if listed. | `Databases` |

| Field | Description | Example |
|---|---|---|
| Alternative course name | Second chance for the course lookup; either name may match. Also checked against the ignore list. | `DB` |

**Sample**

```
GroupName;Students;Program;Year;Course;CourseAlt
G1;30;Undergraduate;1;Databases;DB
G2;28;Undergraduate;1;Databases;DB
```

**Notes.**

- **Ordering requirement:** run **after** Courses2. Groups2 reads the temporary relation file written by Courses2; if it is missing the import stops with the I/O error popup. The relation decides which execution types of a course a group is allocated into — course parts whose type is not registered for that program are skipped.
- The course must already exist and must be linked to a branch of the matching program and year, otherwise the line is skipped (logged) or, when the course has no branches at all, the whole import aborts.
- Allocation is "fill the emptiest": the group is added to the turn part that currently holds the fewest groups. A group already present anywhere in that course part is not added again, so re-importing is safe.
- The ini file must exist; a missing ini file stops the import immediately.

## Data ▸ CSV import ▸ Students2

**What it does.** The individual-student variant of Groups2: every student becomes a one-person group identified by the student id, and is allocated into the turn parts of the courses listed for them. When the emptiest turn part is already full (within a configurable overbooking percentage), the import splits the turn with the fewest turn parts into two and moves half of the students across. At the end the "actual number of students" of every custom-sized turn part is set to the number of allocated groups.

**How to start it.** `Data ▸ CSV import ▸ Students2`

**File format.** Plain CSV, one line per student and course. The **first line is skipped as a header**; empty lines are skipped and a line whose first field starts with `//` is a comment. Delimiter, column positions and the placeholder come from an ini file; the global separator is temporarily replaced by it. Exactly *number of columns* fields are read per line.

Controlling ini file (`MISC_IMPORT_STUDENTS_FILE` in the program data directory):

| Ini line | Meaning |
|---|---|
| 1 | title, skipped |
| 2 | number of columns per CSV line |
| 3 | delimiter character |
| 4 | placeholder text meaning "empty" (cut at the first `;`) |
| 5 | new-turn percentage: how far above the nominal turn-part size a turn part may be filled before a new one is created |
| 6 | column number of the surname |
| 7 | column number of the first name |
| 8 | column number of the student id |

| Ini line | Meaning |
|---|---|
| 9 | column number of the program |
| 10 | column number of the year |
| 11 | column number of the course name |
| 12 | column number of the alternative course name |
| 13 | ignore list — course names in double quotes |
| 14 | skipped |
| 15.. | programs dictionary, terminated by a line containing `Courses dictionary:` |
| after | courses dictionary |

Column numbers in the ini file are **1-based**. CSV fields:

| Field | Description | Example |
|---|---|---|
| Surname | Stored together with the first name in the group note. | `Smith` |
| First name | Stored in the group note as `Surname FirstName`. | `John` |
| Student id | Key. Used as the group name; a one-student group is created when unknown. | `63120345` |
| Program | Translated through the programs dictionary if listed. Must already exist, otherwise the line is skipped. | `Undergraduate` |
| Year | 1–9. `0` or non-numeric skips the line silently; greater than 9 aborts the import. | `1` |
| Course name | Translated through the courses dictionary if listed. | `Databases` |
| Alternative course name | Second chance for the course lookup; also checked against the ignore list. | `DB` |

**Sample**

```
Surname;FirstName;StudentId;Program;Year;Course;CourseAlt
Smith;John;63120345;Undergraduate;1;Databases;DB
Brown;Alice;63120346;Undergraduate;1;Databases;DB
```

**Notes.**

- **Ordering requirement:** run **after** Courses2 (and normally after Groups2). It reads the same temporary relation file that Courses2 writes; a missing file stops the import.
- Execution-type names can be scoped: a course-part type whose name looks like `X_Program` or `X_Program_Year` (second character is `_`) is only used for the matching program and, when a year is embedded, for the matching year. All other course parts of that course are skipped for this student.
- **Turn splitting:** when the emptiest turn part would exceed its actual student count plus the configured percentage, the turn with the fewest turn parts gets a new turn part and half of its first turn part's students are moved into it, then the current student is added to the original one. **Changed in this revision:** the branch used the chosen turn part without a null check, so a course part with no turns at all crashed the import; such course parts are now skipped.
- The student's line is repeated once per course; each line only allocates that one course.
- A missing ini file stops the import immediately with the I/O error popup.

## Exam data (button, not on the CSV import menu)

**What it does.** Marks courses and course parts as requiring a final exam and assigns their chief examiner. Every line lists one or more course references and ends with the tutor code of the examiner; for a plain course reference the flag and chief teacher are set on the course, for a reference that names an execution type they are set on that course part.

**How to start it.** It is not on the `Data ▸ CSV import` menu. It is a control callback that fires when a button is pressed; pressing that button opens the standard `*.csv` file dialog and runs the import. The source gives no panel or control identifier, so the exact dialog cannot be named from the code — the completion popup labels it with the "exam data" resource string.

**File format.** Plain CSV, one line per examiner assignment. No header line is expected; lines shorter than 3 characters and lines without at least one separator are rejected. The separator is the globally configured one. The number of separators on the line decides the split: with N separators, the first N fields are course references and the last field is the tutor code.

| # | Column | Description | Example |
|---|---|---|---|
| 1..n-1 | Course reference | Either `CODE`, `NAME`, `CODE (TYPE)` or `NAME (TYPE)`. The part before the bracket is looked up first as a course code, then as a course name; the text inside the brackets is looked up as an execution-type code and then as an execution-type name. A single space before the opening bracket is removed. Unresolvable references are logged and skipped. | `DB1 (Lecture)` |
| n | Tutor code | Examiner. A leading `P` is stripped before the lookup. The tutor must exist, otherwise the line is rejected. | `PT0123` |

**Sample**

```
DB1 (Lecture);DB1 (Tutorial);MATH101;PT0123
PHYS200;T0124
```

**Notes.**

- A reference without brackets sets the chief teacher and the "needs final exam" flag on the **course**; a reference with brackets sets them on the matching **course part**.
- **Changed in this revision.** The execution-type code used to be fetched with the course part's *id* instead of its *type*, so the code-based match ran against an unrelated property and only the name-based fallback worked. Referring to execution types by code is now reliable.
- Courses and tutors must already be imported. Failing lines are collected and their line numbers are listed in an error popup at the end; the import stops after 2047 warnings.

# Part 2 — Custom import codes

Sixty-seven codes, reached through `Data ▸ Custom import`. Rather than listing them alphabetically, this part groups them by **what the file contains**, so that the right code can be found from the data at hand:

| Chapter | Use it when the file describes |
|---|---|
| Programmes, branches and groups | Study programmes, specialisations, years, student groups |
| Rooms | Rooms, their capacity, their equipment, their external codes |
| Lecturers | Teaching staff, their codes, e-mail addresses, passwords, availability |
| Students and enrolments | Students, which group they belong to, which courses they take |
| Courses and teaching load | Courses and how many hours of what kind are taught — but not when |
| Placed timetables | Lessons that already have a day, a time and a room |
| Bookings and exams | Room bookings, external commitments, examination terms |

Codes that share one reader are documented together and both names appear in the heading. **Appendix B** maps every single code to its section alphabetically, and **Appendix A** lists the chains where one import has to run before another.

## Choosing between two codes that look alike

Several codes read very similar files. These are the distinctions worth knowing before picking one:

| If you are unsure between | The difference |
|---|---|
| `STUDBYCRS` / `STUDBYCRS2` / `STUDBYCRS3` | Names only, programme columns only, or both. The column positions shift — a file for one is silently imported as nothing by another |
| `ENRLS` / `ENRLST` | E-mail first with 8 fixed course slots, or student number first with a free number of course columns |
| `ENRLSD` / `ENRLSTD` | Both clear the student list first, each for its own file layout (`ENRLSD` did nothing before this revision) |
| `PROGCAT` / `PROGCATPLUS` / `PROGCATCRS` | Whether the course web link is read, and whether the course name is rewritten |
| `CRSHOURS` / `CRSHOURSBR` | Whether the branch is taken from the file (two extra columns) or from the programme name |
| `MODGRPM` / `MODGRPMI` / `MODGRPB` | Intake stream, and whether the line is laid out around the course or around the lecturer |
| `FULLTT` / `FULLTTSTART` | `FULLTTSTART` only for the first file of a session — it resets the “already cleaned” list |
| `DBBOOKINGS` / `DBBOOKINGSDEL` | `DBBOOKINGSDEL` deletes **every** reservation in the database first |
| `MODGRPA` / `AVAIL` | Identical — two names for the same reader |

## Programmes, branches and groups

Run these first. Everything else in this part looks up a programme, a branch or a group by name or by code, and creates nothing when the lookup fails. ## ENRLP — Programs, branches and groups

**What it does.** Creates the academic structure. Each line names a department (which becomes the program) and a study program/branch inside it, together with a code and the number of study years. For every year from 1 up to that number the import creates the branch (with the given code) and a student group named after the branch, with a default size of 30 students. The program is created if it does not exist and its number of years is raised when necessary.

**How to start it.** `Data ▸ Custom import`, then type `ENRLP` at the prompt and pick the file.

**File format.** Semicolon (`;`) separated, fixed separator, no quoting. UTF-8 transcoded on read; `.xls`/`.xlsx` is converted first. Line 1 is a header line and is skipped. One line = one branch (with all its years).

| # | Column | Description | Example |
|---|---|---|---|
| 1 | (ignored) | Ignored by the import; may hold a running number or be empty. | `1` |
| 2 | Department | Program name. **May be empty** — the value from the previous line is then reused, so a department only needs to be written once at the top of its block. | `Faculty of Engineering` |
| 3 | Branch name | Study program / branch name. Must contain a space for the group name to be derived; if it is empty, nothing is created. | `Computer Science` |
| 4 | Branch code | Code stored on the branch. May be empty. | `CS` |
| 5 | Years | Number of study years; `0`, empty or non-numeric is treated as 1. | `3` |

**Sample**

```
No;Department;Program;Code;Years
1;Faculty of Engineering;Computer Science;CS;3
2;;Software Engineering;SE;4
```

**Notes.**

- Does not depend on the workspace selection.
- This is the import that creates everything the other BIUST imports look up, so it must be run **first**.
- Group names are `<first word of branch name>-<capital letters of the remaining words>-L<year>` — `Computer Science` with 3 years produces `Computer-S-L1`, `Computer-S-L2`, `Computer-S-L3`. This is exactly the name `ENRLC`, `ENRLE`, `ENRLS` and `ENRLST` build from their program column, so the branch names used there must be written identically.
- A branch name without a space produces no group for that year (the branch itself is still created).
- Branch and program translations are cleared on every run; existing branches and groups are reused, so re-running the file is safe and does not duplicate anything. Group sizes already edited in the application are preserved (the 30-student default is only applied when the group is created).

## BRANCHCODE — Assign codes to existing branches by database ID

**What it does.** The branch counterpart of `PROGRAMCODE`. It sets the code of branches (specialisations) that already exist, addressing each one by its numeric database ID. No branch is created and no other branch field is touched.

**How to start it.** `Data ▸ Custom import`, then type `BRANCHCODE` at the prompt and pick the file.

**File format.** Semicolon-separated text, always `;` regardless of the configured CSV separator; no quote handling. UTF-8 or legacy ANSI. No header line is expected — a header line would be read as ID `0` and trigger the "does not exist" popup. One line = one branch. Any columns after the second are ignored.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Branch ID | Numeric internal database ID of the branch. Non-numeric text is read as `0`. | `3307` |
| 2 | Code | The branch code to store; surrounding whitespace is trimmed. | `SE` |

**Sample**

```
3307;SE
3308;AI
3311;DS
```

**Notes.**

- Same modal-popup behaviour as `PROGRAMCODE`: one dialog per unknown ID. Both the crashing format string and the code buffer that was not cleared between lines were **changed in this revision**.
- The code is copied straight into the branch's fixed-size code field — keep codes short.
- Lines are never reported as failures; the warning count in the completion message is not a quality check for this import.
- The file addresses branches directly by ID, so the workspace selection has no effect.

## MEDPROG — Medical programmes, areas and courses

**What it does.** Builds the programme skeleton for the medical imports: for each line it provides the programme, extends its number of years if needed, provides the area (branch) for that programme and year, and provides the course inside that branch, splitting an optional course code out of the course name. Every course created or found is marked as modular. It is the first import of the MED family, because `MEDCLASS` later resolves courses by their code.

**How to start it.** `Data ▸ Custom import`, then type `MEDPROG` at the prompt and pick the file.

**File format.** Semicolon-separated (`;`), one course-in-a-programme-year per line. No header line is expected and none is skipped — a header row would be treated as data, so remove it before importing. Fields after the fourth are ignored.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Programme name | Programme is looked up by name and created when missing. | `Pre-clinical Medicine` |
| 2 | Year | Study year as an integer. The programme's year count is raised to this value when it is larger. | `1` |
| 3 | Area (branch) name | Optional, may be empty. A surrounding pair of parentheses is stripped. When empty, the branch is | `(Cardiovascular)` |

| # | Column | Description | Example |
|---|---|---|---|
| | | named `<programme name> - <year>`. | |
| 4 | Course name | May carry a code in the form `Name (CODE)`: the text after the last `(` up to `)` becomes the course code and the rest becomes the name. Without `(` the code stays empty. | `Cardiovascular System (CVS101)` |

**Sample**

```
Pre-clinical Medicine;1;(Cardiovascular);Cardiovascular System (CVS101)
Pre-clinical Medicine;1;;Introduction to Anatomy (ANAT100)
Pre-clinical Medicine and Biomedical Sciences;2;(Neuroscience);Neuroscience (NEU201)
```

**Notes.**

- **Special programme name:** a line whose programme is exactly `Pre-clinical Medicine and Biomedical Sciences` is processed twice — first the programme is renamed to `Pre-clinical Medicine`, then the whole programme/branch/course block is repeated for the programme `Biomedical Sciences`, so the same course lands in both programmes.
- **Changed in this revision.** The second pass used to work on a course name the first pass had already truncated at `(`, so the `Biomedical Sciences` copy of the course was created with an **empty** code — and `MEDCLASS` could then never attach lectures to it. The original text is now kept and the code is derived again for the second programme.
- Programmes, branches and courses are all resolve-or-create. Nothing is deleted first, so re-running only re-provides the same entities.
- **Ordering:** run `MEDPROG` before `MEDCLASS`. `MEDCLASS` finds courses only by the code produced here, so a course imported without a code cannot receive lectures.
- No schedule, turn or group is created here. Groups for the branches must be created separately — `MEDCLASS` never creates them.

## PROGCAT / PROGCATPLUS / PROGCATCRS — Programme, branch, group and course catalogue

**What it does.** Builds the academic structure from a catalogue export: it creates or updates the programme (with its code), creates a branch per study year, creates the standard group for each branch/year, and creates the course, linking it to every branch listed in the years column. The three codes are the same reader with three variations that differ only in whether the course web link is read and whether the course name is rewritten.

**How to start it.** `Data ▸ Custom import`, then type `PROGCAT` (or `PROGCATPLUS`, or `PROGCATCRS`) at the prompt and pick the file.

**File format.** Semicolon (`;`) separated, 9 columns, one course-in-a-branch per line. A header line is expected and recognised only when its first cell is exactly `degreePartName`; besides that, lines shorter than 10 characters, lines starting with `#` and lines whose years cell does not start with a digit are skipped. Columns 4 and 8 are read past but never used, so they must be present as placeholders.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Branch name | Branch (degree part). If empty, the programme name from column 2 is used as the branch name. | `Computer Science` |
| 2 | Programme name | Programme; created if it does not exist. | `Computer Science` |
| 3 | Programme code | Written into the programme's code field and used as the prefix of the generated group names. | `CS` |

| # | Column | Description | Example |
|---|---|---|---|
| 4 | (ignored) | Placeholder, any value. | |
| 5 | Course name | Course title. For `PROGCAT` and `PROGCATPLUS` a leading `N.N` numbering is moved to the end in brackets (see Notes). | `1.2 Machine Learning` |
| 6 | Course code | Course code. | `CS-ML` |
| 7 | Course web link | Stored as the course document link. Read by `PROGCATPLUS` and `PROGCATCRS` only; ignored by `PROGCAT`. May be empty. | `https://example.edu/ml` |
| 8 | (ignored) | Placeholder, any value. | |
| 9 | Years | A digit string; **every digit** is treated as one study year (`23` means years 2 and 3). A non-numeric or `0` value skips the line. | `23` |

**Sample**

```
degreePartName;degreeName;degreeCode;;courseName;courseCode;courseURL;;years
Computer Science;Computer Science;CS;;1.2 Machine Learning;CS-ML;https://example.edu/ml;;23
Computer Science and Philosophy;Computer Science;CS;;2.1
Algorithms;CS-ALG;https://example.edu/alg;;3
```

**Notes.**

- **Variations.** `PROGCAT` ignores column 7 and applies the course-name rewrite; `PROGCATPLUS` reads column 7 into the course link and applies the rewrite; `PROGCATCRS` reads column 7 but leaves the course name exactly as written. Column 7 is trimmed of surrounding whitespace only in `PROGCATPLUS` — with `PROGCATCRS` a link with a leading or trailing space is stored as-is.
- **Course-name rewrite** (`PROGCAT` and `PROGCATPLUS`): if the first 8 characters of the name contain a dot with digits on both sides, the numeric prefix is cut off and appended in brackets, so `1.2 Machine Learning` becomes `Machine Learning (1.2)`. **Changed in this revision:** a name containing no space at all used to crash the rewrite; such a name is now left exactly as written. Keeping a space after the numbering is still the right form.
- **Auto-created:** programmes, branches (one per digit in column 9), courses, and groups. Generated group name = programme code + `-` + all upper-case letters and digits of the branch name + `-Y<year>`; e.g. branch `Computer Science and Philosophy`, code `CS`, year 3 gives `CS-CSP-Y3`. New groups get 30 students.
- The programme's number of years is raised to the highest digit seen in column 9.
- Nothing is deleted first, so re-running the file updates/extends the structure rather than replacing it.
- Does not depend on the workspace selection.

## PROGRAMCODE — Assign codes to existing programs by database ID

**What it does.** Sets the short code of study programs that already exist, addressing each one by its numeric database ID. It is a bulk code-assignment tool, typically used after programs were created without codes, or to align codes with an external student information system. No program is created and nothing else on the program is changed.

**How to start it.** `Data ▸ Custom import`, then type `PROGRAMCODE` at the prompt and pick the file.

**File format.** Semicolon-separated text, always `;` and not the configured CSV separator; no quote handling. UTF-8 or legacy ANSI. No header line is expected — a header would be treated as ID `0` and

trigger the “does not exist” popup. One line = one program. Any columns after the second are ignored.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Program ID | Numeric internal database ID of the program. Non-numeric text is read as `0`. | `1024` |
| 2 | Code | The program code to store; surrounding whitespace is trimmed. | `BSC-CS` |

**Sample**

```
1024;BSC-CS
1025;MSC-CS
1031;BSC-MATH
```

**Notes.**

- Every unknown ID pops up a modal message box that must be acknowledged before the import continues — one dialog per bad line. Validate the ID list before running the file.
- **Changed in this revision.** That “Program with ID … does not exist” message formatted the numeric ID with a string placeholder, so on an unknown ID the text was garbage and the application could crash. It prints the ID correctly now.
- **Changed in this revision.** The code buffer was not cleared between lines, so a line with no `;` at all re-assigned the previous line’s code.
- Lines are never reported as failures — the summary always shows them as read.
- The file addresses programs directly by ID, so the workspace selection has no effect.

## Rooms

Rooms and their equipment. Several of the timetable imports match rooms by **code** rather than by name, so the code-setting imports here have to run before them. ## ENRLR / ENRLRE — Rooms with capacity and equipment

**What it does.** Creates or updates rooms. Each line gives a room name, its seating capacity and one or two equipment lists; the room is created when the name is unknown, its capacity is set, and every listed equipment item is attached as a room property. Room properties that do not yet exist are created automatically. `ENRLRE` additionally marks every imported room as an exam room.

**How to start it.** `Data ▸ Custom import`, then type `ENRLR` (or `ENRLRE`) at the prompt and pick the file.

**File format.** Semicolon (`;`) separated, fixed separator, no quoting. UTF-8 transcoded on read; `.xls`/`.xlsx` is converted first. No header line is skipped by position; header rows are recognised by value (see Notes). One line = one room.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Room name | Room name; matched exactly against existing rooms, otherwise a new room is created. Empty lines are ignored. | `A101` |
| 2 | Equipment | Comma-separated list of equipment / room properties. May be empty. In `ENRLR` the single value `skip` makes the line be ignored. | `projector,whiteboard` |
| 3 | Capacity | Number of seats; only applied when it parses to a value above 0. May be empty. | `120` |
| 4 | Additional equipment | A second comma-separated equipment list, appended to column 2. May be empty. | `computer lab` |

**Sample**

```
Room Name;Equipment;capacity;Additional equipment
A101;projector,whiteboard;120;
B204;computers,projector;36;computer lab
```

**Notes.**

- Does not depend on the workspace selection.
- `ENRLRE` starts the equipment list with the property `exam room`, so every room in the file receives that property in addition to the ones listed in columns 2 and 4. This is the list `ENRLE` relies on.
- Rooms **and** room properties are auto-created. Property names are matched case-sensitively and exactly, so `Projector` and `projector` become two different properties — keep the spelling consistent across the whole file and with the property names configured for `ENRLC` and `ENRLE`.
- Header rows are detected by content: the line is skipped when column 1 is exactly `Room Name`, `ROOM NAMING IN USE` or `Name`, or when column 3 starts with `capacity`.
- The combined equipment text must be at least 2 characters long, otherwise no property is attached — a single-character property name is silently dropped.
- The `skip` marker only works for `ENRLR`; in `ENRLRE` the equipment string already starts with `exam room`, so `skip` is appended instead of matching and the line is imported.
- Existing rooms keep every property they already have; this import only adds. Capacity is overwritten when column 3 is a positive number.
- **Ordering:** run before `ENRLC` and `ENRLE`, which look up rooms and room properties by name.

## EXTROOMIDS — Room external IDs

**What it does.** Attaches external system identifiers to rooms that already exist. Each line is matched against the room list by **room name** (column 1, compared after trimming spaces on both sides), and the identifier from column 3 is stored in that room's source ID field. Only the first matching room is updated. Nothing is created and no other room property is modified.

**How to start it.** `Data ▸ Custom import`, then type `EXTROOMIDS` at the prompt and pick the file.

**File format.** Semicolon (`;`) separated, one room per line. No header line is expected. Quoting is not supported. UTF-8 preferred; `.xls`/`.xlsx` are converted to CSV first. Leading and trailing spaces are removed from every field. Lines shorter than 2 characters are skipped silently.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Room identifier | Despite being an external "code" column, it is compared against the **room name**. Must match exactly (apart from surrounding spaces). | `Lecture hall A` |
| 2 | Description | Read but not used. May be empty. | `Main building, ground floor` |
| 3 | Source ID | External identifier written into the room's source ID field. | `4711` |

**Sample**

```
Lecture hall A;Main building ground floor;4711
Computer lab B;Annex, 24 workstations;4712
Seminar room 3;Library wing;4713
```

**Notes.**

- Nothing is auto-created. A line whose column 1 matches no room name is silently ignored — it is not reported as a warning, so the run can finish with "0 warnings" while having updated nothing.
- The source ID is a separate field from the room code used by `MODCRSEVENTS`; this import does not set the room code. Use `ROOMCODES` for codes.
- **Changed in this revision.** The field buffers were reused between lines, so a line with fewer than 3 columns stamped the previous line's source ID onto this room. Such a line is now skipped instead.
- Independent of the workspace selection.

## MEDVENUE — Medical venues (rooms)

**What it does.** Creates rooms from a venue list. For each line it takes the numeric venue id, the room code and the room name, and adds a new room only when no room with that name exists yet. New rooms get the id and code from the file and a default capacity of 30 seats.

**How to start it.** `Data ▸ Custom import`, then type `MEDVENUE` at the prompt and pick the file.

**File format.** Semicolon-separated (`;`), one room per line. No header line is expected. At least two separators must be present; the room name is everything after the second separator, so the name itself may contain further `;` characters. Only the room name is trimmed of surrounding whitespace.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Venue id | Integer; becomes the room id of a newly created room. | `12` |
| 2 | Room code | Short code, stored on a newly created room. Used later by `MEDCLASS` to resolve the room. | `LT1` |

| # | Column | Description | Example |
|---|---|---|---|
| 3 | Room name | Rest of the line. Used to decide whether the room already exists. | `Main Building Lecture Theatre 1` |

**Sample**

```
12;LT1;Main Building Lecture Theatre 1
13;SR2;Seminar Room 2
27;ANAT;Anatomy Dissection Room
```

**Notes.**

- Existing rooms are matched by **name** only and are then left untouched: their id, code and capacity are not updated. A room that already exists under a slightly different name will be created a second time.
- Capacity is always 30 for created rooms; correct it afterwards if it matters for the placement.
- The id from column 1 is written directly into the created room. Make sure these ids do not clash with rooms already in the base.
- **Changed in this revision.** A line without at least one `;` left the parsing buffers untouched and the previous line's values were reused; the buffers are cleared per line now and a line without a venue name is skipped.
- **Ordering:** run `MEDVENUE` before `MEDCLASS`.

## ROOMCODES — Room codes

**What it does.** Reads a list of rooms with their external codes and reconciles them with the rooms in the open schedule. A room whose name already exists is updated in place; a room whose name is not found is created automatically with a default capacity of 30 seats. In both cases the external code from column 2 is written into the room's code field, which is what `MODCRSEVENTS` later uses to resolve rooms. Finally the room name is cleaned: a trailing `, <number>` suffix at the very end of the name is cut off.

**How to start it.** `Data ▸ Custom import`, then type `ROOMCODES` at the prompt and pick the file.

**File format.** Semicolon-separated (`;`), one room per line. No header line is expected — a header would be imported as if it were a room. Quoting is not supported, so room names must not contain a semicolon. UTF-8 preferred; `.xls`/`.xlsx` may be selected and are converted to CSV first. Leading and trailing spaces are removed from every field. Lines shorter than 2 characters are skipped silently.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Room name | Name of the room as it is (or will be) in the application. Matched against existing room names; if no match, a new room is created with 30 seats. | `Lecture hall A` |
| 2 | Room code | External room code, stored in the room's code field. May be empty, but then the room cannot be referenced by `MODCRSEVENTS`. | `LH-A` |

**Sample**

```
Lecture hall A;LH-A
Computer lab B, 24;LAB-B
Seminar room 3;SR-03
```

**Notes.**

- Nothing else is touched: capacity is set only for newly created rooms (30); existing rooms keep their capacity, properties and equipment.

- **Name clean-up rule:** after the room is found or created, the name is scanned for a comma followed by a number greater than 0 that sits within the last 4 characters of the name; the name is then truncated at that comma. In the sample above `Computer lab B, 24` is stored as `Computer lab B`.
- **Trap — re-running the import:** because the stored name has already been truncated, a second run of the same file no longer matches by name and creates a duplicate room. Either strip the `, <number>` suffix from the file before the first run, or run this import only once per room list.
- **Changed in this revision.** The field buffers were reused between lines, so a line containing only the room name stamped the previous line's code onto this room. The buffers are cleared per line and an empty code no longer overwrites a good one.
- **Ordering:** run this import before `MODCRSEVENTS`, so that every room referenced by an event already has its code.
- Independent of the workspace selection.

## ROOMSCAP — Rooms with code and capacity

**What it does.** Loads a plain room list: for each line it looks up the room by name and creates it if it does not exist, then sets its capacity and stores the external identifier from column 1 as the room code. Nothing else on the room is touched, so re-running the import is safe and only refreshes code and capacity.

**How to start it.** `Data ▸ Custom import`, then type `ROOMSCAP` at the prompt and pick the file.

**File format.** Semicolon-separated; the separator is fixed. UTF-8 or legacy ANSI. A header line does not need to be removed — every line whose first column is not a non-zero number is silently skipped. One line = one room. Columns after the third are ignored.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Room ID | Must be a non-zero number, otherwise the line is skipped. Stored as the room code. | `101` |
| 2 | Room name | Looked up among existing rooms; created when not found. | `Lecture room A1` |
| 3 | Seats | Integer, stored as room capacity. Empty or non-numeric gives 0. | `60` |

**Sample**

```
ID;NAME;SEATS
101;Lecture room A1;60
102;Computer lab B2;24
```

**Notes.**

- Rooms are matched by **exact name**; a renamed room in the source file produces a second room instead of an update.
- The room code is taken from column 1, not from any later column.
- Independent of the workspace selection.

## ROOMTRANS — Import translated room names

**What it does.** Fills the translation (second-language name) field of existing rooms. Each line pairs a room, identified either by its code or by its name, with the translated name. Rooms are never created; a line for an unknown room is discarded.

**How to start it.** `Data ▸ Custom import`, then type `ROOMTRANS` at the prompt and pick the file.

**File format.** Semicolon-separated text, always `;` and not the configured CSV separator; no quote handling, so the translated name must not contain `;`. Read through the UTF-8 reader. No header line is expected. One line = one room. Lines shorter than 3 characters and lines starting with `;` or `'` are skipped.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Room code or room name | Matched case-insensitively: the code is tried first, then the name, for each room in turn; the first room that matches either wins. | `A-101` |
| 2 | Translated room name | Stored in the room's translation field. Everything after the first `;` belongs to this field. | `Large Lecture Hall` |

**Sample**

```
'room code or name;translation
A-101;Large Lecture Hall
LAB-CHEM-2;Chemistry Laboratory 2
```

**Notes.**

- **Changed in this revision.** Both fields were trimmed *before* they were filled, so the trim did nothing and a space around the `;` became part of the stored translation. Both fields are trimmed properly now.
- Rooms that do not match anything are skipped silently, with no warning. Because a code and a name can collide across different rooms, check the result on a few rooms after the import.
- The translation is written into a fixed-size field; keep translated names to a normal room-name length.
- Independent of the workspace selection.

## Lecturers

Lecturer records, their codes, e-mail addresses, passwords and weekly availability. Note how differently these imports match a person: by name, by surname alone, by code, or ignoring academic titles altogether. ## MODGRPA / AVAIL — Lecturer weekly availability

**What it does.** Replaces the weekly availability ("wanted") reservations of the listed lecturers. For each line it finds the lecturer by code, deletes every existing wanted reservation that lecturer is attached to, and then creates one reservation per time range found in the seven day columns, spanning weeks 1 to 52. `MODGRPA` and `AVAIL` are two names for exactly the same reader — there is no difference in behaviour. The file has the same layout as the availability export produced by the application, so an exported file can be edited and re-imported.

**How to start it.** `Data ▸ Custom import`, then type `MODGRPA` (or `AVAIL`) at the prompt and pick the file.

**File format.** Semicolon (`;`) separated, one line per lecturer. The separator is fixed and quoting is not supported. Read through the UTF-8 reader that also accepts a legacy ANSI export; `.xls`/`.xlsx` files are converted to CSV first. The header line written by the export (`Lecturer ID;Lecturer Name;Monday;…`) is tolerated because its first column is not a number. Lines starting with `;` are skipped.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Lecturer code | Must be a non-zero integer, and must match the lecturer's code field exactly as text. Unknown codes are skipped without a warning | `1042` |
| 2 | Lecturer name | Read but never used — matching is by code only. May be empty | `John Smith` |
| 3 | Monday | Comma-separated list of `HH:MM-HH:MM` ranges. May be empty | `08:00-12:00` |
| 4 | Tuesday | as above, may be empty | |
| 5 | Wednesday | as above, may be empty | `14:00-18:00` |
| 6 | Thursday | as above, may be empty | |
| 7 | Friday | as above, may be empty | `08:00-10:00,14:00-16:00` |
| 8 | Saturday | as above, may be empty | |
| 9 | Sunday | as above, may be empty. Note that the matching export only writes through Saturday | |

### Sample

```
Lecturer ID;Lecturer Name;Monday;Tuesday;Wednesday;Thursday;Friday;Saturday;Sunday
1042;John Smith;08:00-12:00;;14:00-18:00;;08:00-10:00,14:00-16:00;;
1117;Jane Doe;;10:00-14:00;;;09:00-13:00;;
```

### Notes.

- **Both codes are identical.** They dispatch to the same function with the same arguments; use whichever is more convenient.
- **Full replacement, not a merge.** Every existing wanted reservation of the matched lecturer is discarded before the new ranges are created, on all days. The file must therefore carry the lecturer's complete weekly availability. If the same lecturer appears on two lines, the second line wipes the reservations created by the first — one line per lecturer only.

- **Matching depends on lecturer codes.** Codes are set by `MODGRPM` (column 6) and `MODGRPB` (column 1), so run one of those first. A lecturer whose code is empty or does not match is silently ignored with no warning, and the line still counts as read.
- **Reservations created:** type "wanted", weeks 1 to 52, from-day equals to-day, no owner. The start and end of each range are snapped to the nearest time slot with no tolerance limit, and the duration is the difference in slots — a range whose ends snap to the same slot produces a zero-length reservation, and end times are exclusive of the closing slot (`08:00-12:00` covers the slots from 08:00 up to but not including 12:00, matching what the export writes back).
- Does not use the Program/Branch/Year selected in the workspace. `-d` and `-f` have no effect.
- **Changed in this revision.** Within a day column, a segment without a `-` was skipped *without* advancing the parse pointer, so every remaining range for that day was lost. A malformed segment is now skipped on its own and the rest of the day is read. Keeping each day column as clean `HH:MM-HH:MM` items is still the right thing to do.
- **Ordering.** Run after `MODGRPM`/`MODGRPMI`/`MODGRPB`.

## EXTLECIDS — Lecturer external IDs

**What it does.** Reads a lecturer list exported from an external system and reconciles it with the tutors in the application. The tutor is looked up by first name and last name and is created if not found. The external identifier from column 3 is stored both as the tutor's code (which is what `MODCRSEVENTS` and other code-based imports use) and as the internal source ID, and the e-mail address from column 1 is stored on the tutor. A tutor that has no role yet gets the default role value `100`.

**How to start it.** `Data ▸ Custom import`, then type `EXTLECIDS` at the prompt and pick the file.

**File format.** Semicolon (`;`) separated, one lecturer per line. No header line is expected — a header would be imported as a tutor. Quoting is not supported. Note that the name column itself is split on a comma, so column 2 must use the `Surname, Name` convention. UTF-8 preferred; `.xls`/`.xlsx` are converted to CSV first. Leading and trailing spaces are removed from every field. Lines shorter than 2 characters are skipped silently.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | E-mail | Tutor e-mail address. May be empty. | `j.smith@university.edu` |
| 2 | Full name | `Surname, Name`. Everything before the first comma is the surname, everything after it is the first name. Without a comma the whole field is taken as the surname and the first name stays empty. | `Smith, John` |
| 3 | External ID | External identifier; written to the tutor code field and to the internal source ID field. | `JSMITH` |

**Sample**

```
j.smith@university.edu;Smith, John;JSMITH
m.jones@university.edu;Jones, Maria;MJONES
r.brown@university.edu;Brown, Robert;RBROWN
```

**Notes.**

- Tutors are auto-created when the name is not found, so a spelling difference between the file and the application produces a duplicate person instead of an update. Check the name spelling before running the import.

- The tutor code is what makes `MODCRSEVENTS` able to resolve its tutor column; run `EXTLECIDS` before `MODCRSEVENTS` when the tutor codes are not maintained manually.
- Only role, code, source ID and e-mail are touched; availability, contracts and other tutor data are left untouched. The default role `100` is written only when the tutor currently has no role.
- The line is rejected (logged and counted as a warning) only when the tutor can neither be found nor created.
- **Changed in this revision.** A line with fewer than 3 columns silently kept the previous line's values for the missing columns. The buffers are cleared per line now, a line without a name is skipped, and an empty code or e-mail no longer overwrites a good one.
- Independent of the workspace selection.

## GETMAILS — Harvest e-mail addresses from an HTML page

**What it does.** This reads data *out of* a file rather than importing anything. It looks for the first occurrence of `mailto:` on each line, takes everything after it up to the first apostrophe, and appends that address to `c:\temp\srednjeMails.txt`, separated by `;`. Nothing is written into the schedule.

**How to start it.** `Data ▸ Custom import`, then type `GETMAILS` at the prompt and pick the file.

**File format.** Saved HTML (or any text containing `mailto:` links); not parsed as CSV. The file dialog filters on `*.csv`, `*.xlsx`, `*.xls`, so rename the saved page to a `.csv` extension before running the import. No header line. One line may yield at most one address.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | whole line | Any text. The first `mailto:` on the line starts the address; the address ends at the first `'` after it. | `<a href='mailto:office@school-01.example.edu'>` |
| — | terminator line | A line starting with `</html>` closes the output file. | `</html>` |

**Sample**

```
<td><a href='mailto:office@school-01.example.edu'>Office</a></td>
<td><a href='mailto:office@school-02.example.edu'>Office</a></td>
</html>
```

**Notes.**

- The address must be delimited by a single quote (`href='mailto:…'`). If the page uses double quotes, nothing terminates the address and the rest of the line is written out as if it were the address.
- Only the first `mailto:` per line is taken. Reformat the source so each link sits on its own line.
- The output directory `c:\temp` must already exist.
- The last line must start with `</html>` or the output file is never closed.
- The output file is opened only once per application session — restart the application before running the harvest a second time.

## GETMAILS_UNI — Harvest e-mail addresses from a text dump

**What it does.** Also a read-out rather than an import. It scans every line character by character, and for each `@` it finds it walks backwards and forwards to the nearest delimiter to isolate the whole e-mail address. Every address found is appended to the text file `c:\temp\uniMails.txt`, separated by `;`. Nothing at all is written into the schedule.

**How to start it.** `Data ▸ Custom import`, then type `GETMAILS_UNI` at the prompt and pick the file.

**File format.** Free-form text; it is not parsed as CSV. The file dialog filters on `*.csv`, `*.xlsx`, `*.xls`, so give the source dump a `.csv` extension before running the import. UTF-8 or legacy ANSI. No header line. One line may contain any number of addresses. Address boundaries are the characters `"`, space, `;`, `'`, plus CR/LF/start-of-line.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | whole line | Any text. Every substring around an `@` that is bounded by `"`, space, `;`, `'` or the start of the line is extracted as an address. **Both** boundaries must be present, otherwise that address is skipped. | `"info@university.example.edu";` |
| — | terminator line | A line starting with `end` closes the output file. | `end` |

**Sample**

```
"info@university.example.edu"; "registry@university.example.edu";
j.smith@faculty.example.edu; k.jones@faculty.example.edu;
end
```

**Notes.**

- The output directory `c:\temp` must already exist; if the output file cannot be created the import fails hard on the first address found.
- An address sitting at the very end of a line with no trailing `"`, space, `;` or `'` is silently dropped — always terminate each address with `;`.
- The last line of the file must start with `end`, otherwise the output file is never closed and the harvested list can be left incomplete on disk.
- The output file is opened only once per application session — restart the application between runs.
- The result is a single long line of addresses joined by `;`, ready to paste into a mail client.

## IMPORTLECT — Import / update lecturers (code, name, e-mail, password)

**What it does.** Creates and updates lecturer records. If a lecturer code is supplied and a lecturer with that code exists, that record is updated in place with the name, surname and e-mail from the file — this is the way to rename a lecturer without losing their teaching assignments. If no lecturer has that code, or the code column is empty, the lecturer is looked up by first name + surname and created if not found. An optional fifth column sets the lecturer's password.

**How to start it.** `Data ▸ Custom import`, then type `IMPORTLECT` at the prompt and pick the file.

**File format.** Semicolon-separated text, always `;` and not the configured CSV separator; no quote handling. UTF-8 or legacy ANSI. Each line is trimmed as a whole and every field is trimmed individually. Lines shorter than 2 characters are skipped. A header line is tolerated — any line containing the text `email` (case-insensitive) anywhere is skipped. One line = one lecturer.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Lecturer code | Optional, may be empty. If filled and matched, the existing lecturer with this code is renamed; if filled and unmatched, the code is assigned to the lecturer | `L-0421` |

| # | Column | Description | Example |
|---|---|---|---|
| | | found/created by name. | |
| 2 | First name | Used for the name lookup and stored on the record. | `Maria` |
| 3 | Surname | Used for the name lookup and stored on the record. | `Andersen` |
| 4 | E-mail | May be empty. | `maria.andersen@university.example.edu` |
| 5 | Password | Optional, may be empty. Stored exactly as written in the file — it is **not** hashed by this import. | `Autumn-2024` |

**Sample**

```
code;name;surname;email;password
L-0421;Maria;Andersen;maria.andersen@university.example.edu;Autumn-2024
;Peter;Whitfield;peter.whitfield@university.example.edu;
```

**Notes.**

- **Changed in this revision.** When the code column was filled but no lecturer carried that code, the lecturer was created by name and the code was set, but the **e-mail from column 4 was silently dropped**. It is stored on that path now, so one pass is enough.
- The header-skip rule drops *any* line containing `email`, not just the first one. A lecturer whose address or name contains that string is silently skipped.
- Supply all five separators on every line.
- The password is stored unhashed, unlike `STUDPASSSET` / `LECPASSSET`. The global "passwords were just imported from a file" flag is set on every line regardless of whether a password was supplied.
- Existing lecturers matched by name (empty code column) keep their existing code.
- Independent of the workspace selection.

## LECIDMAIL — Lecturers with identifier and e-mail

**What it does.** Loads the lecturer list. Each line provides or creates a lecturer by first name and surname, stores the external identifier from column 1 as the lecturer code and fills in the e-mail address. If the lecturer has no role yet, the role is set to `100`. Run this before `CRSHOURS`/`CRSHOURSBR`, which look lecturers up by that code.

**How to start it.** `Data ▸ Custom import`, then type `LECIDMAIL` at the prompt and pick the file.

**File format.** Semicolon-separated; the separator is fixed. UTF-8 or legacy ANSI. A header line may be left in place — every line whose first column is not a non-zero number is silently skipped. One line = one lecturer.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Lecturer ID | Must be a non-zero number, otherwise the line is skipped. Stored as the lecturer code. | `4711` |
| 2 | Surname | Used together with column 3 to find or create the lecturer. | `Novak` |
| 3 | First name | | `Andrej` |
| 4 | Code | Read by the parser but **not stored anywhere** — the | `AN` |

| # | Column | Description | Example |
|---|---|---|---|
| | | lecturer code comes from column 1. May be empty. | |
| 5 | E-mail | Written to the lecturer record. May be empty. | `andrej.novak@example.edu` |

**Sample**

```
ID;SURNAME;NAME;CODE;EMAIL
4711;Novak;Andrej;AN;andrej.novak@example.edu
4712;Stubelj;Igor;IS;igor.stubelj@example.edu
```

**Notes.**

- Lecturers are auto-created when the name/surname pair is not found; if the lecturer could not be provided the line is reported as a warning in the final popup.
- The role is only defaulted to `100` when it is still empty; an existing role is kept.
- Existing code and e-mail are overwritten on every run, so the import is repeatable.
- Independent of the workspace selection.

## LECMAILNOTE — Teacher e-mail addresses and notes

**What it does.** Fills in contact data for teaching staff. Each line contains a surname, first name, e-mail and a free-text comment. The teacher is looked up by first name / surname and created if not found, then the e-mail and the note are overwritten with the values from the line.

**How to start it.** `Data ▸ Custom import`, then type `LECMAILNOTE` at the prompt and pick the file.

**File format.** Plain text, semicolon (`;`) separated, one line per teacher. No quoting. Read through the UTF-8 reader. **The first line is always skipped**, so a header row is expected. Lines shorter than 2 characters and lines starting with `;` are skipped. 4 columns.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Surname | Together with column 2 identifies the teacher; created if not found. | `Novak` |
| 2 | First name | Note the reversed order: surname first. | `Ana` |
| 3 | E-mail | Overwrites the teacher's e-mail; may be empty (which clears it). | `ana.novak@example.edu` |
| 4 | Note | Free text stored as the teacher's note; may be empty (which clears it). | `Visiting lecturer` |

**Sample**

```
Surname;Name;Email;Note
Novak;Ana;ana.novak@example.edu;Head of department
Kovac;Peter;peter.kovac@example.edu;
Horvat;Maja;maja.horvat@example.edu;Visiting lecturer
```

**Notes.**

- The name spelling must match exactly what `CURRICULUM` (or `LECTOCRS`) produced, otherwise a second, duplicate teacher record is created. Export both files from the same source so the spellings agree.
- **Both the e-mail and the note are overwritten unconditionally, including with an empty value** — an empty column 3 or 4 wipes whatever was stored there before. Fill the whole column or do not include those teachers in the file.

- Run this after `CURRICULUM`.

## LECPASSSET — Set lecturer passwords from a file

**What it does.** Assigns a password to existing lecturers, matching each line to a lecturer by surname only. The password is hashed before being stored, so the file holds clear text and the database does not. No lecturer is created.

**How to start it.** `Data ▸ Custom import`, then type `LECPASSSET` at the prompt and pick the file.

**File format.** Semicolon-separated text, always `;` and not the configured CSV separator; no quote handling. UTF-8 or legacy ANSI. No header line. One line = one lecturer. Lines shorter than 3 characters and lines starting with `;` or `'` are skipped.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Lecturer surname | Matched case-insensitively against the surname of an existing lecturer. First name is not used. | `Andersen` |
| 2 | Password | Clear-text password; hashed on import. Everything after the first `;` belongs to this field. | `Autumn-2024` |

**Sample**

```
'surname;password
Andersen;Autumn-2024
Whitfield;Rowan.88
```

**Notes.**

- **Matching is by surname only.** If two lecturers share a surname, only the first one found gets the password and the second is left untouched with no warning.
- Unknown surnames, empty fields and lines without a `;` are skipped silently; the import always reports success.
- **Changed in this revision.** Both fields were trimmed *before* they were filled, so the trim did nothing and a space around the `;` became part of the password. Both fields are trimmed properly now.
- The hashed password is stored into a 33-byte field, i.e. a 32-character hash; the clear-text length in the file is not limited by that.
- Sets the global “passwords were just imported from a file” flag. Changes are in memory only — save after the import.

## LECTOCRS — Assign teachers to courses by course code

**What it does.** Reads a semicolon-separated list in which each line carries a course code and a teacher’s full name, and registers that teacher as a *suitable tutor* on the course. The course is found by an exact match of the code column against the code of an already existing course; if no course has that code the line is silently ignored. When a match is found, the teacher is looked up (or created) by first name / last name and their id is added to the suitable-tutor list of **every** course part of that course.

**How to start it.** `Data ▸ Custom import`, then type `LECTOCRS` at the prompt and pick the file.

**File format.** Plain text, semicolon (`;`) separated, one line per course-code / teacher pair. No quoting. Read through the UTF-8 reader. No header line is skipped: a header is simply ignored because its first column will not match any course code. Lines shorter than 2 characters are skipped. At least 6 columns must be present.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Course code | Must match the code of an existing course exactly; otherwise the line is skipped without a warning. | `63201` |
| 2 | (ignored) | Read past but not used. | `Programming I` |
| 3 | (ignored) | Read past but not used. | `Lectures` |
| 4 | (ignored) | Read past but not used. | `1` |
| 5 | (ignored) | Read past but not used. | `15` |
| 6 | Teacher full name | Text up to the first space becomes the first name, everything after it becomes the surname. If there is no space, the whole text is the first name and the surname is empty. | `Ana Novak` |

**Sample**

```
63201;Programming I;Lectures;1;15;Ana Novak
63315;Machine Learning;Lectures;2;30;Maja Horvat
63315;Machine Learning;Tutorials;2;30;Peter Kovac
```

**Notes.**

- Courses are **not** created by this import — they must already exist (typically imported first with `CURRICULUM`). Teachers **are** created if they do not exist yet.
- The name split is naive: `Ana Marija Novak` becomes first name `Ana`, surname `Marija Novak`.
- The teacher is added to the suitable-tutor list of all course parts of the course, not to a specific one — column 3 is not evaluated.
- If courses were created by `CURRICULUM`, a course that received more than one code has them stored comma-joined (for example `63201,63202`). The exact comparison used here will then fail for both codes. Feed `LECTOCRS` codes that match the stored value exactly.
- Re-running the import adds no visible duplicates, but no teacher is ever removed — the import only adds.

## MEDLECT — Medical lecturers (people)

**What it does.** Creates or updates lecturers from a staff list. The tutor first-name field is composed as `title first name` and the surname is taken as given; e-mail, staff initials (stored as the tutor code) and department (stored as a note) are filled in, the tutor id is forced to the id from the file, and a password is generated for the person.

**How to start it.** `Data ▸ Custom import`, then type `MEDLECT` at the prompt and pick the file.

**File format.** Semicolon-separated (`;`), one person per line. No header line is expected and none is skipped. At least six separators must be present; the e-mail is everything after the sixth separator.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Person id | Integer; written into the tutor record as its id. | `501` |
| 2 | Initials | Optional, may be empty. Stored as the tutor code and used by `MEDCLASS` to resolve the lecturer. | `JDS` |
| 3 | Title | Optional, may be empty. Prepended to the first name. | `Prof.` |
| 4 | First name | Given name. | `John` |
| 5 | Surname | Family name; used as the tutor surname and for the generated password. | `Smith` |
| 6 | Department | Optional, may be empty. Stored in the tutor note as | `Physiology` |

| # | Column | Description | Example |
|---|---|---|---|
| | | `Dept.: <department>`. | |
| 7 | E-mail | Rest of the line. Optional; written only when non-empty. | `j.smith@example.ac.uk` |

### Sample

```
501;JDS;Prof.;John;Smith;Physiology;j.smith@example.ac.uk
502;AMB;Dr.;Anna;Brown;Anatomy;a.brown@example.ac.uk
503;;;Peter;Clark;;p.clark@example.ac.uk
```

### Notes.

- The tutor is looked up by the composed name (title plus first name) and the surname, and created when not found. Because the title is part of the stored first name, changing a person's title between two runs produces a second tutor record. The title-insensitive helpers used by `PARTTIME` are not applied here.
- The tutor id is overwritten with the value from column 1 on every run, also for tutors that already existed — make sure those ids do not clash with people already in the base.
- **A password is generated on every run:** lowercased first name + the first whole character of the lowercased surname + a random number between 1010 and 9999. The initial is taken as a complete UTF-8 character, so accented names do not produce invalid UTF-8 in the later XML export. Re-importing the file therefore changes everybody's password.
- E-mail, initials and department are only written when non-empty.
- **Ordering:** run `MEDLECT` before `MEDCLASS`; `MEDCLASS` resolves the lecturer by the initials from column 2 and skips the line entirely when the code is unknown.

## Students and enrolments

Student records, their group membership, their course enrolments and their passwords. Almost all of these need the courses and the groups to exist first. ## ENRLS / ENRLSD — Students with enrolled courses (e-mail first, fixed 8 course slots)

**What it does.** Imports students and their course enrolment in a fixed-width layout: e-mail first, student number in column 4, exactly eight course-code slots, and the program name in the last column. The student is matched by student number and created if unknown; each recognised course is linked to the student, gets its chief teacher set from the first teacher of its first turn, and is flagged as requiring a final exam. Finally the student is allocated to the group derived from the program name.

**How to start it.** `Data ▸ Custom import`, then type `ENRLS` (or `ENRLSD`) at the prompt and pick the file.

**File format.** Semicolon (`;`) separated, fixed separator, no quoting. UTF-8 transcoded on read; `.xls`/`.xlsx` is converted first. No header line is skipped explicitly — a header row is normally discarded because its first cell contains no `@`. One line = one student; exactly 13 columns are expected.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | E-mail | Must contain `@`, or be exactly `NULL`; otherwise the line is skipped without warning (this is what filters out the header). | `dana.levi@example.edu` |
| 2 | First name | Student given name. May be empty. | `Dana` |
| 3 | Surname | Student family name. May be empty. | `Levi` |
| 4 | Student number | Identifier used to match an existing student. | `123456` |
| 5–12 | Course codes | Eight fixed course-code slots. Values shorter than 3 characters are ignored, so unused slots may be left empty. Unknown codes are logged and skipped. | `CS 201` |
| 13 | Program | Program/branch name, must contain a space; the group name is derived from it. May be empty. | `Computer Science` |

**Sample**

```
dana.levi@example.edu;Dana;Levi;123456;CS 201;CS 305;MA 101;;;;;;Computer Science
amir.katz@example.edu;Amir;Katz;123457;CS 201;SE 210;;;;;;;Software Engineering
```

**Notes.**

- Does not depend on the workspace selection.
- The study year used for the group name comes from the **last recognised course code** on the line: the number after the first space in the code, divided by 100, minus 5 when above 5 (so `CS 601` gives year 1). Because the program column is last, this works here, unlike in `ENRLST`. If no course was matched, year 1 is used.
- Group names follow the same rule as the other BIUST imports: `<first word of program>-<capital letters of the remaining words>-L<year>`, e.g. `Computer Science` year 2 becomes `Computer-S-L2`. A missing group is logged and the student simply gets no group.
- Only the student record is auto-created; courses, programs, branches and groups must exist beforehand.
- All 13 columns must be present — a shorter line never reaches the program column, and the course columns rely on the student record already having been created from column 4.

- **`ENRLSD` now really deletes.** The dispatcher tested the `ENRLSTD` code on both sides of its `||`, so `ENRLSD` never ran its pre-pass and behaved exactly like `ENRLS`. **Changed in this revision:** `ENRLSD` clears the whole student list before the import, as its name always implied. If you have been using `ENRLSD` as a merge, switch to `ENRLS`.
- Course links are appended without a duplicate check, so re-running the same file duplicates them.
- **Ordering:** run after `ENRLP` and `ENRLC`.

## ENRLST / ENRLSTD — Students with enrolled courses (student-number first)

**What it does.** Imports students and their course enrolment. Each line identifies a student by student number (creating the student when the number is unknown, updating them otherwise), sets e-mail, first name and surname, allocates the student to the group derived from the program name, and links every course code listed in the trailing columns. For each linked course it also sets the course's chief teacher from the first teacher of the course's first turn and marks the course as requiring a final exam.

**How to start it.** `Data ▸ Custom import`, then type `ENRLST` (or `ENRLSTD`) at the prompt and pick the file.

**File format.** Semicolon (`;`) separated, fixed separator, no quoting. UTF-8 transcoded on read; `.xls`/`.xlsx` is converted first. No fixed header line, but any line containing `;email`, `;Courses`, `name;`, `Student email;` or `surname;` (case-insensitive) is treated as a header and skipped. One line = one student, with a variable number of course codes at the end.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Student number | Unique student identifier used for matching existing students. | `123456` |
| 2 | E-mail | Must contain `@`, or be exactly `NULL`; otherwise the whole line is skipped without warning. | `dana.levi@example.edu` |
| 3 | First name | Student given name. May be empty. | `Dana` |
| 4 | Surname | Student family name. May be empty. | `Levi` |
| 5 | Program | Program/branch name, must contain a space; the group name is derived from it. May be empty (student then gets no group). | `Computer Science` |
| 6 … n | Course codes | One course code per column, any number of columns. Values shorter than 3 characters are ignored, so empty slots are allowed. Unknown codes are logged and skipped. | `CS 201` |

### Sample

```
Student ID;Student email;Name;Surname;Program;Courses
123456;dana.levi@example.edu;Dana;Levi;Computer Science;CS 201;CS 305;MA 101
123457;amir.katz@example.edu;Amir;Katz;Software Engineering;CS 201;SE 210
```

### Notes.

- Does not depend on the workspace selection.
- **`ENRLSTD` is the "delete first" variant:** before the main pass it clears the whole student list, so the file becomes the complete new student population. `ENRLST` merges into the existing students.

- Nothing is auto-created except the student record: courses, programs, branches and groups must already exist. Run `ENRLP` and `ENRLC` before this import.
- Course links are appended without duplicate checking, so re-running `ENRLST` over the same file adds the same course ids again; use `ENRLSTD` for a clean reload.
- The student's year is taken from the branch of the matched group; if the group's branch is invalid the line stops there.
- **Changed in this revision.** The study year comes from the course codes, but the program column (5) was handled *before* them, so the group name was always built with year 1 (`…-L1`). The group is now resolved after the whole line has been read, exactly as `ENRLS` does — which means files that were built around the old `-L1` behaviour now land in the correct year's group.
- The line must contain at least the first four columns.

## STUDBYCRS — Student enrolments per course (names in file)

**What it does.** Creates or updates students and enrols them into a course. Each line carries a student registration number, first name, surname, e-mail and a course code. The course is found by exact code match among existing courses; lines whose code matches nothing are skipped silently. The student is looked up by registration number: if found, the existing record is overwritten with the values from the line, otherwise a new student record is created. The course id is then appended to the student's course list.

**How to start it.** `Data ▸ Custom import`, then type `STUDBYCRS` at the prompt and pick the file.

**File format.** Plain text, semicolon (`;`) separated, one line per student-course enrolment (a student enrolled in five courses appears on five lines). No quoting. Read through the UTF-8 reader. No header line is skipped, but a header is harmless because its code column will not match a course. Lines shorter than 2 characters are skipped. At least 8 columns must be present.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Registration number | Key used to find an existing student; stored as the student number. | `64180012` |
| 2 | First name | Overwrites the stored first name. | `Ana` |
| 3 | Surname | Overwrites the stored surname. | `Novak` |
| 4 | E-mail | Overwrites the stored e-mail; may be empty. | `ana.novak@student.example.edu` |
| 5 | (ignored) | Read past but not used. | `1` |
| 6 | (ignored) | Read past but not used. | `Computer Science` |
| 7 | (ignored) | Read past but not used. | `Software Engineering` |
| 8 | Course code | Must match the code of an existing course exactly; otherwise the line is skipped without a warning. | `63201` |

**Sample**

```
64180012;Ana;Novak;ana.novak@student.example.edu;1;Computer Science;Software
Engineering;63201
64180012;Ana;Novak;ana.novak@student.example.edu;1;Computer Science;Software
Engineering;63315
64180117;Peter;Kovac;peter.kovac@student.example.edu;1;Computer Science;Software
Engineering;63201
```

**Notes.**

- Three related codes exist: `STUDBYCRS` (names in the file, no program/branch), `STUDBYCRS2` (no names, program/branch/year columns) and `STUDBYCRS3` (names *and* program/branch/year columns). They are separate readers with different layouts — do not mix the layouts.
- Courses must already exist. Students are auto-created; programs, branches and years are **not** touched by this variant, so a student created here has no program, branch or year assigned.
- The execution type recorded with each enrolment is `-1` (unspecified) — the student is enrolled into the whole course, not into a particular lecture/tutorial part.
- The enrolment is appended without a duplicate check, so importing the same file twice enrols the student twice into the same course.
- The import never deletes: students or enrolments that disappear from the source file stay in the schedule.

## STUDBYCRS2 — Student enrolments with program, branch and year (no names)

**What it does.** Same purpose as `STUDBYCRS`, but the source file contains no personal names — only a registration number and an e-mail — and it additionally carries program code, branch code and year, which are used to attach the student to a study program/branch. The student is matched by registration number and created if missing. Because there is no name column, the reader stores a literal `-` as the first name and the **e-mail address as the surname**.

**How to start it.** `Data ▸ Custom import`, then type `STUDBYCRS2` at the prompt and pick the file.

**File format.** Plain text, semicolon (`;`) separated, one line per student-course enrolment. No quoting. Read through the UTF-8 reader. No header line is skipped. Lines shorter than 2 characters are skipped. At least 8 columns must be present.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Registration number | Key used to find an existing student; stored as the student number. | `64180012` |
| 2 | E-mail | Stored as the e-mail **and** as the surname; the first name is forced to `-`. | `ana.novak@student.example.edu` |
| 3 | Program code | Used to find an existing program by code; may be empty. | `CS` |
| 4 | (ignored) | Read past but not used (typically the program name). | `Computer Science` |
| 5 | Branch code | Used to find an existing branch by code **and** year; may be empty. | `SE` |
| 6 | (ignored) | Read past but not used (typically the branch name). | `Software Engineering` |
| 7 | Year | Study year, numeric; also used as the year of the branch lookup. `0` or empty leaves the student's year unchanged. | `1` |
| 8 | Course code | Must match the code of an existing course exactly; otherwise the line is skipped without a warning. | `63201` |

### Sample

```
64180012;ana.novak@student.example.edu;CS;Computer Science;SE;Software Engineering;1;63201
64180012;ana.novak@student.example.edu;CS;Computer Science;SE;Software Engineering;1;63315
64180117;peter.kovac@student.example.edu;CS;Computer Science;SE;Software
Engineering;2;63201
```

**Notes.**

- **Program/branch resolution order:** the branch is first looked up by branch code + year (columns 5 and 7). If that fails and a program code is given, the program is looked up by code and then the first branch of that program with the matching year is taken. Nothing is auto-created — an unknown program code or branch code simply leaves the student unassigned.
- If a program was resolved but no branch, the student's branch code is set to a synthetic marker `P:<program code>` (or `P:<program id>` when the program has no code), which keeps the student attached to the program only.
- The student's branch code is cleared at the start of every line, so the **last** line for a given student decides their program/branch/year. Keep all lines of one student consistent.
- Surname = e-mail address is by design for this variant, since the source has no names. If names are available in the export, use `STUDBYCRS3` instead.
- Courses must already exist; the enrolment execution type is `-1`. Enrolments are appended without a duplicate check.

## STUDBYCRS3 — Student enrolments with names, program, branch and year

**What it does.** The full variant: creates or updates students from registration number, first name, surname and e-mail, enrols them into a course found by exact code match, and attaches them to a program/branch/year exactly as `STUDBYCRS2` does. Lines whose course code matches no existing course are skipped silently.

**How to start it.** `Data ▸ Custom import`, then type `STUDBYCRS3` at the prompt and pick the file.

**File format.** Plain text, semicolon (`;`) separated, one line per student-course enrolment. No quoting. Read through the UTF-8 reader. No header line is skipped. Lines shorter than 2 characters are skipped. At least 10 columns must be present.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Registration number | Key used to find an existing student; stored as the student number. | `64180012` |
| 2 | First name | Overwrites the stored first name. | `Ana` |
| 3 | Surname | Overwrites the stored surname. | `Novak` |
| 4 | E-mail | Overwrites the stored e-mail; may be empty. | `ana.novak@student.example.edu` |
| 5 | Program code | Used to find an existing program by code; may be empty. | `CS` |
| 6 | (ignored) | Read past but not used (typically the program name). | `Computer Science` |
| 7 | Branch code | Used to find an existing branch by code **and** year; may be empty. | `SE` |
| 8 | (ignored) | Read past but not used (typically the branch name). | `Software Engineering` |
| 9 | Year | Study year, numeric; also the year of the branch lookup. `0` or empty leaves the student's year unchanged. | `1` |
| 10 | Course code | Must match the code of an existing course exactly; otherwise the line is skipped without a warning. | `63201` |

**Sample**

```
64180012;Ana;Novak;ana.novak@student.example.edu;CS;Computer Science;SE;Software
Engineering;1;63201
64180012;Ana;Novak;ana.novak@student.example.edu;CS;Computer Science;SE;Software
Engineering;1;63315
64180117;Peter;Kovac;peter.kovac@student.example.edu;CS;Computer Science;SE;Software
Engineering;2;63201
```

**Notes.**

- Program/branch resolution, the `P:<program code>` fallback marker and the "last line wins" behaviour are identical to `STUDBYCRS2`.
- Courses must already exist; the enrolment execution type is `-1`; enrolments are appended without a duplicate check.
- **Note the column shift against `STUDBYCRS2`:** this layout has two extra name columns at the front, which pushes program code from 3 to 5, branch code from 5 to 7, year from 7 to 9 and the course code from 8 to 10. Importing a `STUDBYCRS2` file under `STUDBYCRS3` will not fail loudly — it will simply match no courses and import nothing.

## STUDCRSENR — Students and their course enrolments

**What it does.** Reads a student list with a course code per line and links students to courses. The course is located by an exact match on the course code; if no course carries that code the line is reported in the log and skipped. If a student with the same e-mail already exists, the course is added to that student's course list (with execution type `-1`, i.e. all types). If the student does not exist yet, a new student record is created with name, surname, e-mail and a student number derived from the e-mail.

**How to start it.** `Data ▸ Custom import`, then type `STUDCRSENR` at the prompt and pick the file.

**File format.** Semicolon-separated plain text, one student-course pair per line, no header line expected. The line must contain at least three semicolons; shorter lines are skipped without a warning. Lines must not begin with whitespace.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | E-mail | Identifies the student. The part before `@` also becomes the student number. | `an1234@student.uni.edu` |
| 2 | First name | Used only when the student is created. | `Ana` |
| 3 | Surname | Used only when the student is created. | `Novak` |
| 4 | Course code | Everything after the third semicolon, compared **exactly** (case-sensitive) with the course code stored in the database. | `SOC101` |

**Sample**

```
an1234@student.uni.edu;Ana;Novak;SOC101
js5678@student.uni.edu;John;Smith;SOC101
js5678@student.uni.edu;John;Smith;POL204
```

**Notes.**

- Courses are **not** created by this import — they must already exist with the correct code. Run `CRSCODESET` or `CRSRECODE` first.

- **Changed in this revision.** When the student was *created* by this import the course was not attached — only the personal data was written — so the file had to be imported twice. The enrolment is written on the creation path now; one pass is enough.
- The student's code field is deliberately cleared; the student number is the e-mail local part.
- Enrolments are added with execution type `-1`, so the student is attached to the course as a whole.
- Independent of the workspace selection.

## STUDLIST / STUDLISTPLUS — Student list

**What it does.** Loads students (number, surname, first name, programme, e-mail, external code) and attaches each of them to a programme when the programme name can be matched. The two codes are the same reader taking two different column layouts. Every line always adds a **new** student record.

**How to start it.** `Data ▸ Custom import`, then type `STUDLIST` (or `STUDLISTPLUS`) at the prompt and pick the file.

**File format.** Semicolon (`;`) separated, one student per line. Lines shorter than 10 characters, lines starting with `#` and lines whose **first cell does not start with a digit** are skipped, so a text header line is dropped automatically. All values are trimmed.

`STUDLIST`:

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Student number | Must be numeric at the start or the line is skipped. Stored as the student number. | `12345` |
| 2 | Surname | | `Smith` |
| 3 | First name | | `John` |
| 4 | Programme name | Matched against existing programmes (see Notes). May be empty. | `BA Computer Science` |
| 5 | E-mail | May be empty. | `john.smith@example.edu` |
| 6 | External code | Stored in the student's code field. May be empty. | `abcd1234` |

`STUDLISTPLUS`:

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Student number | Must be numeric at the start. | `12345` |
| 2 | (ignored) | Placeholder, any value. | |
| 3 | Surname | | `Smith` |
| 4 | First name | | `John` |
| 5 | Programme name | Additionally rewritten before matching (see Notes). | `BA Computer Science` |
| 6 | (ignored) | Placeholder, any value. | |
| 7 | E-mail | May be empty. | `john.smith@example.edu` |
| 8 | External code | Stored in the student's code field. May be empty. | `abcd1234` |

**Sample** (`STUDLIST`)

```
12345;Smith;John;BA Computer Science;john.smith@example.edu;abcd1234
12346;Jones;Mary;MSc in Computer Science;mary.jones@example.edu;efgh5678
```

**Sample** (`STUDLISTPLUS`)

```
12345;x;Smith;John;BA Computer Science;x;john.smith@example.edu;abcd1234
12346;x;Jones;Mary;MSc Comp Science and Philosophy;x;mary.jones@example.edu;efgh5678
```

**Notes.**

- **`STUDLISTPLUS` only:** the programme cell is cut at the first space. If the remainder is exactly `Comp Science and Philosophy`, the programme becomes `Computer Science` when the prefix is `BA` and `MSc in Computer Science` otherwise; in every other case the programme becomes the text after the first space (`BA Computer Science` becomes `Computer Science`).
- Programme assignment uses the first programme whose name is **contained in** the programme cell. If nothing matches, the student is imported without a programme. Programmes are never created, and no group is assigned.
- **There is no de-duplication:** each line always creates a new student, so importing the same file twice doubles the student list. Clear the students first if you are re-importing.
- Independent of the workspace selection.

## STUDPASSSET — Set student passwords from a file

**What it does.** Assigns a password to existing students, matching each line to a student by student number. The password from the file is hashed before it is stored, so the file contains the passwords in clear text and the database does not. Only students that already exist are touched — no student record is ever created.

**How to start it.** `Data ▸ Custom import`, then type `STUDPASSSET` at the prompt and pick the file.

**File format.** Semicolon-separated text, always `;` and not the configured CSV separator; no quote handling. UTF-8 or legacy ANSI. No header line is expected. One line = one student. Lines shorter than 3 characters and lines starting with `;` or `'` are skipped, so `'` can be used to comment a line out.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Student number | Matched case-insensitively against the student number of an existing student. Leading/trailing spaces are trimmed. | `S1000123` |
| 2 | Password | Clear-text password; hashed on import. Leading/trailing spaces are trimmed. Everything after the first `;` belongs to this field. | `Welcome2024` |

**Sample**

```
'student number;password
S1000123;Welcome2024
S1000124;Chestnut-91
```

**Notes.**

- Unknown student numbers, empty fields and lines without a `;` are skipped silently — no warning is raised and the import always reports success.
- Only the first student matching a given student number is updated.
- The import sets the global "passwords were just imported from a file" flag.
- Changes are made in memory only — save after the import.
- Note that `IMPORTLECT` stores its optional password field **without** hashing, while this import hashes; the two are not interchangeable.

## Courses and teaching load

Courses, course parts and turns — the weekly teaching load, without a fixed day, time or room. What these imports produce still has to be scheduled, by hand or by the optimiser. ## ENRLC — Courses, teaching modalities, teachers and groups

**What it does.** Builds the teaching offer: for each line it finds or creates the course by its code, attaches a course part for the teaching modality named in the line (lecture, lab, tutorial, …), creates one turn on that course part with the weekly hours taken from the line, assigns the lecturer (created if unknown), sets the student count on the turn part, and allocates the turn part to the student groups of every program listed in the first column. It also links the required room property that corresponds to the modality, and registers the course on the branch of each allocated group.

**How to start it.** `Data ▸ Custom import`, then type `ENRLC` at the prompt and pick the file.

**File format.** Semicolon (`;`) separated; the separator is fixed in this reader, so the CSV separator configured elsewhere is ignored and quoting is not supported. Read through the UTF-8 reader; `.xls`/`.xlsx` files are converted to CSV first. Line 1 is a header line and is skipped — its first cell must not be empty (see Notes). One line = one course + modality + teacher + weekly hours + the list of programs that take it.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Programs | Comma-separated list of program/branch names. Each name must contain a space; the group name is derived from it (see Notes). Must not be empty. | `Computer Science,Software Engineering` |
| 2 | Course name | Course title. If it starts with `<column 3> -`, that prefix is stripped off. | `CS 201 - Data Structures` |
| 3 | Course code | Course code, used to find or create the course. The leading letters plus the following character are also taken as an equipment/department prefix. | `CS 201` |
| 4 | Course number / level | Text of the form `<text> <number>`; the number after the first space, divided by 100, is the study year, and 5 is subtracted when the result is above 5 (so `601` is graduate year 1). If there is no space, year 1 is used. | `CS 201` |
| 5 | Modality | Teaching type; must match the name of an existing course modality (case-insensitive, exact). | `Lecture` |
| 6 | Hours per week | Weekly duration written into the turn's duration list. May be empty — treated as `0`. | `3` |
| 7 | Lecturer first name | Teacher given name. May be empty. | `David` |
| 8 | Lecturer surname | Teacher family name. | `Cohen` |
| 9 | Number of students | Integer; `0` or empty means "not set" (see the lab/research rule in the Notes). Must be present. | `60` |

### Sample

```
Programs;Course name;Course code;Course number;Modality;Weekly hours;First
name;Surname;Students
Computer Science;CS 201 - Data Structures;CS 201;CS 201;Lecture;3;David;Cohen;60
Computer Science,Software Engineering;Operating Systems;CS 305;CS 305;Lab;2;Sarah;Levi;0
```

**Notes.**

- Does not depend on the workspace selection.
- **Group names are derived** from each program name in column 1 as `<first word>-<capital letters of the remaining words>-L<year>`, e.g. `Computer Science` with year 1 becomes `Computer-S-L1`. A program name without a space is skipped with a message; a derived group that does not exist stops processing of the remaining programs on that line (the line is not reported as an error). Programs, branches and groups are therefore **not** auto-created — run `ENRLP` first.
- Courses and teachers **are** auto-created when the code or name is unknown. Room properties, modalities and rooms are not.
- The modality in column 5 must exist as a course modality; if it does not, the line is skipped and a message is written to the log. Modalities listed in the "ignore modality" advanced setting are skipped silently.
- The room property attached to the course part is chosen by modality: `lecture`, `lab`, `project`, `research` and `tutorial` are mapped to the room properties named in the corresponding advanced settings. These property ids are looked up **only while processing line 1**, so the header line must be present and its first cell non-empty.
- For `lab` and `research`, a student count of `0` in column 9 is replaced by 40.
- The turn always spans the weeks given by the "first/last import week" advanced settings, and a **new turn is created for every line** — running the same file twice duplicates the turns.
- **`-f` suffix** (`ENRLC -f`): while processing line 1, **all existing branches are cascade-deleted** before the import. This flag is not reset between imports inside the same application session — once used, it stays active for every further custom import until the application is restarted.
- The dispatcher announces a first preprocessing pass for `ENRLC`, but that pass immediately aborts because no line handler is wired to it, so it has no effect.
- **Changed in this revision.** The room property matched from the course-code prefix (column 3) was computed but never added to the course part, because the statement that added it sat after the loop's exit. It is added now — course parts imported by `ENRLC` gain that prerequisite where they previously had none.
- **Changed in this revision.** The study year and the student count were read from uninitialised variables when the line had fewer than 4 or 9 columns; they now default to year 1 and 0 students. Still write all 9 columns.
- **Ordering:** `ENRLP` » `ENRLR`/`ENRLRE` » **`ENRLC`** » `ENRLE` » `ENRLS`/`ENRLST`.

## COURSETRANS — Import translated course names

**What it does.** Fills the translation (second-language name) field of existing courses. Each line pairs a course, identified either by its course code or by its name, with the translated name. Courses are never created; a line for an unknown course is discarded.

**How to start it.** `Data ▸ Custom import`, then type `COURSETRANS` at the prompt and pick the file.

**File format.** Semicolon-separated text, always `;` regardless of the configured CSV separator; no quote handling, so the translated name must not contain `;`. Read through the UTF-8 reader, so a legacy ANSI export with diacritics is transcoded correctly. No header line is expected. One line = one course. Lines shorter than 3 characters and lines starting with `;` or `'` are skipped.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Course code or course name | Matched case-insensitively: for each course the code is tried first (when the course has one), then the name; the first course that matches either wins. | `CS-201` |

| # | Column | Description | Example |
|---|---|---|---|
| 2 | Translated course name | Stored in the course's translation field (allocated to fit, so long names are allowed). Everything after the first `;` belongs to this field. | `Introduction to Algorithms` |

**Sample**

```
'course code or name;translation
CS-201;Introduction to Algorithms
MATH-110;Linear Algebra I
```

**Notes.**

- **Changed in this revision.** Both fields were trimmed *before* they were filled, so the trim did nothing: a space after the `;` ended up inside the stored translation and a trailing space broke the match. Both fields are trimmed properly now.
- Unmatched courses are skipped silently, with no warning; the import always reports success.
- Matching runs over all courses regardless of the workspace selection.
- A line starting with `'` is treated as a comment, which is a convenient way to keep a header in the file.

## CRSCODESET — Assign course codes to existing courses

**What it does.** Reads a two-column code-to-name mapping and writes the code onto the course whose name matches. Only the first matching course is updated. Courses are never created or renamed; a name that cannot be matched is written to the log with its code so the operator can correct the source file.

**How to start it.** `Data ▸ Custom import`, then type `CRSCODESET` at the prompt and pick the file.

**File format.** Semicolon-separated plain text, one course per line, no header line expected. A line without a semicolon is skipped silently. Lines must not begin with whitespace.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Course code | Written into the course record. | `SOC101` |
| 2 | Course name | Everything after the first semicolon. Matched against the existing course names. Must not contain a further semicolon, otherwise the remainder becomes part of the name being searched for. | `Introduction to Sociology` |

**Sample**

```
SOC101;Introduction to Sociology
POL204;Comparative Politics
COM330;Media and Communication
```

**Notes.**

- This is a code-stamping pass only: nothing is auto-created, so all courses must already exist with exactly the names used in column 2.
- Typically run before `STUDCRSENR`, which can only match students to courses that already carry a code.
- **Changed in this revision.** The line was trimmed *after* the position of the first semicolon had been remembered, so a line starting with whitespace was split at the wrong character. It is trimmed first now.

- Independent of the workspace selection.

## CRSEMBEDDED — Courses with embedded code, lecturer and group allocation

**What it does.** Reads a course list in which the course code is embedded in the course name in brackets. It finds or creates the lecturer, splits the code out of the name, finds or creates the course, gives it a lecture course part, and creates one turn spanning weeks 1–17 with a fixed duration of 2 hours. Courses marked as elective get the "selectable" flag. Finally the group whose name equals the derived code plus an underscore is allocated to the turn part, and the branch of that group is attached to the course.

**How to start it.** `Data ▸ Custom import`, then type `CRSEMBEDDED` at the prompt and pick the file.

**File format.** Semicolon-separated plain text, one course-lecturer pair per line. A header line starting with `Učna enota;` is recognised and skipped, and so is any line starting with `;`. The file is transcoded to UTF-8 on the way in.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Course name with code | Must contain the code in brackets preceded by a space: `Name (CODE)`. Everything before `(` becomes the course name; the text in brackets, cut at the first `-`, plus a trailing underscore becomes the group name to look up. A line without `(` and a closing `)` is skipped. | `Anatomy (MED-1)` |
| 2 | Category | Read but not used. May be empty. | `compulsory` |
| 3 | Elective marker | The exact value `Izbirna` marks a newly created course as selectable. Any other value, including empty, leaves the course compulsory. | `Izbirna` |
| 4 | Lecturer first name | Lecturer is created if not found. | `Ana` |
| 5 | Lecturer surname | Lecturer is created if not found. | `Novak` |

### Sample

```
Učna enota;Kategorija;Izbirnost;Ime;Priimek
Anatomy (MED-1);compulsory;;Ana;Novak
Medical Statistics (MED-2);compulsory;Izbirna;John;Smith
```

**Notes.**

- In the sample, both courses derive the group name `MED_` (the bracketed code is cut at the first `-` and an underscore is appended). A group with exactly that name must already exist; if it does not, the turn is created without any group and without a branch on the course.
- Groups and branches are never created by this import — only lecturers, courses, course parts, turns and turn parts are.
- Every line creates a **new** turn. Two lines for the same course produce two turns.
- Existing courses reuse their **first** course part regardless of its execution type; only newly created courses get an explicit lecture part.
- Course names are matched exactly, including case, against the part of column 1 that precedes `(`.
- Fixed values written by this import: weeks 1–17, duration `2`, pauses allowed.
- The lecturer is created before the code is validated, so lines that are later skipped for a malformed column 1 can still leave a new lecturer behind.
- Independent of the workspace selection.

## CRSHOURS / CRSHOURSBR — Courses with weeks, hours and lecturer

**What it does.** Builds the study structure from a course list: it provides the program, the branch and the year, provides the course (by name, under that branch, with the code from the file), and gives the course one course part with one turn that carries the week range, the weekly duration and the lecturer. The lecturer is looked up by code first and only created from name/surname when the code is unknown. `CRSHOURS` has no branch columns and uses the program name as the branch name; `CRSHOURSBR` reads the branch from the file — everything after the branch columns is shifted by two positions.

**How to start it.** `Data ▸ Custom import`, then type `CRSHOURS` (or `CRSHOURSBR`) at the prompt and pick the file.

**File format.** Semicolon-separated; the separator is fixed. UTF-8 or legacy ANSI. **The first line is always skipped**, so a header line is expected. One line = one course/lecturer combination for one program, one branch and one or more years. Column numbers below are given as `CRSHOURS / CRSHOURSBR`.

| # | Column | Description | Example |
|---|---|---|---|
| 1 / 1 | Program name | Provided (created when missing). An empty value skips the line. | `Business Administration` |
| 2 / 2 | Year(s) | Scanned digit by digit; several years must be separated by a non-digit character. `12` would be read as year 12, `1,2` as years 1 and 2. | `1,2` |
| — / 3 | Branch name | `CRSHOURSBR` only. Empty falls back to the program name. | `Marketing` |
| — / 4 | Branch code | `CRSHOURSBR` only. Read by the parser but never used. May be empty. | `MKT` |
| 3 / 5 | Course name | Course is looked up/created under the branch with this name. | `Marketing Management` |
| 4 / 6 | Course code | Passed to the course lookup as the course code. | `MNG101` |
| 5 / 7 | (ignored) | Read and discarded. | |
| 6 / 8 | (ignored) | Read and discarded. | |
| 7 / 9 | First week | Turn start week. | `1` |
| 8 / 10 | Last week | Turn end week. | `15` |
| 9 / 11 | Total hours | Used for the weekly duration: total hours divided by the number of weeks, integer division. | `45` |
| 10 / 12 | Lecture hours | Only tested for being above 0 to decide whether the duration is written; the value itself is not used in the calculation. | `30` |
| 11 / 13 | Exercise hours | Read, written into the course note only — no exercise course part is created. | `15` |
| 12 / 14 | Lecturer surname | Used only when the lecturer code is empty or unknown. | `Stubelj` |

| # | Column | Description | Example |
|---|---|---|---|
| 13 / 15 | Lecturer first name | | `Igor` |
| 14 / 16 | Lecturer code | Primary lookup key; written back onto the lecturer. May be empty (then the lecturer code is cleared). | `4712` |

**Sample**

```
PROGRAM;YEAR;COURSE;CODE;;;FIRSTWEEK;LASTWEEK;TOTAL;LECTURES;EXERCISES;SURNAME;NAME;LECTID
Business Administration;1;Marketing Management;MNG101;;;1;15;45;30;15;Stubelj;Igor;4712
Business Administration;1,2;Operations Research;MNG205;;;1;10;30;20;10;Novak;Andrej;4711
```

`CRSHOURSBR` variant:

```
Business Administration;1;Marketing;MKT;Marketing
Management;MNG101;;;1;15;45;30;15;Stubelj;Igor;4712
```

**Notes.**

- **Two variants of one reader.** The only difference is the two extra branch columns and the resulting shift of all later columns.
- **Auto-created:** program, branch (per year), course, course part, turn, turn part, and the lecturer when the code does not resolve. The workspace selection is **not** used — everything comes from the file.
- **Destructive on re-run.** If the course already has two course parts, the second one is deleted and only the first is kept. Every row reuses the first course part and the first turn of the course, so several rows for the same course in the same branch/year overwrite the week range and duration and merely add another lecturer to the same turn instead of creating separate turns.
- The course note is overwritten with a generated summary of total, lecture and exercise hours and the year string.
- No student groups are allocated: the turn part is created with zero allocated hours.
- The weekly duration divides the *total* hours (column 9), not the lecture hours, by the number of weeks — that is by design, but worth knowing when the two differ.
- **Changed in this revision.** A last week before the first week produced a division by zero; the divisor now falls back to one week. The programme's number of years was compared against the initial value 1 instead of the year just read, so it was never raised to the years present in the file; it is raised correctly now.
- Still true: the running year value only ever increases, so a year list written in descending order (e.g. `2,1`) processes the higher year twice. Write year lists in ascending order.
- **Ordering:** run `LECIDMAIL` first so the lecturer codes resolve.

## CRSRECODE — Recode and rename courses, plus English translation

**What it does.** Migrates courses from an old code to a new one. For each line it looks up the course by the old code and then overwrites its name, its code and its English translation. If the old code column is empty, the course is instead provided by the new code and name (found or created) and only the translation is set. Courses whose old code cannot be found are reported in the log and on the console.

**How to start it.** `Data ▸ Custom import`, then type `CRSRECODE` at the prompt and pick the file.

**File format.** Semicolon-separated plain text, one course per line. A header line is tolerated as long as it begins with the letter `K` — such lines are skipped. Lines shorter than 10 characters are skipped as well.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Old course code | May be empty. When empty, the course is looked up (or created) by the new code and name instead. | `SOC101` |
| 2 | New course code | Written into the course record. | `SOC-1001` |
| 3 | Course name | Replaces the existing course name. | `Uvod v sociologijo` |
| 4 | English course name | Optional. Written into the course translation field; an empty value clears it. | `Introduction to Sociology` |

**Sample**

```
SOC101;SOC-1001;Uvod v sociologijo;Introduction to Sociology
POL204;POL-2004;Primerjalna politika;Comparative Politics
;COM-3300;Mediji in komuniciranje;Media and Communication
```

**Notes.**

- **Trap:** any line whose first character is `K` is skipped. This is meant to skip a header, but it also silently drops every course whose old code starts with `K`. Rename or re-order such lines before importing.
- **Trap:** lines shorter than 10 characters are dropped without a warning, so very short code pairs never reach the database.
- Only rows with an empty first column create a course; every other row requires the course to already exist under its old code.
- On the first `CRSRECODE` run after the application starts, every course that has no code at all is listed on the console as a coverage check. The check does not repeat on later runs in the same session.
- Independent of the workspace selection.

## CRSTERMLECT — Computing lectures per term

**What it does.** For each line it finds the first course whose name contains the text from column 1, adds a lecture course part with a turn spanning the weeks of the given term, one hour per week, assigns the lecturer from the file, and allocates the group whose name equals column 1 exactly. It also adds that group's branch to the course. No schedule entry is placed — this import only builds the teaching load.

**How to start it.** `Data ▸ Custom import`, then type `CRSTERMLECT` at the prompt and pick the file.

**File format.** Semicolon-separated (`;`), one lecture assignment per line. No header line is expected and none is skipped. Four fields are read; anything after the fourth separator is ignored.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Course / group name | Matched as a **substring** against course names (first hit wins) and as an **exact** match against group names. | `Computer Science Year 1` |
| 2 | Number | Read but not used. | `30` |
| 3 | Term code | `MT` = weeks 1–8, `HT` = weeks 15–22, `TT` = weeks 29–36. Any other | `MT` |

| # | Column | Description | Example |
|---|---|---|---|
| | | value falls back to weeks 1–8. | |
| 4 | Lecturer | `Title. First Last`. Split at the first dot: everything up to and including the dot becomes the first-name field, everything from two characters after the dot becomes the surname. Without any dot the first-name field is set to `prof.` and the whole text becomes the surname. | `Dr. John Smith` |

**Sample**

```
Computer Science Year 1;30;MT;Dr. John Smith
Computer Science Year 1;30;HT;Prof. Anna Brown
Software Engineering Year 2;25;TT;Peter Clark
```

**Notes.**

- The lecturer is resolved by the split name/surname pair and created when not found, so a typo silently produces a new person.
- **Changed in this revision.** The character immediately after the dot used to be dropped, so `Dr.John Smith` gave the surname `ohn Smith`. The parser now skips the dot and any spaces after it, so both `Dr. John Smith` and `Dr.John Smith` resolve correctly.
- A **new** course part of type 1 (lecture) and a new turn are created on every line — existing course parts are never reused. Running the file twice duplicates the parts and turns.
- The turn gets one duration entry of `1` for the whole term period, no pauses, and the lecturer. Its turn part receives the matching group; if no group with exactly that name exists, the turn part stays without groups (no warning is shown).
- Only the first course whose name contains the text from column 1 is processed; if nothing matches, the line is silently ignored.
- No schedule entry is placed and nothing appears in the timetable grid; the result is visible as course load and group allocation.
- Independent of the workspace selection.

## CRSTURNGRP — Course, turn and group import

**What it does.** Reads a semicolon-separated list of teaching units and builds the whole teaching structure from it: it finds or creates the study programme (with a year-1 branch of the same name), finds or creates the groups named in the line, finds or creates the course and its course part for the given execution type, then creates a turn with the weekly duration, the lecturer and the group allocation. Consecutive lines that repeat the same course name, course code, execution type and group list are merged into the *same* turn and their hours are added together instead of creating a second turn. Turns are additionally restricted by day of week: programmes whose name starts with `MSc` are allowed on Saturday only, all other programmes are forbidden on Saturday and Sunday.

**How to start it.** `Data ▸ Custom import`, then type `CRSTURNGRP` at the prompt and pick the file.

**File format.** Plain text, one teaching slot per line, fields separated by a semicolon `;`. The separator is fixed, so the separator configured for the standard CSV imports is irrelevant and quoted fields are **not** supported. No header line is expected; a header would be imported as data unless its first character is `;`. Lines shorter than 2 characters and lines starting with `;` are skipped. Lines must not begin with whitespace. The file is transcoded to UTF-8 on the way in.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Course name | Course is matched by name, case-insensitively; created if it does not exist. | `Financial Accounting` |
| 2 | Course code | Up to two trailing digits are stripped before use. If the course already exists and does not contain this code, the code is appended to the existing one separated by a comma. | `ACC1001A` |
| 3 | Programme name | Programme matched by name; if missing, the programme is created with 1 year plus a branch of the same name in year 1. If the programme already exists, a year-1 branch with exactly this name must already exist. | `BSc Business` |
| 4 | Execution type | Matched case-insensitively against the execution types defined in the database. May be empty; empty or unknown falls back to type id 1. | `Lecture` |
| 5 | Number of students | Optional. If filled, it is written to the turn part as a custom student number; if empty, the custom student number is switched off. | `120` |
| 6 | Hours (weekly duration) | Duration string written into the turn's duration list for the whole week range. | `2` |
| 7 | Groups | Comma-separated list of group names. All spaces are removed first, a trailing comma is ignored. Missing groups are created under the year-1 branch with 14 students. Must not be empty. | `BSc1A,BSc1B` |
| 8 | (ignored) | Read past but ignored. The column must still be present so that column 9 lands in the right place. | |
| 9 | Lecturer | `Surname, Firstname`. If the field contains no comma, every space is treated as a separator, so the **first** word is taken as the surname and the second as the first name. Only the first two tokens are used. The lecturer is created if not found. | `Brown, Michael` |

### Sample

```
Financial Accounting;ACC1001A;BSc Business;Lecture;120;2;BSc1A, BSc1B;;Brown, Michael
Financial Accounting;ACC1001A;BSc Business;Lecture;120;1;BSc1A, BSc1B;;Brown, Michael
Marketing Principles;MKT2003B;MSc Marketing;Seminar;30;2;MSc1;;Taylor, Anne
```

### Notes.

- **Ordering inside the file matters.** The merge test compares the line only with the line immediately before it, so all lines belonging to one turn must be adjacent. In the sample above the two `Financial Accounting` lines produce one turn: the previous duration `2` and the new `1` do not add up to less than 3, so the duration becomes `2+1`. If the sum of the last digit of the previous duration and the new hours is below 3, the digits are added instead (`1` then `1` gives `2`).
- **Changed in this revision.** On a merged line the turn-part pointer was never assigned, yet the student number and the group allocation were written through it — two consecutive identical-key lines could corrupt memory or crash the application. The merge path now reuses the existing turn's first turn part, so consecutive lines belonging to one turn are safe.
- The week range comes from an advanced setting (formats `B-E` or a single week number); when that setting is empty the range is weeks 1–13. "No pauses" is switched off for every turn.
- The group column must not be empty: an empty column 7 makes the import create a group with an empty name. The same applies to column 9, where an empty value creates a lecturer with empty name and surname.

- At most 16 groups per line are handled.
- **Auto-created:** programmes, year-1 branches, groups, courses, course parts, turns, turn parts, lecturers. Execution types must already be defined if column 4 is to be honoured.
- The programme always comes from column 3, so the workspace selection is irrelevant.

## CURRICULUM — Full curriculum import: programs, branches, courses, turns and teaching load

**What it does.** This is the main structural import of its family. Each line describes one teaching activity: a course of a given program/branch/year, its execution type, semester, total hours, group size, weekly duration pattern and the teacher who runs it. The reader creates the program if it does not exist, creates the branch if a branch name is given, creates or reuses the course by name, creates or reuses the course part for the execution type, and then creates **one new turn** with a duration pattern and the teacher attached. After the whole file has been read, a post pass links courses that ended up without a branch.

**How to start it.** `Data ▸ Custom import`, then type `CURRICULUM` at the prompt and pick the file.

**File format.** Plain text, semicolon (`;`) separated, one line per teaching activity (course part turn). No quoting — no field may contain a semicolon. Read through the UTF-8 reader. **The first line is always skipped**, so a header row is expected (any first line will be discarded). Lines shorter than 2 characters and lines that start with `;` are skipped. 23 columns.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Program name | Looked up by name; created if it does not exist. | `Computer Science` |
| 2 | Program name (English) | Stored as the program translation, **only when the program is newly created**; may be empty. | `Computer Science` |
| 3 | Year | Numeric study year. Also raises the program's number of years when larger. | `1` |
| 4 | Branch name | Branch within the program for that year; created if it does not exist. **May be empty** — then no branch is attached and the post pass handles the course. | `Software Engineering` |
| 5 | Branch name (English) | Stored as the branch translation whenever column 4 is non-empty; may be empty. | `Software Engineering` |
| 6 | Course name | Key for finding the course — case-insensitive match against existing course names. Created if not found. | `Programming I` |
| 7 | (ignored) | Course name in English — parsed but never used. | `Programming I` |
| 8 | Course code | Set on a newly created course. On an existing course, appended to the stored code as `<old>,<new>` when it is not already contained in it. | `63201` |
| 9 | Elective marker | If the first character is a lowercase `i`, the course is flagged as selectable (elective). Anything else, including empty, leaves it non-elective. | `o` |
| 10 | Semester | If the first character is `Z`, the turn runs in weeks 1–16; **any other value, including empty**, puts it in weeks 21–35. | `Z` |
| 11 | Execution type | Matched case-insensitively against the names of the execution types already defined. If it does not match (or is | `Lectures` |

| # | Column | Description | Example |
|---|---|---|---|
| | | empty), execution type id 1 is used. | |
| 12 | (ignored) | Read past but not used. | |
| 13 | (ignored) | Parsed as “from week” but **never used** — the week range comes from column 10. | `1` |
| 14 | (ignored) | Parsed as “to week” but **never used**. | `15` |
| 15 | Total hours | Numeric; stored on the turn as its total (dynamic) hours. | `45` |
| 16 | (ignored) | Parsed as number of groups but **never used**. | `2` |
| 17 | (ignored) | Read past but not used. | |
| 18 | Number of students | Numeric. When greater than 0 the turn part gets this as a custom student count; when 0 or empty the custom count is switched off. | `120` |
| 19 | (ignored) | Read past but not used. | |
| 20 | Weekly duration | Session pattern per week, all spaces removed before use. A single number means one session of that many hours; `+` separates several sessions in the same week, e.g. `2+1`. | `3` |
| 21 | Teacher surname | Teacher is looked up by first name / surname and created if missing. | `Novak` |
| 22 | Teacher first name | Note the reversed order: surname comes before the first name. | `Ana` |
| 23 | Course note | Free text; when non-empty it **overwrites** the course note. | `Core course` |

**Sample**

```
Program;Program EN;Year;Branch;Branch EN;Course;Course
EN;Code;Elective;Semester;Type;;From;To;Hours;Groups;;Students;;PerWeek;Surname;Name;Note
Computer Science;Computer Science;1;Software Engineering;Software Engineering;Programming
I;Programming I;63201;o;Z;Lectures;;1;15;45;2;;120;;3;Novak;Ana;Core course
Computer Science;Computer Science;1;Software Engineering;Software Engineering;Programming
I;Programming I;63201;o;Z;Tutorials;;1;15;30;2;;60;;2+2;Kovac;Peter;
Computer Science;Computer Science;2;Data Science;Data Science;Machine Learning;Machine
Learning;63315;i;P;Lectures;;21;35;30;1;;45;;2;Horvat;Maja;Elective
```

**Notes.**

- **Auto-created:** programs, branches, courses, course parts, turns, turn parts and teachers. **Must already exist:** the execution types (column 11). An execution type name that is not already defined is not created — the line silently falls back to execution type id 1, which usually means everything lands under the first type.
- **Every line creates a new turn.** The course and the course part are reused across lines, but the turn is not. Running the same file twice therefore doubles all turns.
- The course lookup is by **name only, case-insensitive, across the whole schedule** — it is not scoped to the program or branch. Two different programs that use the same course name will share one course object; the program id and year written on the course are those of the last line that mentioned it, and its code column accumulates all the codes seen, comma-joined.
- Columns 13, 14 and 16 look meaningful but are dead: the week range is derived solely from the semester letter in column 10 (`Z` = 1–16, anything else = 21–35), and the group count is ignored. If your file genuinely needs other week ranges or a group split, this import will not honour them.
- The workspace selection is **not** used by this import.

- Column 20 is copied into a 64-character buffer; keep the weekly pattern short.
- **The post pass.** After the file is closed, every course in the schedule is checked. For a course with no branch attached (typically because column 4 was empty), all branches whose year and program match the course are linked to it. If no such branch exists at all, a branch named after the program is created for that program and year. **Changed in this revision:** that newly created branch was not linked back to the course, so such a course still ended up without a branch; it is linked now, which makes leaving column 4 empty a usable option. The post pass finishes by recalculating the teaching-load / allocation information.
- **Recommended order:** `CURRICULUM` first (programs, branches, courses, turns), then `LECMAILNOTE` for teacher e-mails and notes, then `LECTOCRS` and the `STUDBYCRS*` imports, which both need courses to exist and match them by code.

## EXTCRSIDS — Course external IDs

**What it does.** Attaches external system identifiers to courses that already exist. Each line is matched against the course list by course code (column 1, compared exactly after the file field has been trimmed), and the identifier from column 3 is stored in that course's source ID field. Only the first matching course is updated. Nothing is created and no other course data is modified.

**How to start it.** `Data ▸ Custom import`, then type `EXTCRSIDS` at the prompt and pick the file.

**File format.** Semicolon (`;`) separated, one course per line. No header line is expected — a header line simply matches nothing. Quoting is not supported. UTF-8 preferred (legacy ANSI is transcoded on the way in); `.xls`/`.xlsx` are converted to CSV first. Leading and trailing spaces are removed from every field. Lines shorter than 2 characters are skipped silently.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Course code | Compared with the course code stored in the application. Must match exactly (the file field is trimmed; the stored code is compared as it is, so a stored code with stray spaces will not match). | `BIO101` |
| 2 | Description | Read but not used. May be empty. | `Introduction to Genetics` |
| 3 | Source ID | External identifier written into the course's source ID field. | `88231` |

**Sample**

```
BIO101;Introduction to Genetics;88231
CHEM210;Organic Chemistry II;88245
MATH150;Linear Algebra;88260
```

**Notes.**

- Nothing is auto-created. A line whose code matches no course is **silently ignored** — no warning is logged, so a completely mismatched file still reports a successful import. Spot-check a few courses afterwards.
- The source ID is a separate field from the course code used by `MODCRSEVENTS` for lookup; this import never changes the course code itself.
- **Changed in this revision.** The field buffers were reused between lines, so a line with fewer than 3 columns stamped the previous line's source ID onto this course. Such a line is now skipped instead.
- Independent of the workspace selection.

## MODUL — Wide module export (programmes, years, execution types, hours)

**What it does.** Reads a wide spreadsheet export in which each line describes one teaching activity: the lecturer with a free-text note, the course with its code and execution type, the list of programme codes, the semester numbers it is taught in and the total number of hours. It finds or creates the lecturer and the course, resolves the programme codes to branches by the year derived from the semester numbers, attaches those branches to the course, creates or reuses the course part for the execution type, and creates a turn over weeks 1–18 with a duration derived from the hours column. One group per matching branch is allocated to the turn part.

**How to start it.** `Data ▸ Custom import`, then type `MODUL` at the prompt and pick the file.

**File format.** Semicolon-separated plain text with at least 30 columns per line, one activity per line. No header line is expected — a header must be commented out by starting the line with `;`. Only the columns listed below are read; the others must still be present so that the numbering stays correct.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Lecturer surname | The literal value `#N/A` makes the line be skipped. The literal value `0` is replaced by `tbd`. | `Novak` |
| 2 | Lecturer first name | May be empty. | `Ana` |
| 6 | Semester (fall/summer) | Read but not used. May be empty. | `fall` |
| 9 | Lecturer note | Optional. Appended to the lecturer's existing note on a new line. | `Prefers mornings` |
| 22 | Programme codes | Comma-separated list of programme codes. A code of exactly 4 characters is truncated to its first 3. Each code must exist. | `BUS,ECO` |
| 23 | Semester numbers | Digits; each digit `d` yields the year `(d-1)/2+1` (semesters 1–2 = year 1, 3–4 = year 2, …). Digit `0` is treated as 1. Empty or without digits means year 1. | `1` |
| 24 | Course name | Matched exactly, including case; created if not found. | `Introduction to Modules` |
| 25 | Course code | Used only when the course is created. | `MOD101` |
| 26 | Execution type | Must match an execution type name **exactly**, case-sensitively. An unknown value aborts the line with a warning. | `Lecture` |
| 27 | Groups | Read but not used. May be empty. | `1` |
| 30 | Hours | Total hours. A comma is accepted as decimal separator. | `4` |

### Sample

```
Novak;Ana;;;;fall;;;Prefers mornings;;;;;;;;;;;;;BUS,ECO;1;Introduction to
Modules;MOD101;Lecture;1;;;4
Smith;John;;;;summer;;;;;;;;;;;;;;;ECO;34;Applied Statistics;STA204;Seminar;2;;;8
0;;;;;fall;;;;;;;;;;;;;;;;BUS;2;Project Work;PRJ110;Lecture;1;;;0,5
```

### Notes.

- **Duration mapping from column 30:** values below 1 become `0.5`; 6 becomes `3+3`, 8 becomes `4+4`, 10 becomes `5+5`, 12 becomes `3+3+3+3`, 16 becomes `4+4+4+4`; every other value is written through unchanged as a single weekly duration.

- Programmes and branches are never created. A programme code that does not exist stops the line with a `Non-existing program` message and a warning — except when the code starts with `0`, in which case the line is skipped silently. Branches must already exist for the years derived from column 23.
- Only the **first** group found for each matching branch is allocated, so branches with several parallel groups are only partly covered.
- **Auto-created:** lecturers, courses, course parts, turns, turn parts. Everything else must already exist.
- Every line creates a new turn; course parts are reused per execution type.
- Semester numbers are read digit by digit, so a two-digit semester such as `10` is interpreted as the two digits `1` and `0` and yields year 1 twice.
- Fixed values written by this import: weeks 1–18 with pauses disabled.
- At most 32 programme years and 32 groups are handled per line.
- Lecturer notes accumulate: re-importing the same file appends the note text again.
- Independent of the workspace selection — programmes come from column 22.

## Placed timetables

These imports do not only create the teaching load — they also place it: a day, a start time and a room, either as a weekly pattern or as individual dated events. Everything they reference has to be in place before they run, and almost none of them de-duplicate, so importing the same file twice doubles the timetable. ## MODGRPB — Lecturer-centred module and timetable import

**What it does.** Produces the same result as `MODGRPM` — program, branch, course, course part, turn, turn part, group, room and optionally a locked schedule entry — but the line is laid out around the lecturer: the lecturer code is the first column and a free-text comment is appended to the lecturer's note. It additionally writes the credit points onto the course and decides master's versus undergraduate week settings from the branch name.

**How to start it.** `Data ▸ Custom import`, then type `MODGRPB` at the prompt and pick the file.

**File format.** Semicolon (`;`) separated, one line per teaching event. The separator is fixed and quoting is not supported, so no field may contain a `;`; commas inside a field are significant (they truncate the program and branch names and separate room properties and week ranges). Read through the UTF-8 reader; `.xls`/`.xlsx` files are converted to CSV first. A header line may be present: any line whose first column does not parse as a non-zero integer is skipped silently, as is any line starting with `;`.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Lecturer code | Must be a non-zero integer or the line is skipped without a warning. Overwrites the lecturer's code field | `1042` |
| 2 | Lecturer | `First Last`; everything after the first space is the surname | `John Smith` |
| 3 | Comment | Optional. Appended to the lecturer's note (existing note kept, new text added on a new line) | `Visiting lecturer` |
| 4 | Course code | Combined with column 11 into the effective course code `code:subject` | `BUS310` |
| 5 | Semester number | Study year is `(semester+1)/2`. The program's number of years is extended if needed | `3` |
| 6 | (ignored) | Not read | |
| 7 | Course name | Used only when no course carries the effective code yet | `Corporate Finance` |
| 8 | Credits | Written to the course as points | `6` |
| 9 | Program name | **Truncated at the first comma** | `Business Administration, Faculty of Economics` |
| 10 | Branch name | **Truncated at the first comma.** If the remaining text contains `(MA)` or `(MBA)` the line uses the master's week settings and group prefix. If empty, the program name is used | `Finance (MA), full-time` |
| 11 | Course subject / variant | Optional. Appended to the course code after a `:` | `A` |
| 12 | Term | `Fall` (case-insensitive) selects the autumn week settings; any other value, including empty, is treated as spring | `Fall` |
| 13 | Modality | Name of an execution type configured in the | `Lecture` |

| # | Column | Description | Example |
|---|---|---|---|
| | | application | |
| 14 | Day of week | English day name, see the accepted list in the Notes. May be empty | `Tuesday` |
| 15 | Duration in minutes | Converted to hours and then to 30-minute units | `90` |
| 16 | Start time | `HH:MM`, snapped to the nearest time slot. May be empty | `10:00` |
| 17 | (ignored) | Not read | |
| 18 | (ignored) | Not read | |
| 19 | Number of students | `0` or non-numeric becomes 1 | `35` |
| 20 | Room name | Optional. Created with 30 seats if unknown. Decides whether a locked schedule is written | `A-12` |
| 21 | Required room properties | Optional. Comma-separated list, added to the course part's prerequisites | `Projector` |
| 22 | Week list | Optional. Comma-separated relative week numbers/ranges (`1` = first week of the term). If empty, a default two-block list around the mid-term week is generated | `1-7,10-15` |

**Sample**

```
1042;John Smith;Visiting lecturer;BUS310;3;;Corporate Finance;6;Business Administration,
Faculty of Economics;Finance (MA), full-time;A;Fall;Lecture;Tuesday;90;10:00;;;35;A-
12;Projector;1-7,10-15
1117;Jane Doe;;BUS310;3;;Corporate Finance;6;Business Administration;Finance
(MA);A;Fall;Seminar;Thursday;90;14:00;;;18;;Projector;
```

**Notes.**

- **No group column.** The group name is always generated: `G-<year>-<branch>` normally, `GM-<year>-<branch>` when the branch name contains `(MA)` or `(MBA)`. The branch name used here is the part before the first comma.
- **Week settings.** Undergraduate lines use the "B" set of first/last/mid-term week settings, master's lines the "BM" set, separately for autumn and spring. Latent bug: for an autumn master's line with an empty week list, the first generated block ends at the **undergraduate** mid-term week while the second block starts from the master's one — supply column 22 explicitly if the two mid-term settings differ.
- **Auto-created:** rooms (30 seats), groups, lecturers and courses. The program is looked up by name and used without a null check.
- **Strictness.** No warning is produced for a skipped line (non-numeric first column, leading `;`, or no execution types configured). Only a lecturer that cannot be provided marks the line as a parse failure.
- **Day vocabulary.** Case-sensitive substring match. Accepted: `Monday`, `Mondey`, `Tueday`, `Tuesday`, `Wednesday`, `Wednsday`, `Thurday`, `Thursday`, `Friday`, `Saturday`, `Sunday`.
- **Modality vocabulary.** Must equal an execution type name exactly; an unknown value silently falls back to the first execution type. A modality containing `Lecture` disables group splitting.

- **Time snapping.** Nearest time slot wins, with no tolerance limit. Duration is minutes divided by 60, doubled and truncated to whole 30-minute units.
- **Room / start time combinations.** Both present — a locked schedule with all week flags `N`. Room empty and the day recognised — only the turn's preferred start hour and day flag. Room present and day unrecognised — the schedule gets an invalid day.
- **Group splitting.** Identical rule to `MODGRPM`: non-lecture modalities with more than one turn in the course part produce `-1`, `-2`, … sub-groups under the base group.
- **Known traps.** The start time, comment, course name, program and branch buffers are static and are **not** cleared between lines — a line with fewer than 22 columns silently reuses the previous line's start time and names, which can create schedules that are not in the file. Always write all 22 separators. Where `MODGRPM` checks the existing group for a null pointer before splitting, this reader does not, so a dangling group reference in the workspace can abort the import.
- **Ordering.** Run before `MODGRPA`/`AVAIL`, which match lecturers only by the code from column 1.

## CLASSLABSCH — Course, class and lab schedule import

**What it does.** Reads one already-scheduled class section per line and builds the whole chain: course (looked up by code, otherwise created from name + subject-area code), course part (Lecture or Lab, derived from the letter in front of the section suffix), turn with its weekly duration pattern, turn part with student count and allocated group, plus one placed schedule entry per weekday letter. Teachers, groups and rooms named in the line are created automatically if they do not exist yet. Start/end clock times are snapped to the nearest defined time slot, and the turn is placed over the globally configured import week range.

**How to start it.** `Data ▸ Custom import`, then type `CLASSLABSCH` at the prompt and pick the file.

**File format.** Semicolon-separated (`;`) plain text; quoting is **not** supported. Read as UTF-8; `.xls`/`.xlsx` is converted first. A header line is optional: a line beginning exactly with `Course Code` is skipped. Lines shorter than 2 characters and lines with an empty column 1 are skipped silently. Non-printable characters inside a field are replaced with spaces. One line = one section (one lecture group or one lab group) of one course.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Course code with type letter and section | Code, a trailing type letter (`L` = lecture, `B` = lab), and a `-00n` section suffix. If `-00` is missing, `-001` is appended automatically. The section digit must not be 0. The type letter is stripped before the course is looked up by code. | `AI701L-001` |
| 2 | Course name | Course title. Anything from `-Lab` onwards (case-insensitive) is cut off, so the lecture and the lab of one course resolve to the same course. | `Machine Learning` |
| 3 | Subject-area code | Must match the *Code* of a subject area, matched as a prefix, case-insensitive. The special value `ALL` (case-insensitive, may be part of a longer text) means "all groups" instead of one group. | `MML` |
| 4 | Weekdays | Letters `M`=Mon, `T`=Tue, `W`=Wed, `R`=Thu, `F`=Fri, `S`=Sat, `N` or `U`=Sun. Spaces are removed. One schedule entry is created per letter. May instead contain a plain number, which is then only used as the count of weekly meetings. | `MW` |
| 5 | Start time | `H:MM`, `HH:MM` or `HHMM`, with optional `AM`/`PM`; `PM` adds 12 hours. Snapped to the slot whose start time is closest. May be empty | `10:00 AM` |

| # | Column | Description | Example |
|---|---|---|---|
| | | — then slot 1 is used. | |
| 6 | End time | Same formats. Snapped to the slot whose end time is closest; that pair defines the duration. May be empty — then slot 4 is used. | `11:30 AM` |
| 7 | Maximum capacity | Number of planned students. Sets the group size when a new group is created and the turn part's custom student number. May be empty or 0. | `60` |
| 8 | Teacher(s) | One or more teacher names separated by `/`. Each name is split at the **last** space into first name + surname; a trailing comma on the first part means the name was given as `Surname, Firstname` and the two are swapped. Missing teachers are created. Should not be left empty. | `John Smith / Ana Kovac` |
| 9 | Elective flag | Anything whose first character is an uppercase `N` means "not elective". Any other value (including empty) marks the course as selectable/elective. | `Y` |
| 10 | Room(s) | One or more room names separated by `,`. Only the **first** name is used as the preferred room; the number of commas drives how many additional rooms are requested. Aliases are normalised: `LAB1`/`Lab1`/`L1`/`L 1` become `Lab 1` (…4), `CR 1` becomes `CR1` (…7), `LH 1` becomes `LH1` (…4). An unknown room is created with 30 seats. Must not be empty. | `CR1` |
| 11 | Note | Free text stored in the course note together with the planned student count. May be empty. | `Core course` |

**Sample**

```
Course Code;Course Name;Program;Days;Start;End;Capacity;Instructor;Elective;Room;Note
AI701L-001;Machine Learning;MML;MW;10:00 AM;11:30 AM;60;John Smith / Ana Kovac;Y;CR1;Core course
AI701B-001;Machine Learning-Lab;MML;R;2:00 PM;4:00 PM;40;Ana Kovac;N;Lab 1, Lab 2;
NLP702L-002;Natural Language Processing;MML;TR;9:00 AM;10:30 AM;35;Peter Novak;Y;LH1;
```

**Notes.**

- **Prerequisites.** At least one branch must be defined, otherwise the import does nothing at all. Every subject area used in column 3 must have its *Code* filled in; if it is missing and column 3 is not `ALL`, the line is skipped with a message. Time slots must be configured.
- **Auto-created:** courses, course parts, turns, turn parts, teachers, groups, rooms and schedule entries. **Must already exist:** subject areas (with codes) and time slots.
- **Weeks come from the settings, not from the file.** Turn and schedule week range is always the global import week range.
- **Re-import behaviour.** When a matching course part already exists, its turns covering exactly that same week range are deleted first — unless they are flagged as "newly imported" by the current run. **Changed in this revision:** that flag reset used to run only when line 1 was *not* the `Course Code` header, so re-importing a file with a header in one session left the previous run's turns flagged and duplicate turns accumulated. The reset now happens once per run, header or not, so re-importing replaces instead of duplicating.
- **Course part type.** The letter immediately before `-00n` decides: `L` = lecture (room property *Classroom* required), `B` = lab (room property *Lab* required). Any other letter falls back to lecture.

- **Duration.** Weekly hours = (end slot − start slot + 1) / 2, repeated once per weekday letter (maximum 3 terms) into the duration pattern, e.g. `1.5+1.5`. The original text of columns 5 and 6 is preserved on the schedule entry as `{10:00 AM - 11:30 AM}`.
- `ALL` **in column 3.** Instead of one group, the turn part is allocated to every existing group whose name starts with the same letter as the value in column 3 — i.e. with the literal value `ALL`, every group whose name starts with `A`. Use it only if that is genuinely what you want.
- **Additional rooms.** Listing several comma-separated rooms in column 10 requests that many additional rooms (capped at 3, with a warning), and rebalances the per-room student count. **Changed in this revision:** the automatic "lab too big, split it" path could never run — its room-name test used `||` where a character cannot be both `L` and `l`, so the condition was always true and the split was cancelled every time. A lab whose capacity exceeds 20 students is now really split across additional rooms, so lab lines with a large capacity will produce more rooms than they did before.
- **Empty room column.** Column 10 must contain a room name. If it is empty, no room object exists and the schedule-entry creation dereferences it.
- **12-hour times.** `12:xx PM` becomes 24:xx and `12:xx AM` stays 12:xx. Avoid noon/midnight values with an `AM`/`PM` suffix, or write them in 24-hour form.
- **Teacher names.** A teacher entry with no space in it is not handled cleanly. Always give first name and surname.
- The elective flag only recognises an uppercase `N`; a lowercase `no` is read as "elective".

## DATEDLECT — Dated lecture schedule with one group per line

**What it does.** Imports dated teaching events; each line produces one schedule entry in the week that contains the given date. The course is matched by exact name and created if unknown, the lecturer is created if unknown, and the room is created if unknown. Turns are reused when an existing turn of the course has the same lecturer and already contains the group, in which case the turn's week span is widened to include the new date.

**How to start it.** `Data ▸ Custom import`, then type `DATEDLECT` at the prompt and pick the file.

**File format.** Semicolon (`;`) separated, one event per line, 8 columns. No header line is required; lines shorter than 10 characters and lines beginning with `#` or `;` are skipped, so a comment-style header is tolerated. UTF-8; `.xls`/`.xlsx` is converted to CSV automatically.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Date | `d.m.yyyy`; two dots required, otherwise the line is rejected. | `5.10.2026` |
| 2 | Start time | `H:MM`; the nearest defined time slot is used as the begin slot. | `08:00` |
| 3 | End time | `H:MM`; the length is rounded to the nearest 30 minutes. | `09:40` |
| 4 | Course name | Exact name match; created (execution type 1, one lecture course part) if not found. | `Calculus I` |
| 5 | Lecturer first name | Passed as the first name; the lecturer is created if new. | `Andi` |
| 6 | Lecturer surname | Passed as the surname. | `Wijaya` |
| 7 | Room | Room name; created automatically if unknown (web-visible, 20 seats). | `Building West 2201` |
| 8 | Group | Group name; **must already exist with exactly this name**, otherwise the line is skipped without a warning. | `IF-1` |

**Sample**

```
5.10.2026;08:00;09:40;Calculus I;Andi;Wijaya;Building West 2201;IF-1
5.10.2026;10:00;11:40;Physics I;Rina;Sutanto;Building West 2202;IF-2
```

**Notes.**

- One schedule entry is created per line, with begin week = end week = the week of the date. Running the same file twice produces duplicate schedules.
- Auto-created: courses (with one lecture course part), lecturers, rooms, turns and turn parts. Must already exist: groups, time slots and the week calendar.
- An unknown group ends the line quietly (it counts as read, not as a warning), so a typo in column 8 silently loses the event.
- When a **new** turn is created, the group is looked up a second time with a substring match and the first partial hit is attached. If several group names contain the same text, the turn may get a different group than the one matched exactly.
- The turn's duration list is filled with the duration divided by 2 using integer division, so a 90-minute event is recorded as 1 hour for that week.
- **Changed in this revision:** turn matching read the first lecturer of a turn without checking that the turn had any, which read past the end of an empty list; such turns are skipped now. Still true: an existing course found by name is assumed to have at least one course part.
- Independent of the workspace selection.

## DATEDWEEKLY — Dated events collapsed into weekly turns

**What it does.** Reads a list of dated events (date, time span, course code, up to four teachers, room, up to ten groups) and turns each of them into a course part / turn / turn part plus one scheduled entry in the timetable grid. The date is used only to derive the weekday; the week span of every created turn and schedule is taken from the import code itself, not from the file. Missing rooms are created automatically, and lines whose groups already exist in another turn of the same course are merged into that turn instead of producing a second schedule.

**How to start it.** `Data ▸ Custom import`, then type `DATEDWEEKLY1-17` at the prompt (see the week-range syntax in the Notes) and pick the file.

**File format.** Semicolon (`;`) separated, one event per line, no header line required (lines starting with `#` and lines shorter than 10 characters are skipped). Read through the UTF-8 reader; `.xls`/`.xlsx` files are converted to CSV first. **Column 1 is not read at all** — the first value on the line is discarded, so the file must carry a leading key/ID column.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | (ignored) | Any value; the parser skips this field entirely. Must be present as a placeholder. | `1` |
| 2 | Date | `d.m.yyyy`; two dots required, otherwise the line is rejected. Only the weekday is used. | `5.10.2026` |
| 3 | Start time | `H:MM`. The nearest time slot start is used as the begin slot. | `09:00` |
| 4 | End time | `H:MM`. | `12:00` |
| 5 | Course code | Looked up / provided by code. The course execution type is set to 1. | `KIP-201` |
| 6 | Teacher 1 | `First Last` in one field, split at the **first** space. Rejected if it contains no space. May be empty. | `Ana Novak` |
| 7 | Teacher 2 | Same format. Optional, may be empty. | |

| # | Column | Description | Example |
|---|---|---|---|
| 8 | Teacher 3 | Same format. Optional, may be empty. | |
| 9 | Teacher 4 | Same format. Optional, may be empty. | |
| 10 | Room | Room name; created automatically if unknown (web-visible, 20 seats). | `Building A Lecture Room EC 7` |
| 11 | Group 1 | Group name; **must already exist**, otherwise the line is rejected. | `Sculpture BA1` |
| 12–20 | Group 2 … Group 10 | Further groups, optional, may be empty. Fields beyond column 20 are ignored. | `Design MA1` |

**Sample**

```
1;5.10.2026;09:00;12:00;KIP-201;Ana Novak;;;;Building A Lecture Room EC 7;Sculpture BA1;
2;6.10.2026;13:00;16:00;GRA-105;Marko Kos;Eva Zupan;;;Building B Lecture Room 111;Design
MA1;Design MA2
```

**Notes.**

- **Week-range syntax.** The dispatcher matches any typed code that *contains* `DATEDWEEKLY` (the text is upper-cased first, so `aluo1-17` works too). The default range is weeks 1–17. If the typed code contains a `-`, the start week is read from the 5th character of the code — directly after `DATEDWEEKLY` — and the end week from the text after the `-`. So `DATEDWEEKLY1-17` = weeks 1–17, `DATEDWEEKLY5-12` = weeks 5–12, plain `DATEDWEEKLY` = weeks 1–17. Because the start is read from a fixed offset, the code must literally begin with `DATEDWEEKLY` and the number must follow immediately — `DATEDWEEKLY-12` is parsed as start week −12. The prompt accepts at most 16 characters.
- Every line of the file gets the same week range; the date in column 2 only supplies the weekday.
- **Group names are normalised** before lookup: `BA 1`/`BA 2`/`BA 3` and `MA 1`/`MA 2`/`MA 3` become `BA1`…`MA3` and the token is forced to be preceded by exactly one space. This normalisation overwrites in place, so **any text after the level token is discarded** (`Design BA1 A` becomes `Design BA1`). Group names in the file should therefore end with the level token.
- Three room names are corrected in code before lookup (a lower-case `predavalnica` is capitalised; `EC14` becomes `EC 14`; `EC7` becomes `EC 7`).
- Courses and rooms are auto-created; **groups and time slots must already exist**. A line with no non-empty group column is skipped silently.
- **Behaviour depends on what already exists for the course:** if the course part has no turn yet, a turn with all teachers (columns 6–9) and all groups is created together with one schedule. Otherwise only the teacher from column 6 is used; if the line carries more than one group, the groups and that teacher are merged into the first existing turn part and **no new schedule is produced**. With exactly one group, an existing schedule in the same room, at the same slot and on the same weekday belonging to the same course is reused (again with no new schedule); otherwise a new turn, turn part and schedule are created.
- **Changed in this revision — check your expectations.** The duration was computed by dividing the minute difference by **15** while the grid works in 30-minute slots, so every event was placed twice as long as it should be and the turn duration list held twice the hours. It is now on the 30-minute basis: a 90-minute event is 3 slots and 1.5 hours. Timetables imported with an older build are twice as long as they should be and have to be re-imported.
- Still true: a line can be rejected *after* the room, course, turn and turn part have already been created, leaving partial data behind. A flag in the single-group branch can never be set (the code that would set it is commented out), so the tail of the function is dead code.

- Does not depend on the workspace selection. `-d` and `-f` have no effect.

## FULLTT / FULLTTSTART — Full timetable with week lists, day lists and per-day durations

**What it does.** Imports a complete timetable in a compact form: one line describes a course delivery with a list of weeks, a list of weekdays, a list of hours per day and a list of groups, and the import creates one turn per week entry, one turn part with all the groups, and one schedule per weekday instance. Before the first line of each course/branch combination it deletes the existing schedules and turns of that course for that branch, so the import replaces rather than extends the timetable of the courses it touches.

**How to start it.** `Data ▸ Custom import`, then type `FULLTTSTART` for the first file and `FULLTT` for every following file in the same session, then pick the file.

**File format.** Semicolon (`;`) separated, 14 columns, one delivery per line. Lines shorter than 10 characters and lines starting with `;`, `#` or `"` are skipped, as is a line whose first cell is exactly `Predmet` — so a header line with that caption is dropped automatically. UTF-8 (Slovene day names are matched); `.xls`/`.xlsx` is converted automatically.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Course name | Only used to recognise the header (`Predmet`); the course itself is found by code. | `Analiza I` |
| 2 | Course code | Primary code; the course **must already exist** with this code. | `PARTTIME-101` |
| 3 | Alternative course code | Used only when column 2 is empty. May be empty. | |
| 4 | Weeks | Comma-separated list of single weeks and ranges `a-b`. Each entry produces one turn. Must start with a digit or the line is skipped. | `1-15` |
| 5 | Weekdays | Comma-separated Slovene day names, case-insensitive: `ponedeljek`, `torek`, `sreda`, `četrtek` (or `cetrtek`), `petek`, `sobota`, `nedelja`. Position n corresponds to duration entry n. | `ponedeljek,sreda` |
| 6 | Durations per day | `+`-separated hours, one entry per weekday, in the same order. Each hour becomes 2 half-hour slots; an entry of `0` skips that day. The whole string is also stored as the turn's duration. | `2+2` |
| 7 | Hours | Comma-separated `start-end` in whole hours, `:30` allowed on either side; one entry per weekday, same order. | `10-12,08-10` |
| 8 | Teaching type | Must match a modality (execution type) name, compared case-insensitively; otherwise the line is skipped with a log entry. | `Predavanje` |
| 9 | Room | Room name; must already exist (see the trailing-number retry in the Notes). | `P-01` |
| 10 | Groups | Comma-separated group names; each must already exist. Unknown names are logged and ignored; if none resolves, the line is skipped. | `PARTTIME 1. letnik` |
| 11 | Lecturer surname | | `Novak` |

| # | Column | Description | Example |
|---|---|---|---|
| 12 | Lecturer first name | If empty, the lecturer in columns 11–12 is ignored entirely. | `Ana` |
| 13 | Second lecturer surname | Optional, may be empty. | |
| 14 | Second lecturer first name | Optional; if empty the second lecturer is ignored. | |

**Sample**

```
Predmet;Šifra;Šifra2;Tedni;Dnevi;Ure;Termini;Izvedba;Prostor;Skupine;Priimek;Ime;Priimek2;I
me2
Analiza I;PARTTIME-101;;1-15;ponedeljek,sreda;2+2;10-12,08-10;Predavanje;P-01;PARTTIME 1.
letnik;Novak;Ana;;
Fizika I;PARTTIME-105;;1-15;torek;3;08-11;Vaje;L-02;PARTTIME 1. letnik,PARTTIME 1. letnik
B;Kos;Marko;;
```

**Notes.**

- **Variants.** The only difference is that `FULLTTSTART` resets, at line 1, the internal list of course-code + branch pairs that have already been cleaned. That list controls the deletion described below, so **run `FULLTTSTART` for the first file of an import session and `FULLTT` for every subsequent file**; starting a second file with `FULLTTSTART` would wipe the courses imported from the first file again.
- **What is deleted:** the first time a given course code together with a given branch appears in the session, all existing schedules of that course belonging to that branch are removed, then every turn of the course whose turn parts do not belong to another branch is deleted, and course parts left without turns are removed. Courses not mentioned in the file are untouched.
- **Must already exist:** the course (by code), the room, all groups, the modality named in column 8 and both lecturers. Nothing is auto-created. Courses whose "filter" flag starts with `N` are skipped.
- The branch used for the cleanup is the branch of the last group resolved in column 10.
- **Room fallback:** if no room matches exactly and the name ends with a digit, a space is inserted before that trailing number and the lookup is retried (`P01` becomes `P 01`); if that fails too, the line is skipped with a log entry.
- **Slot numbering for this import:** 07:00 is slot 0, each slot is 30 minutes, so `10-12` becomes begin 6, end 9. If the requested duration is longer than the hour window, the turn is marked "no pauses" and the schedule length is reduced according to a fixed table (4 to 3, 5 to 4, …, 10 to 8, 12 to 9).
- **Changed in this revision.** (1) Column 6 was consumed destructively while the first week entry was processed, so with more than one week entry in column 4 the second and later turns got only the first duration segment and a single schedule; every week entry now parses a fresh copy and produces its full set of schedules. (2) The guard comparing the day index with the number of weekdays was off by one and read past the day array.
- Still true: a lecturer whose first-name cell is empty is silently dropped even when the surname is filled.
- Does not depend on the workspace selection.

## GRIDLEGEND — Dated event grid plus its course/lecturer legend (two passes)

**What it does.** Reads one spreadsheet export that contains two blocks: at the top a dated timetable grid (one line per concrete event), at the bottom a legend listing every course with its code and its lecturers. The file is read twice: the **first pass** skips everything until the legend header and then creates/updates the lecturers and the courses (course code + course name), attaching each course to the Branch currently selected in the workspace; the **second pass** reads the dated grid at the top, stops when it reaches the legend, and for every dated line creates a turn, a turn part and a placed schedule entry in the given room, week, day and start slot. Rooms that do not exist are created with 30 seats; courses and lecturers are *not* created in the second pass — they must have been produced by the first one.

**How to start it.** `Data ▸ Custom import`, then type `GRIDLEGEND` at the prompt and pick the file.

**File format.** Semicolon-separated; the separator is fixed. UTF-8 or legacy ANSI. No header line is expected. **Column 1 must be empty on every line** — the date scan only looks at the first 12 characters and mis-reads a date placed in column 1. One line in the top block is one dated event; one line in the bottom block is one course with up to three lecturers.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | (empty) | Must be empty — the parser expects the date to start at character index 1. | |
| 2 | Date | `dd.mm.yyyy`; the whole `;dd.mm.yyyy;` must fit in the first 12 characters. Week number and weekday are derived from it. A line without exactly two dots here is silently skipped. | `06.10.2026` |
| 3 | Half day | Exactly `DOP`, `POP`, `morning` or `afternoon`. Anything else (or both) makes the line be skipped. | `morning` |
| 4…n | (empty) | At least one empty column must follow the half-day column; the course cell is taken after the first `;;` that follows it (up to three further leading `;` are skipped). | |
| n+1 | Course cell | `[I - \|E - ]<course code> - <lecturer surname> (<hours> <type>), <room>`. `I -` / `E -` (case-insensitive) marks the course as selectable (elective). | `E - MNG205 - Novak (2 V), C3` |
| — | …course code | Text before the first `-` in the cell; must match a course created by the first pass. | `MNG101` |
| — | …lecturer surname | Text after `-`, cut at `(`; matched case-insensitively against the lecturers collected in the first pass, then against all lecturers. | `Stubelj` |
| — | …hours | Number right after `(`; doubled internally. Missing `(` means type `P` and 4 hours. | `3` |
| — | …type | Letter after the hours; `L` is stored as `P`, `T` as `V`, anything else as written. Created in the execution-type list if unknown. | `P` |
| — | …room | Text after `),` up to the next `;`. If there is no `),`, the last word before `ob` or before `;;` is used. `sk. A,` / `sk. B,` anywhere in the line select the A/B student group. A leading `at` is stripped up to `,`. | `C2` |
| — | Legend header | `;OBVEZNI PREDMETI`, `;IZBIRNI PREDMETI`, `;(L - lectures`, `;ELECTIVE COURSES` or `;ELECTIVE  COURSES` — starts the legend block. | `;OBVEZNI PREDMETI` |
| — | Legend line | `;<course name> (<course code>) - <Name Surname> (<hours> <type>)[, <Name Surname> (<hours> <type>)][,` | see sample |

| # | Column | Description | Example |
|---|---|---|---|
| | | `…]` — up to three lecturers. | |

**Sample**

```
;06.10.2026;morning;;;;MNG101 - Stubelj (3 P), C2;;;;;;
;07.10.2026;afternoon;;;;E - MNG205 - Novak (2 V), C3;;;;;;
;OBVEZNI PREDMETI;;;;;;;;
;Marketing Management (MNG101) - Igor Stubelj (3 P);;;;;;;;
;IZBIRNI PREDMETI;;;;;;;;
;Operations Research (MNG205) - Igor Stubelj (2 P), Andrej Novak (2 V);;;;;;;;
```

**Notes.**

- **Order inside the file is mandatory.** The legend must come *after* the timetable grid: the second pass stops at the first line containing `;OBVEZNI PREDMETI`, `;;;EXAMS;;;` or `;(L - lectures,`. If the legend only uses `IZBIRNI PREDMETI` as its header, the second pass does not stop and will try to parse legend lines as events. The first pass stops after four consecutive lines beginning with nine semicolons.
- **Workspace dependency.** The Branch selected in the workspace is used twice: the first pass adds it to every created course, and the second pass allocates each turn part to a group of that branch. **Groups are never created** — a group of the selected branch must exist whose name does *not* end in `A`/`B` (normal case) or ends in `A`/`B` (when `sk. A,` / `sk. B,` is present). No matching group means the line is skipped.
- **Auto-created:** lecturers and courses (first pass), rooms (second pass, 30 seats), execution types, turns, turn parts and schedule entries. **Must already exist:** programs/branches, student groups.
- **Start slots are hard-coded:** morning gives slot 5 (slot 3 when the cell says 4 hours), afternoon gives slot 12 regardless of length. Real clock times in the file are ignored.
- **Turn reuse:** an existing turn with the same first lecturer, first group and preferred room is extended (week range widened, hours summed); otherwise a new turn is created. A schedule entry is nevertheless added for every line with no duplicate check, so re-running the import on the same file duplicates all placed events.
- **Silent skips.** Only a negative return value is counted as a warning; this reader returns a positive value for "bad line", "lecturer not defined in the legend", "unknown course code" and "no matching group", so those lines are counted as successfully read and the final popup shows zero warnings. Check `wtt_event.log` after every run.
- **Changed in this revision.** Three defects: the "legend started" marker and the collected lecturer list were static and carried over into the next import in the same session (they are reset on line 1 now); a legend entry without the `(hours type)` suffix, and a course cell without `-`, crashed the application instead of warning (both are reported and skipped now).
- Still true: when the execution type in the cell is new it is appended to the type list but the newly created id is not used — define the types (`P`, `V`, …) before importing. The first pass also contains a hard-coded surname normalisation folding two spellings of one surname into one.

## MEDCLASS — Medical lectures (timetabled events)

**What it does.** Imports the actual timetabled events for the medical programmes. Each line resolves a course by code, a teaching modality by code, a lecturer by code and a room by code, works out the groups from a group specification, then creates or reuses a course part and a turn and places one schedule entry at the given term/week/day/time. Notes and historical notes from the file are appended to the course note, and the course credit points are set from the file. Nothing is auto-created here — everything referenced must already exist, which is why this import runs last.

**How to start it.** `Data ▸ Custom import`, then type `MEDCLASS` at the prompt and pick the file.

**File format.** Semicolon-separated (`;`), one event per line. A header line is skipped automatically: any line shorter than 10 characters, or whose first field is not a number, is ignored. At least 15 separators must be present; the last field is the rest of the line. Fields are used as written (no trimming) except the notes fields.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Term | `1`, `2` or `3`. Sets the week offset and the turn's week span (term 1 = weeks 2–9, term 2 = 16–23, term 3 = 31–39). Must be numeric and non-zero or the line is treated as a header and skipped. | `1` |
| 2 | Week within term | Integer. Absolute week = term offset + this value (term 1: +1, term 2: +15, term 3: +30). `0` or empty is logged and treated as week 1. | `3` |
| 3 | Day | Day number, 1-based in the file; stored as value − 1. | `2` |
| 4 | Start time | `HH:MM`. Converted to slot `(HH − 7) × 2`, plus one when the minutes are non-zero. | `09:00` |
| 5 | End time | `HH:MM`, converted the same way; the duration is the slot difference. | `11:00` |
| 6 | Room code | Must match the code of an existing room (see `MEDVENUE`). | `LT1` |
| 7 | Teaching type code | Must match the **code** of an existing execution type / modality; the line is skipped when unknown. | `LEC` |
| 8 | Lecture name | Used as the prefix of the note text appended to the course. | `Cardiac cycle` |
| 9 | Groups | Optional, may be empty. Comma-separated list of group suffixes and ranges (see Notes). Empty means "the master group of each branch of the course". | `A` |
| 10 | Lecturer code | Must match the code (initials) of an existing tutor (see `MEDLECT`); the line is skipped when unknown. | `JDS` |
| 11 | Notes | Optional. Appended to the course note as `<lecture name>: <notes>`. | `Bring the handout` |
| 12 | Historical notes | Optional. Appended to the course note as `<lecture name>(Hist.): <text>`. | `Was 2 hours until 2019` |
| 13 | Course code | Must match an existing course code (see `MEDPROG`); the line is skipped when unknown. | `CVS101` |
| 14 | Course name | Read but not used — the course is identified by the code in column 13. | `Cardiovascular System` |
| 15 | Course points | Integer; overwrites the course credit points. | `10` |
| 16 | Cohort percent | Rest of the line, read as a number but not used further. | `100` |

### Sample

```
Term;Week;Day;Start;End;Venue;Type;Lecture;Groups;Lecturer;Notes;History;Course;Course name;Points;Cohort
1;3;2;09:00;11:00;LT1;LEC;Cardiac cycle;;JDS;Bring the handout;;CVS101;Cardiovascular System;10;100
```

```
1;3;4;14:00;16:00;SR2;PRAC;Heart sounds practical;1-4;AMB;;Was 2 hours until
2019;CVS101;Cardiovascular System;10;50
```

**Notes.**

- Nothing is auto-created: course (by code), modality (by code), lecturer (by code) and groups must all exist. A missing course, modality, lecturer or group means the line is logged and skipped. A missing **room** is different: the course part and the turn are still created and the group allocation still happens, but no schedule entry is placed — the event exists unplaced, with the missing room code in the log.
- **Ordering:** `MEDPROG` (courses with codes) » `MEDVENUE` (room codes) » `MEDLECT` (lecturer codes) » `MEDCLASS`. Groups must exist before `MEDCLASS` as well.
- **Group specification (column 9):** the list is split on commas. A plain token `X` becomes the suffix `_X`. A numeric range `1-4` expands to `_1`, `_2`, `_3`, `_4`; a non-numeric range such as `A-D` expands character by character to `_A` … `_D`. Candidate groups are all groups whose branch is one of the course's branches and which are sub-groups; a group matches when the text from the first `_` in its name onwards equals the suffix, e.g. group `Year1_A` matches token `A`. When the column is empty, the first master group of every branch of the course is used instead.
- **Changed in this revision.** After the first token the group-list parser advanced its pointer into the raw line buffer instead of into the group string, so only the **first** comma-separated token was read. All tokens are read now — a line listing several groups will allocate more groups than it did before.
- **Turn reuse:** an existing turn in the same course part is reused when its first tutor, its begin week and its recorded duration match and its turn part has the same number of groups with the same first group id. In that case the durations are summed; otherwise a new turn and turn part are created with the term's week span and the room as the preferred room. The comparison relies on a temporary duration field that is only filled while a turn is created, so reuse works within one import run, not across runs.
- **Course notes accumulate:** notes and historical notes are appended on every run while the note is shorter than 4000 characters, so repeated imports grow the note. The course points are overwritten on every line.
- The time grid is assumed to start at 07:00 with two slots per hour; times outside that grid produce negative or oversized slot indices.
- Does not use the workspace selection.

## MODCRSEVENTS — Dated events for modular courses

**What it does.** Imports individual dated events (date + start/end clock time) and turns each one into a real placed entry in the schedule. For every line it locates the course by code, selects or creates the course part matching the execution modality code, then creates one turn, one turn part and one placed schedule entry per room listed on that line, with all listed tutors attached to the turn. The event is stored as a single-week entry carrying the exact clock times, the date string and the event title.

**How to start it.** `Data ▸ Custom import`, then type `MODCRSEVENTS` at the prompt and pick the file.

**File format.** Semicolon-separated (`;`), one event per line. No header line is expected — a header line fails to resolve a course and is counted as a warning. Quoting is not supported. Lists inside columns 7 and 8 are comma-separated. UTF-8 preferred; `.xls`/`.xlsx` are converted to CSV first. Leading and trailing spaces are removed from every field. Lines shorter than 2 characters are skipped silently.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Date | `dd.mm.yyyy`. A 2-digit year is accepted and gets +2000. The | `5.10.2026` |

| # | Column | Description | Example |
|---|---|---|---|
| | | year part may be omitted entirely, but then no trailing dot may be written (`5.10`, not `5.10.`); the school year is then derived from the current school year start. Mandatory. | |
| 2 | Start time | `hh:mm`; the colon is mandatory. Snapped to the nearest grid label at or after this time. | `08:00` |
| 3 | End time | `hh:mm`; the colon is mandatory. Snapped to the nearest grid label at or before this time. | `09:30` |
| 4 | Course code | Code of an existing course. The course must already be marked as modular (dated events); a periodic course is rejected. | `BIO101` |
| 5 | Execution modality code | Must equal the code of one of the execution types configured in the application. The matching course part is reused, or created if the course does not have one yet. | `LEC` |
| 6 | Event title | Free text, stored on the schedule entry and in its display flags. May be empty. | `Introduction to Genetics` |
| 7 | Tutor codes | One or more tutor codes separated by commas. Every code must resolve to an existing tutor. Must not be empty. | `JSMITH,MJONES` |
| 8 | Room codes | One or more room codes separated by commas, **or** `@` followed by a room property name (see Notes). Every code must resolve to an existing room. Must not be empty. | `B-204,B-205` |

### Sample

```
5.10.2026;08:00;09:30;BIO101;LEC;Introduction to Genetics;JSMITH;LH-A
5.10.2026;10:00;13:00;BIO101;LAB;Microscopy practical;JSMITH,MJONES;LAB-B,LAB-C
6.10.2026;14:00;15:30;CHEM210;SEM;Organic chemistry seminar;RBROWN;@Projector
```

### Notes.

- Nothing is auto-created except course parts, turns, turn parts and schedule entries. Courses, execution modalities, tutors and rooms must all exist beforehand; **rooms and tutors are matched by code, not by name**. Run `ROOMCODES` and `EXTLECIDS` — or set the codes manually — before this import.
- Multiple rooms on one line do not produce one shared event: each room code creates its own turn, turn part and schedule entry, all with the same course part, tutors, date, time and title.
- `@` **room selection:** if column 8 begins with `@`, the rest is read as a room property (equipment) name and is expanded into the codes of every room carrying that property. The line is rejected if the property name is not found, or if no room has it. A matched room without a code logs a warning but still contributes an empty entry to the list, which then fails the code lookup and aborts the line — make sure every room with that property has a code.
- **Strictness:** an unresolvable course, room or tutor code, a non-modular course, a bad date, a missing `:` in a time, a week outside 1–52, or an empty tutor/room list rejects the line; it is logged and counted as a warning, and the rest of the file continues. An unknown execution modality code (column 5) is rejected **without any explanatory log entry** — if lines fail with no visible reason, check that column first.
- **Partial-line trap:** rejection can happen after some of the room codes on that line have already been processed, so the turns and schedule entries created for the earlier codes stay in the schedule. Likewise, an empty tutor list is detected only after the turn for the current room has been created, leaving an empty turn behind.

- Re-running the file always creates new turns and new schedule entries — existing ones are never matched or replaced.
- Duration is derived from the grid: end label minus start label plus one slot. Every created schedule entry gets an all-`N` week flag string and the single week computed from the date, plus the exact clock times, the date text exactly as written in column 1, and a display string of the form `{(08:00 - 09:30) Introduction to Genetics}`.
- Depends on the currently open school year (used for week numbering and for guessing the year when column 1 has no year), not on the workspace selection.

## MODGRPM / MODGRPMI — Modules, groups and locked timetable events

**What it does.** Reads one teaching event per line and builds the whole academic structure needed for it: program, branch (study year derived from the semester number), course, course part per modality, turn, turn part, student group, lecturer and room. When both a room name and a start time are given, it also writes a locked schedule entry for the week ranges of the line; when the room is empty it only stores the day and start hour as a preference on the turn. `MODGRPM` and `MODGRPMI` run the same reader with an *intake* flag of 0 and 1 respectively, which changes the default group-name prefix and the set of term-week settings used.

**How to start it.** `Data ▸ Custom import`, then type `MODGRPM` (or `MODGRPMI`) at the prompt and pick the file.

**File format.** Semicolon (`;`) separated, one line per teaching event. The separator is fixed, so it does **not** follow the general CSV separator setting, and quoting is not supported. Read through the UTF-8 reader; `.xls`/`.xlsx` files are converted to CSV first. A header line may be present: any line whose first column does not parse as a non-zero integer is skipped silently, as is any line starting with `;`.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Row number | Content unused, but must be a non-zero integer, otherwise the whole line is skipped without a warning | `1` |
| 2 | Semester number | Study year is computed as `(semester+1)/2`, so 1–2 = year 1, 3–4 = year 2, … The program's number of years is extended if needed | `3` |
| 3 | Group name | Optional. If empty, the group name is generated (see Notes) | `G-2-Software Engineering ENG` |
| 4 | Course code | Combined with column 11 into the effective course code `code:subject` | `CS201` |
| 5 | Lecturer | `First Last`; everything after the first space is the surname. If there is no space the whole value is the first name and the surname is empty | `John Smith` |
| 6 | Lecturer code | Optional. Overwrites the lecturer's code field. Must be a non-zero number if `MODGRPA`/`AVAIL` will be used later | `1042` |
| 7 | Program name | | `Computer Science` |
| 8 | Branch name | If it contains `ENG` (case-insensitive) the line is treated as the English stream. If empty, the program name is used | `Software Engineering ENG` |
| 9 | Term | `Fall` (case-insensitive) selects the autumn week settings; **any other value**, including an empty one, is treated as spring | `Fall` |
| 10 | Course name | Used only when no course with the effective code exists yet | `Algorithms and Data Structures` |

| # | Column | Description | Example |
|---|---|---|---|
| 11 | Course subject / variant | Optional. Appended to the course code after a `:` | `A` |
| 12 | Lecture hours | Parsed but not used | `30` |
| 13 | Seminar hours | Parsed but not used | `15` |
| 14 | (ignored) | Not read | |
| 15 | Modality | Name of an execution type configured in the application | `Lecture` |
| 16 | Day of week | English day name, see the accepted list in the Notes. May be empty | `Monday` |
| 17 | Start time | `HH:MM`, snapped to the nearest time slot. May be empty | `08:00` |
| 18 | (ignored) | Not read | |
| 19 | Duration in minutes | Converted to hours and then to 30-minute units | `90` |
| 20 | Room name | Optional. Created with 30 seats if unknown. Decides whether a locked schedule is written | `R-101` |
| 21 | Number of students | `0` or non-numeric becomes 1 | `24` |
| 22 | Required room properties | Optional. Comma-separated list, added to the course part's prerequisites | `Computer lab,Projector` |
| 23 | Week list | Optional. Comma-separated relative week numbers/ranges (`1` = first week of the term). If empty, a default two-block list around the mid-term week is generated | `1-7,10-15` |

**Sample**

```
1;3;;CS201;John Smith;1042;Computer Science;Software Engineering ENG;Fall;Algorithms and
Data Structures;A;30;15;;Lecture;Monday;08:00;;90;R-101;24;Computer lab,Projector;1-7,10-15
2;3;;CS201;Jane Doe;1117;Computer Science;Software Engineering ENG;Fall;Algorithms and Data
Structures;A;30;15;;Tutorial;Wednesday;14:00;;90;;18;Computer lab;
```

**Notes.**

- **Variants.** `MODGRPM` = intake flag 0, `MODGRPMI` = intake flag 1. The flag has two effects. (1) Generated group names: `G-<year>-<branch>` for `MODGRPM`, `GI-<year>-<branch>` for `MODGRPMI`. (2) Week settings for the English stream: with `ENG` in the branch name, `MODGRPM` uses the "M" set of first/last/mid-term week settings and `MODGRPMI` the "MI" set. If the branch name does **not** contain `ENG`, the "MG" set is used for both codes and the intake flag no longer matters for weeks.
- **Workspace selection is irrelevant.** Everything comes from columns 2, 7 and 8.
- **Auto-created vs. required.** Rooms (30 seats) and groups are created on demand; the lecturer is created from the name if unknown; the course is created from column 10 when no course carries the effective code. The program is looked up by name and used without a null check, so an unresolvable program name is not reported as a skipped line.
- **Strictness.** Lines are skipped without any warning when column 1 is not a non-zero number, when the line starts with `;`, or when no execution types are configured at all. Only a lecturer that cannot be provided produces a real warning.

- **Day vocabulary.** Matching is case-sensitive and by substring, so `Monday 08:00` is accepted. Accepted: `Monday`, `Mondey`, `Tueday`, `Tuesday`, `Wednesday`, `Wednsday`, `Thurday`, `Thursday`, `Friday`, `Saturday`, `Sunday`. Anything else, including an empty value, yields "no day".
- **Modality vocabulary.** The value must equal the name of an execution type exactly; **an unknown modality silently falls back to the first execution type in the list** rather than failing. Two values are special-cased: a modality whose name contains `Lecture` disables the group-splitting logic, and a modality named exactly `Seminar` whose week list starts with `2` gets `1,` prepended and the turn is marked "delete first week".
- **Time snapping.** The start time is matched to the time slot with the smallest absolute difference, with no tolerance limit — a time outside the timetable grid still snaps to the nearest slot. Duration is minutes divided by 60, then doubled and truncated to whole 30-minute units, so 100 minutes becomes 90.
- **Room / start time combinations.** Room *and* start time — a locked schedule entry is created for each week block, with all week flags set to `N`. Room empty *and* a recognised day — only the turn's preferred start hour and day flag are set. Room present but the day was not recognised — the schedule is written with an invalid day; check column 16 spelling.
- **Group splitting.** For non-lecture modalities, when the course part already holds more than one turn: with exactly two turns the first turn's group is re-pointed to `<group>-1` and the current line goes to `<group>-2`; with more turns the current line goes to `<group>-<n+1>`, where n counts already-allocated groups whose name starts with the base name. Sub-groups are created as class-1 children of the base group.
- **Known traps.** Several field buffers (lecturer, program, branch, course name) are static and are not cleared per line, so a line that simply ends early inherits the previous line's values for the missing columns — always write all 23 separators. When the first turn's group name does not match the expected base name, the splitting branch clears that turn's group allocation instead of splitting. Week numbers in column 23 are relative to the term's first week; a range that exceeds the term extends the turn's begin/end week.
- **Ordering.** Run before `MODGRPA`/`AVAIL`, which match lecturers only by the code written in column 6.

## PARTTIME — Dated part-time schedules

**What it does.** Reads a file in which every line is one concrete dated teaching event (date, group, course, execution type, tutor, time range, room) and turns each line into its own course part, turn, turn part and a placed schedule entry. Courses, execution types and tutors are created on the fly when they cannot be matched; groups and rooms must already exist and a line referring to an unknown group or room is skipped. Exact clock times from the file are stored on the schedule entry, while the grid slot is only the nearest time label. All problems are collected into one cumulative report shown in a single popup at the end of the run.

**How to start it.** `Data ▸ Custom import`, then type `PARTTIME` at the prompt and pick the file.

**File format.** Semicolon-separated (`;`) plain text, one event per line; the file is opened through the UTF-8 transcoding reader, so a legacy ANSI export is converted on the way in. `.xls`/`.xlsx` may be selected and are converted to CSV first. No header line is expected — lines shorter than 10 characters and lines starting with `#` or `;` are ignored, so a comment header must start with `#`. Quoting is not used. Exactly 7 fields are required (fewer means the line is skipped with a warning; extra fields are ignored). Each field is trimmed of surrounding whitespace.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Date | `dd.mm.yyyy`. A 2-digit year is expanded by +2000. The week is derived from day and month only, so the date must fall | `02.10.2026` |

| # | Column | Description | Example |
|---|---|---|---|
| | | inside the current school year. | |
| 2 | Group | Group name; must already exist (never created). | `VSM izredni 2` |
| 3 | Course | Course name, matched case-insensitively; created as a modular course when missing. | `Fundamentals of Business Finance` |
| 4 | Execution type | Execution-type name, matched case-insensitively; created when missing. Also written to the schedule entry title. | `P` |
| 5 | Tutor | `Name Surname, academic titles`. The first word is the first name, everything after it (including `, prof. dr.`) is the surname. | `Igor Stubelj, prof. dr.` |
| 6 | Time | `HH:MM-HH:MM`. Snapped to the nearest grid labels; the exact minutes are kept on the schedule entry. | `16:30-20:00` |
| 7 | Room | Room name; must already exist. Tried as written, then again with a trailing `(capacity)` suffix removed. | `Lecture room 3 (54)` |

**Sample**

```
# date;group;course;type;tutor;time;room
02.10.2026;VSM izredni 2;Fundamentals of Business Finance;P;Igor Stubelj, prof. dr.;16:30-20:00;Lecture room 3 (54)
03.10.2026;VSM izredni 2;Fundamentals of Business Finance;V;Barbara Svagan, asist. dr.;09:00-12:30;Lecture room 3 (54)
```

**Notes.**

- **Strictness:** group and room are strict (line skipped with a warning); course, execution type and tutor are permissive (created and reported). A line whose course already exists but is *not* modular is skipped, because dated events cannot be placed into a weekly periodic course.
- **Tutor matching ignores academic titles and word order.** The name from the file is compared against every existing tutor's name and surname after both have been stripped: commas and semicolons become spaces, every token ending in a dot (`prof.`, `dr.`, `viš.`, `izr.`, `doc.`, `mag.` …) is dropped as an abbreviation, and a stoplist removes the spelled-out forms (`lektor`, `lektorica`, `predavatelj`, `predavateljica`, `asistent`, `asistentka`, `docent`, `docentka`, `profesor`, `profesorica`, `prof`, `dr`, `mag`, `spec`). What remains are the plain name words, compared case-insensitively as an unordered set of equal size — so `Igor Stubelj` matches a stored `Stubelj, prof. dr. Igor`. When the file carries titles and the stored record does not, the stored name and surname are overwritten with the titled variant from the file (reported as a warning), so the titled form always wins. Only when no existing person matches on name words is a new tutor created. These helpers are deliberately public and should be reused by any future import that matches people.
- **Post-pass:** all collected warnings are shown in one popup instead of one popup per bad line. The buffer holds about 8000 characters; anything beyond that is only counted and the popup ends with a line pointing at `wtt_event.log`. Every warning is also mirrored into the event log.
- Every line always produces a brand-new turn, turn part and schedule entry — nothing is merged or de-duplicated, and nothing is deleted first. Running the same file twice duplicates all events.
- The course branch list is extended with the branch of the matched group, the turn's preferred room is set to the matched room, the turn duration list gets one entry of half the covered slots, and the turn part's allocated hours are set to the number of slots.
- Besides the grid slot, the schedule entry keeps the exact from/to hours and minutes, the raw date string, the execution type as title, and a display marker of the form `{(16:30 - 20:00) P}` used by the web and the exports.

- Does not use the workspace selection.

## Bookings and exams

Room bookings, external commitments and examination terms. These produce reservations rather than lessons, with the exception of the exam course parts. ## ENRLE — Exam course parts

**What it does.** Adds the examination part to existing courses. For each line it locates the course by code, creates (or reuses) the exam course part of the configured exam modality, creates a turn spanning the exam weeks, sets the number of hours to twice the number of exam sessions, requires the configured exam room property, sets the preferred room, inherits the first teacher found on the course, and allocates the exam turn part to the groups of the listed programs, summing their student numbers.

**How to start it.** `Data ▸ Custom import`, then type `ENRLE` at the prompt and pick the file.

**File format.** Semicolon (`;`) separated, fixed separator, no quoting. UTF-8 transcoded on read; `.xls`/`.xlsx` is converted first. Line 1 is a header line and is skipped. One line = the exam of one course.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Course code | Code of an existing course. If empty, the line is ignored; if unknown, the line is logged and skipped. The number after the first space also gives the study year (number/100, minus 5 when above 5). | `CS 201` |
| 2 | Number of sessions | Exam sessions; `0` or empty is treated as 1. The turn duration is set to sessions × 2 slots. | `2` |
| 3 | Preferred room | Room name; the room must already exist. May be empty — no preferred room is then set. | `A101` |
| 4 | Programs | Comma-separated list of program/branch names whose groups sit the exam. Each name must contain a space. | `Computer Science,Applied Mathematics` |

**Sample**

```
Course code;Sessions;Preferred room;Programs
CS 201;2;A101;Computer Science
MA 101;1;;Computer Science,Applied Mathematics
```

**Notes.**

- Does not depend on the workspace selection.
- Requires an **exam modality** and an **exam room property** whose names contain the string configured in the exam room-property advanced setting. Without the modality the line is reported as an error; without the room property the line is skipped with a log entry.
- The exam turn uses the “first/last import week for exams” advanced settings. If a turn with exactly those weeks already exists on the exam course part it is reused and all its turn parts are deleted first, so this import is safely repeatable — unlike `ENRLC`.
- Nothing is auto-created except the exam course part and its turn: the course, the rooms, the modality, the room property, and the programs/branches/groups must already exist.
- The student count of the exam turn part is the sum of the sizes of all allocated groups, capped at 350.
- A program name in column 4 without a space aborts the whole line and it is counted as a warning; a program whose group cannot be found is only logged and the remaining programs are still processed.
- The import also assigns a random display colour to every imported course.

- **Ordering:** run after `ENRLP` and `ENRLC` (the teacher is inherited from the turns created by `ENRLC`).

## DBBOOKINGS / DBBOOKINGSDEL — Room bookings written straight into the database

**What it does.** Imports an external room-booking export and writes one reservation per booked day directly into the database. The booking title becomes the reservation note, all remaining columns are collected into the internal comment, and the room is matched by name. `DBBOOKINGS` adds to the existing reservations, `DBBOOKINGSDEL` first deletes all reservations in the database and then imports the file.

**How to start it.** `Data ▸ Custom import`, then type `DBBOOKINGS` (add to existing bookings) or `DBBOOKINGSDEL` (clear all reservations first) at the prompt and pick the file. Add `-d` after the code for line-by-line debug logging.

**File format.** Semicolon (`;`) separated, 11 columns, one booking per line; anything after column 11 is ignored. Lines shorter than 10 characters are skipped. There is no header handling: a header row survives only because a line whose 4th cell contains no `:` is dropped, so keep the file header-free to be safe. Dates are written in English (`Mon 05 October 2026`), so the day names and month names must be the English ones and the **day number must have two digits**.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Short title | Becomes the reservation note (the text shown in the timetable). | `Chem Seminar` |
| 2 | Area name | Added to the internal comment as `Area name: …`. May be empty. | `Physics Area` |
| 3 | Room name | Matched exactly, then by partial name; if still not found the room is created. | `Lecture Theatre 1` |
| 4 | Start | `HH:MM - <Weekday> DD Month YYYY`. Minutes 1–30 are rounded to `:30`, 31–59 to the next full hour. | `09:00 - Mon 05 October 2026` |
| 5 | End | Same format. May be a later date for a multi-day booking. | `11:00 - Mon 05 October 2026` |
| 6 | Duration | Free text; only copied into the internal comment. | `2 hours` |
| 7 | Long title / description | Added to the internal comment as `Description: …`. May be empty. | `Weekly research seminar` |
| 8 | Type | Added to the internal comment as `Type: …`. May be empty. | `Internal` |
| 9 | Created by | Added to the internal comment as `Created by: …`. May be empty. | `j.smith` |
| 10 | Confirmed | Added to the internal comment as `Confirmed: …`. May be empty. | `Confirmed` |
| 11 | Last updated | Added to the internal comment as `Last updated: …`. May be empty. | `01 Oct 2026` |

### Sample

```
Chem Seminar;Physics Area;Lecture Theatre 1;09:00 - Mon 05 October 2026;11:00 - Mon 05 October 2026;2 hours;Weekly research seminar;Internal;j.smith;Confirmed;01 Oct 2026
Open Day;Physics Area;Main Hall;10:00 - Fri 09 October 2026;16:30 - Sat 10 October 2026;2 days;Departmental open day;Public;a.jones;Confirmed;02 Oct 2026
```

**Notes.**

- **What the pre-pass deletes.** Both codes run a preprocessing step that connects to the export database. With `DBBOOKINGSDEL` it executes `DELETE FROM tbreservation`, `DELETE FROM tbreservation_groups`, `DELETE FROM tbreservation_room` and `DELETE FROM tbreservation_tutor`, each with `WHERE 1=1` — this removes **every** reservation in the database, past and future, including reservations that were entered manually or created by other imports, together with all their group, room and lecturer links; the reservation id counter is reset to 0. With `DBBOOKINGS` nothing is deleted; the highest existing reservation id is read so that new ids continue from there. **Use `DBBOOKINGSDEL` only for a full reload of the booking database.**
- The pre-pass also loads the academic week calendar into memory; the post-pass only disconnects from the database.
- Week and weekday are resolved by walking the weeks in the order returned by the database and taking the first week whose last day is on or after the booking date. The weekday is then derived by counting days forward, giving 1 for the first day of the week and 7 for the last. **Consequences:** a date *after* the last defined week is not found and the booking is skipped; a date *before* the first defined week is placed in the first week with an out-of-range day number. Keep the file within the defined academic weeks.
- **Slot numbering for this import:** slot 1 = 07:00, each slot is 30 minutes; the end is capped at slot 25 (19:00) and the start at slot 1.
- Multi-day bookings are split into one reservation per day: the first day runs from the start slot to slot 25, full days in between from slot 1 to slot 25, and the last day from slot 1 to the end slot.
- Reservations are inserted with SQL directly into the reservation and reservation-room tables (type 4, status 2, no owner, no course, flagged as an imported booking, a fresh GUID per row). They do **not** pass through the in-memory model, so they appear after the data is reloaded rather than as unsaved changes.
- Rooms are created automatically when neither an exact nor a partial name match is found; no groups or lecturers are attached.
- `-d` (typed in lower case, e.g. `DBBOOKINGS -d`) logs every line before it is parsed — useful when a booking silently disappears.
- Independent of the workspace selection.

## GETEXAMS — Exam sittings as reservations

**What it does.** Imports a published exam timetable as reservations. Each line describes one exam sitting: date, start and end time, exam type, course, room, an online flag, up to two invigilators/examiners and a free note. The import converts the date into the school-year week and weekday, snaps the start and end times to the nearest timetable slots, then creates a reservation (or extends an existing one) with the matched course, teachers and room; whatever cannot be matched is written into the reservation comment instead.

**How to start it.** `Data ▸ Custom import`, then type `GETEXAMS` at the prompt and pick the file.

**File format.** Semicolon (`;`) separated, fixed separator, no quoting. UTF-8 transcoded on read; `.xls`/`.xlsx` is converted first. Line 1 is a header line and is skipped. One line = one exam sitting.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Date | `d.m.yyyy`, dot-separated. Day 1–31 and month 1–12 are validated; an invalid or missing date makes the line be skipped with a log entry. | `3.2.2026` |

| # | Column | Description | Example |
|---|---|---|---|
| 2 | Start time | `H:MM` (minutes after a colon). Must not be empty. Snapped to the nearest timetable slot. | `09:00` |
| 3 | End time | `H.MM` — minutes are read **after a dot**, not a colon. May be empty; the default exam duration from the advanced settings is then used. | `11.30` |
| 4 | Exam type | Free text; copied into the reservation comment in brackets. May be empty. | `Final exam` |
| 5 | Course name | Course title; matched against existing course names (see Notes). May be empty. | `Data Structures` |
| 6 | Room | Room name; matched against existing rooms. If not matched, the text is added to the comment. | `A101` |
| 7 | Online | `true` (case-insensitive) prefixes the comment with `ONLINE:`. Any other value, including empty, means on-site. | `false` |
| 8 | First examiner | `Surname FirstName` — everything before the **last** space is the surname, the last word is the first name. May be empty. | `Cohen David` |
| 9 | Second examiner | Same format as column 8. May be empty. | `Levi Sarah` |
| 10 | Note | Free text; appended to the comment after `>>`. May be empty. | `Bring a calculator` |

**Sample**

```
Date;Start;End;Type;Course;Room;Online;Examiner 1;Examiner 2;Note
3.2.2026;09:00;11.30;Final exam;Data Structures;A101;false;Cohen David;Levi Sarah;Bring a calculator
5.2.2026;14:00;16.00;Resit;Operating Systems;B204;true;Katz Amir;;
```

**Notes.**

- Independent of the workspace selection.
- Teachers **are** auto-created when the name is unknown (a log entry records each creation). Courses and rooms are never created — an unmatched course simply leaves the reservation without a course link, an unmatched room name ends up in the comment text.
- **Course matching is done in three steps:** exact name match (case-insensitive), then a substring match in either direction, then a match against the part of the course name that precedes `-`. **Room matching** is: exact name, then the name prefixed with `predavalnica_`, then a substring match. These fallbacks are loose — verify the result after import.
- An examiner name without a space is never resolved to a teacher; the raw text goes into the comment instead.
- Reservations are created with reservation type 3 and no owner. The date must fall inside the configured school year.
- **Merging trap:** before creating a reservation, the import looks for an existing reservation whose first teacher is the first examiner, whose weekday equals the line's weekday and whose start slot is the same — **the week is not compared**. An exam on the same weekday and hour with the same examiner in a different week is therefore merged into the earlier reservation instead of being created separately, and the two comments are concatenated.
- The reservation comment is rebuilt into a fixed 1024-character buffer on every import, so long comments are truncated.
- If the end time equals the start time (or snaps to the same slot), the resulting duration is 0.

- Link quality improves when courses, rooms and teachers already exist — run it after the course and room imports.

## PROGCATEXT — External timetable grid converted into group reservations

**What it does.** Reads an external weekly timetable laid out as a grid (a line per time slot, a column per activity, preceded by lines that name the weekday) and creates a one-hour "External timetable" reservation for every group it can recognise in a cell. The reservations block the groups so that the external commitments are respected when scheduling.

**How to start it.** `Data ▸ Custom import`, then type `PROGCATEXT` at the prompt and pick the file.

**File format.** Semicolon (`;`) separated. Two kinds of lines: weekday lines and time-slot lines. Lines shorter than 10 characters and lines starting with `#` are skipped — **a weekday line must therefore be padded to at least 10 characters** (for example `Monday;;;;`). No header line is used.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Weekday **or** placeholder | On a weekday line: exactly `Monday`, `Tuesday`, `Wednesday`, `Thursday` or `Friday` (case-sensitive); the rest of that line is ignored and the day stays in force for all following lines. On a time-slot line: any other value, normally empty. | `Monday` |
| 2 | Time | `HHMM` without a separator; the first two characters are the hour, the rest the minutes. `15` is rounded down to `00`, `45` is rounded up to the next full hour. Must match a defined time slot, otherwise the line is skipped. A cell that does not start with a digit skips the line. | `0900` |
| 3 … n | Activity cells | One group token per cell; the cell is cut at the first space, so `CS1 Group A` is read as `CS1`. Cells with fewer than 2 characters, tokens longer than 6 characters, and tokens containing `Logic`, `Data`, `Group` or `High` are ignored. All cells to the end of the line are examined. | `CS1` |

### Sample

```
Monday;;;;;;;;
;0900;CS1;M2;;;;;
;1015;CS3;;;;;;
```

### Notes.

- The group token must be a prefix of an existing group name and the character right after the prefix in that group name must not be alphanumeric (so `CS1` matches `CS1-A` but not `CS10`). The first matching group wins. Groups are never created.
- Every match creates a reservation of duration 2 slots (one hour) on the current weekday at the matched slot, with the comment `External timetable`, no owner, and reservation type "other".
- **The week span is hard-coded in the source** as weeks 15 to 22. Importing for a different term requires changing those two values in the code and rebuilding — the file cannot override them.
- The current weekday and time slot are static and are reset only when the import starts at line 1, so the file must begin with a weekday line before any time-slot line.
- Nothing is deleted beforehand: re-running the file duplicates the reservations.
- Does not depend on the workspace selection.

## ROOMBOOKINGS — Room bookings (reservations)

**What it does.** Creates one reservation (room booking) per line, with its week range, day range, starting slot, duration, type and comment. The reservation is created without any room; the external booking identifier from the file is stored on the reservation so that the companion import `ROOMBOOKROOMS` can attach rooms to it afterwards.

**How to start it.** `Data ▸ Custom import`, then type `ROOMBOOKINGS` at the prompt and pick the file.

**File format.** Semicolon-separated (`;`), no quoting support. Read as UTF-8; `.xls`/`.xlsx` is converted automatically. **The first line is always skipped**, so a header line is expected (if the data starts on line 1, that first booking is lost). Lines shorter than 2 characters are skipped. One line = one booking.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | (ignored) | Not read. May hold any value, may be empty. | `1001` |
| 2 | Booking ID | External numeric identifier of the booking. Stored on the reservation and used by `ROOMBOOKROOMS` to match room lines to this booking. | `5001` |
| 3 | Reservation type | Numeric reservation type code. | `2` |
| 4 | Note | Free text, stored as the reservation comment. May be empty. | `Faculty meeting` |
| 5 | (ignored) | Not read. May be empty. | |
| 6 | From week | Absolute week number; **52 is subtracted** to get the internal week. | `60` |
| 7 | To week | Absolute week number; 52 is subtracted. | `60` |
| 8 | From day | Day of week, 1-based in the file; 1 is subtracted. | `1` |
| 9 | To day | Day of week, 1-based; 1 is subtracted. | `1` |
| 10 | Begin slot | Starting time slot, 1-based in the file; 1 is subtracted. | `3` |
| 11 | Duration | Length in time slots. | `4` |

### Sample

```
BookingRef;BookingId;Type;Note;Status;FromWeek;ToWeek;FromDay;ToDay;Begin;Duration
1001;5001;2;Faculty meeting;;60;60;1;1;3;4
1002;5002;1;External seminar;;61;63;3;3;5;2
```

### Notes.

- **Ordering.** Run `ROOMBOOKINGS` first; then run `ROOMBOOKROOMS` on the room file.
- Every line creates a **new** reservation — there is no matching against existing bookings, so re-running duplicates them.
- Reservations are created with no owner and with no room; without a subsequent `ROOMBOOKROOMS` run they stay room-less.
- The week offset is fixed at 52 and is not configurable — the source file must number weeks continuously past the year boundary.
- Columns 3 and 6–11 are read with a plain numeric conversion: a non-numeric or empty value becomes 0 (and after the offset, −52 for weeks, −1 for day/begin).

## ROOMBOOKROOMS — Rooms for imported bookings

**What it does.** Attaches rooms to the reservations previously created by `ROOMBOOKINGS`. Each line pairs a booking identifier with a source room index; the reader finds the reservation carrying that identifier and adds the mapped internal room to its room list. Several lines with the same booking identifier add several rooms to the same booking.

**How to start it.** `Data ▸ Custom import`, then type `ROOMBOOKROOMS` at the prompt and pick the file.

**File format.** Semicolon-separated (`;`), no quoting support. Read as UTF-8; `.xls`/`.xlsx` is converted automatically. **No header line is skipped** — every line is treated as data. Lines shorter than 2 characters are skipped. One line = one room assigned to one booking.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Booking ID | Must equal the booking identifier from column 2 of the `ROOMBOOKINGS` file. Lines whose booking is not found are skipped silently. | `5001` |
| 2 | Source room index | Numeric room index from the source system, 1–12. Mapped to internal room IDs: 1–6 map to rooms 1–6; 7 to 10, 8 to 11, 9 to 12, 10 to 13; 11 and 12 both map to room 1. Any other value causes the line to be skipped. | `7` |

**Sample**

```
5001;1
5001;7
5002;3
```

**Notes.**

- **Ordering.** Must be run *after* `ROOMBOOKINGS` in the same session. Nothing is created here.
- **The room mapping is hard-coded**, not read from the file and not name-based. The workspace must contain rooms with internal IDs 1–6 and 10–13 for the mapping to land on the intended rooms; if the room table differs, bookings end up in the wrong rooms.
- Source room indexes 11 and 12 are both folded onto room 1 — verify those bookings manually.
- Unmatched booking IDs and out-of-range room indexes are dropped without any warning and are **not** counted as import warnings.

# Part 3 — Custom operations

These are not file imports. They are reached through `Data ▸ Custom operation`, which asks for a code in the same kind of prompt as the custom imports (the code is upper-cased, and confirming an empty prompt repeats the previous one), and then either converts something in the open schedule or writes an export file.

Two of them — `CONVERT15` and `CONVERT60` — change every placed lesson and every reservation in one pass and cannot be undone except by closing the schedule without saving. Read their notes before running them.

## CHECKCRSCODE — List courses that have no course code

**What it does.** Walks the whole course map of the open schedule and checks the code field of every course. Every course whose code is missing or empty is written to the application event log with its name. Nothing is changed, nothing is created, no file is produced — it is a read-only data-quality check, meant to be run right after an import to find courses that the import failed to give a code.

**How to start it.** `Data ▸ Custom operation`, then type `CHECKCRSCODE` and confirm. A success message box appears when it has finished.

**Output / effect.** No export file. One line per offending course is appended to `wtt_event.log`, in the form `Course without code (even after import): <course name>`. The schedule is not modified.

**Notes.**

- The result is only visible in the log file — the message box says the operation succeeded whether or not anything was found, so the log has to be opened to read the outcome.
- The log is appended to, so results of several runs accumulate.

## COMMENTSUFFIX — Append the configured suffix to schedule comments

**What it does.** Walks all placed lessons of the open schedule and, for every lesson that already carries a comment, appends the configured schedule-comment suffix inside the comment's closing brace. Lessons with no comment, with an empty comment, or with no closing `}` are skipped, and so is any lesson whose comment already contains the suffix — so the operation is safe to run more than once.

**How to start it.** `Data ▸ Custom operation`, then type `COMMENTSUFFIX` and confirm.

**Output / effect.** No file is written. The comment text of the matching lessons is rewritten in place: the text up to the closing brace is kept, then a space, the configured suffix and the closing brace are added. The schedule is marked as needing to be saved. There is no undo other than closing the schedule without saving.

| Field | Description | Example |
|---|---|---|
| lesson comment | The configured suffix is inserted before the closing brace | `{Lab session}` becomes `{Lab session <suffix>}` |

**Notes.**

- Depends on the advanced setting that holds the suffix — if that setting is empty the operation only adds a trailing space.
- Idempotent by design, but the new text is written back into the existing comment buffer, so treat it as a single-pass operation and check a few lessons afterwards.

- Independent of the workspace selection.

## CONVERT15 — Convert the schedule to a finer time grid

**What it does.** Rescales every placed lesson and every reservation from the current time grid to a grid with twice as many slots, i.e. from 30-minute to 15-minute slots. For each placed lesson and each reservation it multiplies the starting slot and the length in slots by 2. Nothing else is touched — course, turn and duration definitions are left exactly as they are (that part of the routine is present in the source but commented out).

**How to start it.** `Data ▸ Custom operation`, then type `CONVERT15` and confirm.

**Output / effect.** No file is written. The open schedule is changed in memory and marked as needing to be saved, so the change becomes permanent only when the schedule is saved. There is no undo.

| Field | Description | Example |
|---|---|---|
| lesson start slot | Starting slot of every placed lesson, doubled | 16 becomes 32 |
| lesson duration | Length in slots of every placed lesson, doubled | 4 becomes 8 |
| reservation start slot | Starting slot of every reservation, doubled | 20 becomes 40 |
| reservation duration | Length in slots of every reservation, doubled | 3 becomes 6 |

**Notes.**

- It does **not** change the time-slot definition itself, and it does **not** rewrite the duration strings of course turns — the slot table must be switched to 15-minute slots separately, otherwise every lesson ends up at the wrong wall-clock time.
- **Run it exactly once.** A second run doubles everything again.
- Independent of the workspace selection; it always processes all lessons and all reservations.

## CONVERT60 — Convert the schedule to a coarser time grid

**What it does.** The counterpart of `CONVERT15`: it halves the starting slot and the length in slots of every placed lesson and every reservation, i.e. moves the schedule from 30-minute to 60-minute slots (or back from 15 to 30). The arithmetic is integer division by 2. As with `CONVERT15`, the course/turn duration strings are not converted.

**How to start it.** `Data ▸ Custom operation`, then type `CONVERT60` and confirm.

**Output / effect.** No file is written. The open schedule is changed in memory and marked as modified, so the change is written on the next save. There is no undo.

| Field | Description | Example |
|---|---|---|
| lesson start slot | Starting slot of every placed lesson, integer-divided by 2 | 33 becomes 16 |
| lesson duration | Length in slots of every placed lesson, integer-divided by 2 | 6 becomes 3 |
| reservation start slot | Starting slot of every reservation, integer-divided by 2 | 41 becomes 20 |
| reservation duration | Length in slots of every reservation, integer-divided by 2 | 5 becomes 2 |

**Notes.**

- **Lossy.** Integer division discards the remainder, so an odd starting slot moves half a slot earlier and an odd length is rounded down — a lesson one slot long becomes zero slots long. It is therefore **not** a safe inverse of `CONVERT15` on data that was not produced by `CONVERT15`.
- Like `CONVERT15`, it leaves the time-slot table and the course duration definitions untouched.

## EXPORTALL — Export all lessons and all bookings

**What it does.** Produces one file covering the whole schedule from the course side rather than the lecturer side. The first part walks every course and writes one line per placed lesson of that course, with the course code and name, the course note, the real start date, the weekday, the number of weeks, the start time, the duration in hours and minutes, the room, all lecturers, all groups, the subject area and the programme. The second part appends every reservation/booking, ordered by week and then by weekday.

**How to start it.** `Data ▸ Custom operation`, then type `EXPORTALL` and confirm; a save dialog filtered on `*.csv` asks for the destination.

**Output / effect.** The file chosen in the dialog. Data lines use the **configured CSV separator**; dates use the configured date format. A header line is written, but with hard-coded semicolons. An existing file is overwritten. Multi-valued fields (lecturers, groups, rooms) are joined with `/`. Comments are cleaned before writing: non-printable characters and newlines become spaces and any occurrence of the CSV separator becomes `-`.

Lesson lines:

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Code | Course code; empty when not set | `CS201` |
| 2 | Course Name | Course name | `Introduction to Databases` |
| 3 | Description | Course note, cleaned of separators and control characters | `Room booked for the whole semester` |
| 4 | Start Date | Date of the first occurrence, in the configured date format | `29.09.2025` |
| 5 | Day name | Full weekday name in the interface language | `Monday` |
| 6 | Number of weeks | Last week − first week + 1 | `12` |
| 7 | Start time | Start time of the first slot | `08:00` |
| 8 | Duration | Real teaching duration, `<n>h` or `<n>h<m>m` | `1h30m` |
| 9 | Classroom | Room name | `A-101` |
| 10 | Instructor | All lecturers of the turn, `Name Surname`, joined with `/` | `Anna Novak/Peter Brown` |
| 11 | Section/Group | All allocated groups, joined with `/` | `CS1-A/CS1-B` |
| 12 | Subject Area | Branch of the first allocated group | `Computer Science` |
| 13 | Program name | Programme of that branch | `BSc Computing` |

Booking lines reuse the same 13 positions with a different meaning:

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Code | `Booking`, or `Booking (N days)` for a multi-day booking | `Booking (3 days)` |
| 2 | Course Name | Number of days the booking spans | `3` |
| 3 | Description | Booking comment, cleaned | `Faculty council meeting` |
| 4 | Start Date | Date of the booking's first day | `06.10.2025` |
| 5 | Day name | Weekday of the booking's first day | `Monday` |

| # | Column | Description | Example |
|---|---|---|---|
| 6 | Number of weeks | Last week − first week + 1 | `1` |
| 7 | Start time | Start time of the booking | `10:00` |
| 8 | Duration | Real duration, `<n>h` or `<n>h<m>m` | `2h` |
| 9 | Classroom | All booked rooms, joined with `/` | `A-101/A-102` |
| 10 | Instructor | All lecturers on the booking, joined with `/` | `Anna Novak` |
| 11 | Section/Group | All groups on the booking, joined with `/` | `CS1-A` |
| 12 | Subject Area | Name of the course the booking is attached to, if any | `Introduction to Databases` |
| 13 | Program name | Always empty on booking lines | |

**Sample**

```
Code;Course Name;Description;Start Date;Day name;Number of weeks;Start
time;Duration;Classroom;Instructor;Section/Group;Subject Area;Program name
CS201;Introduction to Databases;;29.09.2025;Monday;12;08:00;1h30m;A-101;Anna Novak/Peter
Brown;CS1-A/CS1-B;Computer Science;BSc Computing
Booking (3 days);3;Faculty council meeting;06.10.2025;Monday;1;10:00;2h;A-101/A-102;Anna
Novak;;;
```

**Notes.**

- Does not modify the schedule. The file is overwritten.
- Two different record layouts share one file, so a receiving system must branch on column 1 starting with `Booking`.
- **Trap:** the header line always uses semicolons while the data lines use the configured separator — if the installation is configured with a different separator, the header does not match the data.
- **Changed in this revision.** Rooms, lecturers and groups were looked up by id without null checks in both the lessons and the bookings part, so a record referencing a deleted entity crashed the export. Missing entities are now skipped and their field is left empty.
- Bookings are written for weeks 1–52 only.

## EXPORTAVAIL — Export lecturer availability

**What it does.** Exports the desired availability of the lecturers: for each lecturer it collects all reservations of type “wanted” that include that lecturer, and lists their time ranges per weekday. Each day cell holds the ranges for that day, comma-separated, in `HH:MM-HH:MM` form. Lecturers without a code are skipped, and so are lecturers for whom no wanted reservation was found — only lecturers with at least one availability entry appear in the file.

**How to start it.** `Data ▸ Custom operation`, then type `EXPORTAVAIL` and confirm; a save dialog filtered on `*.csv` asks for the destination.

**Output / effect.** The file chosen in the dialog. Separator is a fixed semicolon. A header line is written. An existing file is overwritten. The lecturer name is written as a single field, first name and surname separated by a space. Seven day columns are written for every lecturer, Monday through Sunday.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Lecturer ID | Lecturer code; lecturers without a code are not exported | `1042` |
| 2 | Lecturer | First name and surname in one field | `Anna Novak` |

| # | Column Name | Description | Example |
|---|---|---|---|
| 3 | Monday | Comma-separated `HH:MM-HH:MM` ranges wanted on Monday | `08:00-12:00,14:00-16:00` |
| 4 | Tuesday | Ranges wanted on Tuesday | `08:00-12:00` |
| 5 | Wednesday | Ranges wanted on Wednesday | |
| 6 | Thursday | Ranges wanted on Thursday | `10:00-14:00` |
| 7 | Friday | Ranges wanted on Friday | `08:00-10:00` |
| 8 | Saturday | Ranges wanted on Saturday | |
| 9 | (Sunday) | Ranges wanted on Sunday — written, but **not named in the header** | |

**Sample**

```
Lecturer ID;Lecturer Name;Monday;Tuesday;Wednesday;Thursday;Friday;Saturday
1042;Anna Novak;08:00-12:00,14:00-16:00;08:00-12:00;;10:00-14:00;08:00-10:00;;
1087;Peter Brown;;09:00-13:00;09:00-13:00;;;;
```

**Notes.**

- Does not modify the schedule. The file is overwritten.
- **The layout matches the `MODGRPA` / `AVAIL` import**, which reads exactly these nine columns, so the file can be re-imported to move availability between schedules. Two traps around that round trip: the header names only six days while the data rows carry seven, so the header line is one field shorter than the data; and the import skips any line whose first field does not start with a digit, which means the header is correctly ignored but a lecturer whose code is not numeric is silently skipped on re-import.
- A reservation spanning several weekdays is repeated in every day column it covers, and no week filtering is applied.

## EXPORTLECT — Export the lecturer list

**What it does.** Writes the complete lecturer list of the open schedule to a CSV file chosen in a save dialog: code, first name, surname, e-mail and password, one line per lecturer. Missing values are written as empty fields. It is the same content as `EXPORTPASS`, but with a freely chosen destination and without the `not defined` placeholder for missing passwords.

**How to start it.** `Data ▸ Custom operation`, then type `EXPORTLECT` and confirm; a save dialog filtered on `*.csv` asks for the destination.

**Output / effect.** The file chosen in the dialog. Separator is a fixed semicolon, regardless of the configured CSV separator. **No header line** is written. An existing file is overwritten. All lecturers are exported, including those with no code.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Code | Lecturer code; empty when not set | `L0142` |
| 2 | Name | Lecturer first name | `Anna` |
| 3 | Surname | Lecturer surname | `Novak` |
| 4 | Email | Lecturer e-mail; empty when not set | `anna.novak@university.edu` |
| 5 | Password | Lecturer password; empty when not set | `Kt7pQz4m` |

**Sample**

```
L0142;Anna;Novak;anna.novak@university.edu;Kt7pQz4m
L0187;Peter;Brown;peter.brown@university.edu;
;Sarah;Miller;;
```

**Notes.**

- Does not modify the schedule. The file is overwritten, not appended.
- Contains plaintext passwords — handle the file accordingly.
- Since there is no header, a spreadsheet will treat the first lecturer as the header row unless the import is configured otherwise.
- **Changed in this revision.** Cancelling the file dialog used to leave the routine with an undefined result, so the success message could appear although nothing was written. Cancelling is now reported as “not done”.

## EXPORTLESSONS — Export lecturer lessons with school year and term

**What it does.** A variant of `EXPORTSCHED` extended with the school year, the term and the student limit, and with duplicate lines removed. One line is produced per lecturer and placed lesson. The term is derived from the first week of the lesson: weeks above 18 are `Spring`, everything else is `Fall`.

**How to start it.** `Data ▸ Custom operation`, then type `EXPORTLESSONS` and confirm; a save dialog filtered on `*.csv` asks for the destination.

**Output / effect.** The file chosen in the dialog. Separator is a fixed semicolon. A header line is written. An existing file is overwritten. The course code is cut at the first colon; a course without a code becomes `unknown`, an unresolvable execution type becomes `unknown`.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Code | Lecturer code; empty when not set | `L0142` |
| 2 | Name | Lecturer first name | `Anna` |
| 3 | Surname | Lecturer surname | `Novak` |
| 4 | Email | Lecturer e-mail; empty when not set | `anna.novak@university.edu` |
| 5 | Course Code | Course code, cut at the first `:`; `unknown` when missing | `CS201` |
| 6 | Course Name | Course name | `Introduction to Databases` |
| 7 | Modality | Execution type name; `unknown` when unresolvable | `Lecture` |
| 8 | School Year | The schedule’s school year | `2025` |
| 9 | Term | `Spring` when the first week is above 18, otherwise `Fall` | `Fall` |
| 10 | Day | Full weekday name in the interface language | `Monday` |
| 11 | Start time | Start time of the first slot | `08:00` |
| 12 | End time | End time of the last slot | `09:30` |
| 13 | Classroom | Room name | `A-101` |
| 14 | Student limit | Number of students on the turn part | `48` |

**Sample**

```
Code;Name;Surname;Email;Course Code;Course Name;Modality;School Year;Term;Day;Start time;End time;Classroom;Student limit
L0142;Anna;Novak;anna.novak@university.edu;CS201;Introduction to Databases;Lecture;2025;Fall;Monday;08:00;09:30;A-101;48
```

```
L0187;Peter;Brown;peter.brown@university.edu;CS201;Introduction to
Databases;Lab;2025;Fall;Wednesday;14:00;16:30;Lab-3;24
```

**Notes.**

- The file is overwritten. Duplicate suppression is exact-text and limited to 8096 distinct lines; **changed in this revision**, a schedule producing more unique lines than that used to write past the internal list — it now keeps exporting and simply stops de-duplicating.
- **Changed in this revision.** This export used to mark the schedule as needing saving even though it changed nothing, so a spurious "unsaved changes" prompt followed every run.
- The term rule is a fixed week-18 boundary and is not derived from the school's actual semester weeks.
- **Changed in this revision.** A lesson referencing a deleted room used to crash the export; the classroom field is now written empty instead.

## EXPORTLESSONSD — Export lecturer lessons with real from/to dates

**What it does.** Exactly the same export as `EXPORTLESSONS`, run with the "show dates" option on: two extra columns carry the real calendar date of the first and the last occurrence of each lesson, computed from the lesson's first and last week together with its weekday.

**How to start it.** `Data ▸ Custom operation`, then type `EXPORTLESSONSD` and confirm; a save dialog filtered on `*.csv` asks for the destination.

**Output / effect.** The file chosen in the dialog. Separator is a fixed semicolon. A header line is written. An existing file is overwritten. The two date columns always use the fixed pattern `dd.mm.yyyy`, independently of the configured date format.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Code | Lecturer code; empty when not set | `L0142` |
| 2 | Name | Lecturer first name | `Anna` |
| 3 | Surname | Lecturer surname | `Novak` |
| 4 | Email | Lecturer e-mail; empty when not set | `anna.novak@university.edu` |
| 5 | Course Code | Course code, cut at the first `:`; `unknown` when missing | `CS201` |
| 6 | Course Name | Course name | `Introduction to Databases` |
| 7 | Modality | Execution type name; `unknown` when unresolvable | `Lecture` |
| 8 | School Year | The schedule's school year | `2025` |
| 9 | Term | `Spring` when the first week is above 18, otherwise `Fall` | `Fall` |
| 10 | From date | Date of the first occurrence, `dd.mm.yyyy` | `29.09.2025` |
| 11 | To date | Date of the last occurrence, `dd.mm.yyyy` | `15.12.2025` |
| 12 | Day | Full weekday name in the interface language | `Monday` |
| 13 | Start time | Start time of the first slot | `08:00` |
| 14 | End time | End time of the last slot | `09:30` |
| 15 | Classroom | Room name | `A-101` |
| 16 | Student limit | Number of students on the turn part | `48` |

**Sample**

```
Code;Name;Surname;Email;Course Code;Course Name;Modality;School Year;Term;From date;To date;Day;Start time;End time;Classroom;Student limit
L0142;Anna;Novak;anna.novak@university.edu;CS201;Introduction to Databases;Lecture;2025;Fall;29.09.2025;15.12.2025;Monday;08:00;09:30;A-101;48
```

**Notes.**

- Same behaviour as `EXPORTLESSONS`, including the fixes made in this revision: the 8096-line duplicate list no longer overruns, the schedule is no longer flagged as modified, deleted rooms no longer crash the export, and cancelling the dialog no longer reports success.
- The dates are the first and the last occurrence of a weekly pattern, not a list of individual dates — the receiving system has to expand the range itself.
- Adding the two date columns shifts every following column by two, so a consumer written for `EXPORTLESSONS` cannot read this file unchanged.

## EXPORTPASS — Export lecturer passwords to a fixed file

**What it does.** Writes one line per lecturer — code, first name, surname, e-mail and password — to a file with a fixed name in the application data directory. No file dialog is shown; the destination is not asked for. Lecturers whose password is empty are still exported, with the literal text `not defined` in the password column. Typically run straight after `GENERATEPASS`.

**How to start it.** `Data ▸ Custom operation`, then type `EXPORTPASS` and confirm.

**Output / effect.** File `TutorPasswords.csv`, written into the application data directory the program uses for its own settings and logs. Separator is a fixed semicolon. **No header line** is written. An existing file is overwritten.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Code | Lecturer code; empty when not set | `L0142` |
| 2 | Name | Lecturer first name | `Anna` |
| 3 | Surname | Lecturer surname | `Novak` |
| 4 | Email | Lecturer e-mail; empty when not set | `anna.novak@university.edu` |
| 5 | Password | Lecturer password, or `not defined` when empty | `Kt7pQz4m` |

**Sample**

```
L0142;Anna;Novak;anna.novak@university.edu;Kt7pQz4m
L0187;Peter;Brown;peter.brown@university.edu;not defined
```

**Notes.**

- Does not modify the schedule. Overwrites the previous file without asking.
- The file contains plaintext passwords in a shared application folder — delete it once the passwords have been distributed.
- The destination cannot be chosen. **Changed in this revision:** if the file could not be opened for writing, the routine still tried to close the failed handle and could bring the application down; it is guarded now, but make sure the file is not open in another program.

## EXPORTSCHED — Export the placed lessons per lecturer

**What it does.** Writes one line for every combination of lecturer and placed lesson: the lecturer's identification, the course code and name, the execution type, the week range, the weekday, the start and end time and the classroom. Only lessons whose turn actually lists that lecturer are exported, so a lesson taught by two lecturers appears twice.

**How to start it.** `Data ▸ Custom operation`, then type `EXPORTSCHED` and confirm; a save dialog filtered on `*.csv` asks for the destination.

**Output / effect.** The file chosen in the dialog. Separator is a fixed semicolon. A header line is written. An existing file is overwritten. Times come from the schedule's time-slot table; days are the full weekday names of the current interface language.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Code | Lecturer code; empty when not set | `L0142` |
| 2 | Name | Lecturer first name | `Anna` |
| 3 | Surname | Lecturer surname | `Novak` |
| 4 | Email | Lecturer e-mail; empty when not set | `anna.novak@university.edu` |
| 5 | Course Code | Course code, cut at the first `:`; `unknown` when the course has none | `CS201` |
| 6 | Course Name | Course name | `Introduction to Databases` |
| 7 | Modality | Execution type name; `unknown` when it cannot be resolved | `Lecture` |
| 8 | From week | First week number of the lesson | `40` |
| 9 | To week | Last week number of the lesson | `51` |
| 10 | Day | Full weekday name in the interface language | `Monday` |
| 11 | Start time | Start time of the first slot | `08:00` |
| 12 | End time | End time of the last slot | `09:30` |
| 13 | Classroom | Room name of the lesson | `A-101` |

**Sample**

```
Code;Name;Surname;Email;Course Code;Course Name;Modality;From week;To week;Day;Start time;End time;Classroom
L0142;Anna;Novak;anna.novak@university.edu;CS201;Introduction to Databases;Lecture;40;51;Monday;08:00;09:30;A-101
L0187;Peter;Brown;peter.brown@university.edu;CS201;Introduction to Databases;Lab;40;51;Wednesday;14:00;16:30;Lab-3
```

**Notes.**

- Does not modify the schedule. The file is overwritten.
- Week numbers, not dates — use `EXPORTLESSONSD` when the receiving system needs real dates.
- Duplicate lines are possible: nothing is de-duplicated, so a lecturer listed twice on the same turn produces the same line twice.
- **Changed in this revision.** The room was looked up by id without a null check, so a lesson referencing a deleted room crashed the export. The classroom field is written empty instead.

## EXPORTTRANS — Export the translation tables

**What it does.** Exports the second-language names of the schedule so they can be edited outside the application or moved to another schedule. Four separate files are written into one chosen directory — rooms, programmes, subprogrammes (branches) and courses — each holding the base name and its translation. For courses only, an entry with no translation is exported with its own name repeated in the translation column.

**How to start it.** `Data ▸ Custom operation`, then type `EXPORTTRANS` and confirm; a directory-selection dialog asks where the four files should be written.

**Output / effect.** Four files in the selected directory: `roomsTrans.csv`, `programsTrans.csv`, `subprogramsTrans.csv` and `coursesTrans.csv`. Separator is a fixed semicolon. **No header line** is written in any of them. Existing files of those names are overwritten. All four have the same two-column layout.

| # | Column | Description | Example |
|---|---|---|---|
| 1 | Name | Base name of the room / programme / subprogramme / course | `Uvod v podatkovne baze` |
| 2 | Translation | Second-language name; empty when not set (courses: the base name repeated) | `Introduction to Databases` |

**Sample**

```
Uvod v podatkovne baze;Introduction to Databases
Racunalniske mreze;Computer Networks
Diskretna matematika;Diskretna matematika
```

**Notes.**

- Does not modify the schedule. Four files are overwritten in one go without any further confirmation — pick an empty directory if previous exports must be kept.
- Because a course with no translation is exported as `name;name`, the export cannot distinguish "not translated" from "translated to the same text" on the way back in.
- Unlike the other exports this one uses the plain C file API rather than the application's own file wrapper, so the character encoding of accented names may differ from the other exported files — check one file before handing it to a translator.
- The counterpart imports are `ROOMTRANS` and `COURSETRANS`.

## GENERATEPASS — Generate lecturer passwords

**What it does.** Runs the application's lecturer-password generator over the lecturers of the open schedule. The generated passwords are stored on the lecturer records; they can afterwards be written out with `EXPORTPASS` (fixed path) or `EXPORTLECT` (chosen path). The operation takes no parameters and asks nothing further.

**How to start it.** `Data ▸ Custom operation`, then type `GENERATEPASS` and confirm.

**Output / effect.** No file is written. The lecturer records receive passwords; the schedule is marked as needing to be saved, so the passwords are stored permanently only when the schedule is saved.

**Notes.**

- Run `EXPORTPASS` or `EXPORTLECT` afterwards if the passwords have to be handed out — there is no other way to read them back out in bulk.
- Independent of the workspace selection: all lecturers are processed.

## GRPSIZE — Recalculate the real size of every group

**What it does.** Recomputes the student count of each group from the actual student records. For every group it counts the students whose group list contains that group, and stores the count in the group's student number. Manually entered or imported group sizes are overwritten by the counted value, so a group with no enrolled students ends up with a size of 0.

**How to start it.** `Data ▸ Custom operation`, then type `GRPSIZE` and confirm.

**Output / effect.** No file is written. The group records are changed in memory and the schedule is marked as needing to be saved. There is no undo — the previous sizes are not kept anywhere.

**Notes.**

- Only meaningful when students have actually been imported — on a schedule that carries groups but no student records every group is set to 0, which then feeds room-capacity checks and statistics.
- Independent of the workspace selection: all groups are processed.

## LASTWEEK — Set the last week published on the web

**What it does.** Asks for a week number and stores it in the web settings table of the connected database, under the key `lastWeekForSchedule`. The web and mobile clients use this value as the cut-off: schedules after that week are not shown. Entering a value outside 1–52 — or a non-numeric value — stores 100, which is effectively "no limit".

**How to start it.** `Data ▸ Custom operation`, type `LASTWEEK` and confirm; a second prompt labelled *Week* then asks for the week number (maximum two characters).

**Output / effect.** No file is written. One SQL statement is executed on the open database connection, inserting or updating the setting `lastWeekForSchedule` with the given value.

**Notes.**

- Requires a live database connection — on a file-based schedule there is nothing to write to.
- The change takes effect on the web and mobile clients immediately, independently of whether the schedule is saved.
- The input field accepts only two characters, so a week number can never exceed 99; anything above 52 or below 1 silently becomes 100.
- Cancelling the *week* prompt is handled cleanly; the setting is only written when a value was entered.

## Appendix A — Import order for the chained families

Several codes only work if another one has been run first. These are the chains that exist; anything not listed here is standalone.

| Family | Order |
|---|---|
| Enrolments (ENRL) | `ENRLP` » `ENRLR` / `ENRLRE` » `ENRLC` » `ENRLE` » `ENRLS` or `ENRLST` |
| Medical (MED) | `MEDPROG` » `MEDVENUE` » `MEDLECT` » *(create groups)* » `MEDCLASS` |
| Curriculum and enrolments | `CURRICULUM` » `LECMAILNOTE` » `LECTOCRS` » `STUDBYCRS` / `STUDBYCRS2` / `STUDBYCRS3` |
| Rooms, lecturers and hours | `ROOMSCAP` and `LECIDMAIL` » `CRSHOURS` / `CRSHOURSBR` |
| Modular courses | `ROOMCODES` and `EXTLECIDS` » `MODCRSEVENTS` |
| Room bookings | `ROOMBOOKINGS` » `ROOMBOOKROOMS` |
| Modules and groups (MODGRP) | `MODGRPM` / `MODGRPMI` / `MODGRPB` » `MODGRPA` / `AVAIL` |
| Course codes | `CRSCODESET` or `CRSRECODE` » `STUDCRSENR` (twice — see its notes) |
| Full timetable (FULLTT) | `FULLTTSTART` for the first file » `FULLTT` for every following file |
| Standard, ini-driven | Courses2 » Groups2 » Students2 |
| Standard, alternative import | Programs (`1.csv` … `4.csv` for Courses, in that order) |

Two general rules hold across all of them:

- Rooms and lecturers before anything that places lessons.
- Programs, branches and groups before anything that allocates students.

## Appendix B — Index of every code

| Code | Kind | Chapter | Section |
|---|---|---|---|
| AVAIL | import | Lecturers | MODGRPA / AVAIL — Lecturer weekly availability |
| BRANCHCODE | import | Programmes, branches and groups | BRANCHCODE — Assign codes to existing branches by database ID |
| CHECKCRSCODE | operation | Part 3 — Custom operations | CHECKCRSCODE — List courses that have no course code |
| CLASSLABSCH | import | Placed timetables | CLASSLABSCH — Course, class and lab schedule import |
| COMMENTSUFFIX | operation | Part 3 — Custom operations | COMMENTSUFFIX — Append the configured suffix to schedule comments |
| CONVERT15 | operation | Part 3 — Custom operations | CONVERT15 — Convert the schedule to a finer time grid |
| CONVERT60 | operation | Part 3 — Custom operations | CONVERT60 — Convert the schedule to a coarser time grid |
| COURSETRANS | import | Courses and teaching load | COURSETRANS — Import translated course names |
| CRSCODESET | import | Courses and teaching load | CRSCODESET — Assign course codes to existing courses |
| CRSEMBEDDED | import | Courses and teaching load | CRSEMBEDDED — Courses with embedded code, lecturer and group allocation |
| CRSHOURS | import | Courses and teaching load | CRSHOURS / CRSHOURSBR — Courses with weeks, hours and lecturer |
| CRSHOURSBR | import | Courses and teaching load | CRSHOURS / CRSHOURSBR — Courses with weeks, hours and lecturer |
| CRSRECODE | import | Courses and teaching load | CRSRECODE — Recode and rename courses, plus English translation |
| CRSTERMLECT | import | Courses and teaching load | CRSTERMLECT — Computing lectures per term |
| CRSTURNGRP | import | Courses and teaching load | CRSTURNGRP — Course, turn and group import |
| CURRICULUM | import | Courses and teaching load | CURRICULUM — Full curriculum import: programs, branches, courses, turns and teaching load |
| DATEDLECT | import | Placed timetables | DATEDLECT — Dated lecture schedule with one group per line |
| DATEDWEEKLY | import | Placed timetables | DATEDWEEKLY — Dated events collapsed into weekly turns |
| DBBOOKINGS | import | Bookings and exams | DBBOOKINGS / DBBOOKINGSDEL — Room bookings written straight into the database |
| DBBOOKINGSDEL | import | Bookings and exams | DBBOOKINGS / DBBOOKINGSDEL — Room bookings written straight into the database |
| ENRLC | import | Courses and | ENRLC — Courses, teaching modalities, |

| Code | Kind | Chapter | Section |
|---|---|---|---|
| | | teaching load | teachers and groups |
| ENRLE | import | Bookings and exams | ENRLE — Exam course parts |
| ENRLP | import | Programmes, branches and groups | ENRLP — Programs, branches and groups |
| ENRLR | import | Rooms | ENRLR / ENRLRE — Rooms with capacity and equipment |
| ENRLRE | import | Rooms | ENRLR / ENRLRE — Rooms with capacity and equipment |
| ENRLS | import | Students and enrolments | ENRLS / ENRLSD — Students with enrolled courses (e-mail first, fixed 8 course slots) |
| ENRLSD | import | Students and enrolments | ENRLS / ENRLSD — Students with enrolled courses (e-mail first, fixed 8 course slots) |
| ENRLST | import | Students and enrolments | ENRLST / ENRLSTD — Students with enrolled courses (student-number first) |
| ENRLSTD | import | Students and enrolments | ENRLST / ENRLSTD — Students with enrolled courses (student-number first) |
| EXPORTALL | operation | Part 3 — Custom operations | EXPORTALL — Export all lessons and all bookings |
| EXPORTAVAIL | operation | Part 3 — Custom operations | EXPORTAVAIL — Export lecturer availability |
| EXPORTLECT | operation | Part 3 — Custom operations | EXPORTLECT — Export the lecturer list |
| EXPORTLESSONS | operation | Part 3 — Custom operations | EXPORTLESSONS — Export lecturer lessons with school year and term |
| EXPORTLESSONSD | operation | Part 3 — Custom operations | EXPORTLESSONSD — Export lecturer lessons with real from/to dates |
| EXPORTPASS | operation | Part 3 — Custom operations | EXPORTPASS — Export lecturer passwords to a fixed file |
| EXPORTSCHED | operation | Part 3 — Custom operations | EXPORTSCHED — Export the placed lessons per lecturer |
| EXPORTTRANS | operation | Part 3 — Custom operations | EXPORTTRANS — Export the translation tables |
| EXTCRSIDS | import | Courses and teaching load | EXTCRSIDS — Course external IDs |
| EXTLECIDS | import | Lecturers | EXTLECIDS — Lecturer external IDs |
| EXTROOMIDS | import | Rooms | EXTROOMIDS — Room external IDs |
| FULLTT | import | Placed timetables | FULLTT / FULLTTSTART — Full timetable with week lists, day lists and per-day durations |
| FULLTTSTART | import | Placed timetables | FULLTT / FULLTTSTART — Full timetable with week lists, day lists and per-day durations |
| GENERATEPASS | operation | Part 3 — Custom | GENERATEPASS — Generate lecturer |

| Code | Kind | Chapter | Section |
|---|---|---|---|
| | | operations | passwords |
| GETEXAMS | import | Bookings and exams | GETEXAMS — Exam sittings as reservations |
| GETMAILS | import | Lecturers | GETMAILS — Harvest e-mail addresses from an HTML page |
| GETMAILS_UNI | import | Lecturers | GETMAILS_UNI — Harvest e-mail addresses from a text dump |
| GRIDLEGEND | import | Placed timetables | GRIDLEGEND — Dated event grid plus its course/lecturer legend (two passes) |
| GRPSIZE | operation | Part 3 — Custom operations | GRPSIZE — Recalculate the real size of every group |
| IMPORTLECT | import | Lecturers | IMPORTLECT — Import / update lecturers (code, name, e-mail, password) |
| LASTWEEK | operation | Part 3 — Custom operations | LASTWEEK — Set the last week published on the web |
| LECIDMAIL | import | Lecturers | LECIDMAIL — Lecturers with identifier and e-mail |
| LECMAILNOTE | import | Lecturers | LECMAILNOTE — Teacher e-mail addresses and notes |
| LECPASSSET | import | Lecturers | LECPASSSET — Set lecturer passwords from a file |
| LECTOCRS | import | Lecturers | LECTOCRS — Assign teachers to courses by course code |
| MEDCLASS | import | Placed timetables | MEDCLASS — Medical lectures (timetabled events) |
| MEDLECT | import | Lecturers | MEDLECT — Medical lecturers (people) |
| MEDPROG | import | Programmes, branches and groups | MEDPROG — Medical programmes, areas and courses |
| MEDVENUE | import | Rooms | MEDVENUE — Medical venues (rooms) |
| MODCRSEVENTS | import | Placed timetables | MODCRSEVENTS — Dated events for modular courses |
| MODGRPA | import | Lecturers | MODGRPA / AVAIL — Lecturer weekly availability |
| MODGRPB | import | Placed timetables | MODGRPB — Lecturer-centred module and timetable import |
| MODGRPM | import | Placed timetables | MODGRPM / MODGRPMI — Modules, groups and locked timetable events |
| MODGRPMI | import | Placed timetables | MODGRPM / MODGRPMI — Modules, groups and locked timetable events |
| MODUL | import | Courses and teaching load | MODUL — Wide module export (programmes, years, execution types, hours) |
| PARTTIME | import | Placed timetables | PARTTIME — Dated part-time schedules |
| PROGCAT | import | Programmes, branches and | PROGCAT / PROGCATPLUS / PROGCATCRS — Programme, branch, group and course |

| Code | Kind | Chapter | Section |
|---|---|---|---|
| | | groups | catalogue |
| PROGCATCRS | import | Programmes, branches and groups | PROGCAT / PROGCATPLUS / PROGCATCRS — Programme, branch, group and course catalogue |
| PROGCATEXT | import | Bookings and exams | PROGCATEXT — External timetable grid converted into group reservations |
| PROGCATPLUS | import | Programmes, branches and groups | PROGCAT / PROGCATPLUS / PROGCATCRS — Programme, branch, group and course catalogue |
| PROGRAMCODE | import | Programmes, branches and groups | PROGRAMCODE — Assign codes to existing programs by database ID |
| ROOMBOOKINGS | import | Bookings and exams | ROOMBOOKINGS — Room bookings (reservations) |
| ROOMBOOKROOMS | import | Bookings and exams | ROOMBOOKROOMS — Rooms for imported bookings |
| ROOMCODES | import | Rooms | ROOMCODES — Room codes |
| ROOMSCAP | import | Rooms | ROOMSCAP — Rooms with code and capacity |
| ROOMTRANS | import | Rooms | ROOMTRANS — Import translated room names |
| STUDBYCRS | import | Students and enrolments | STUDBYCRS — Student enrolments per course (names in file) |
| STUDBYCRS2 | import | Students and enrolments | STUDBYCRS2 — Student enrolments with program, branch and year (no names) |
| STUDBYCRS3 | import | Students and enrolments | STUDBYCRS3 — Student enrolments with names, program, branch and year |
| STUDCRSENR | import | Students and enrolments | STUDCRSENR — Students and their course enrolments |
| STUDLIST | import | Students and enrolments | STUDLIST / STUDLISTPLUS — Student list |
| STUDLISTPLUS | import | Students and enrolments | STUDLIST / STUDLISTPLUS — Student list |
| STUDPASSSET | import | Students and enrolments | STUDPASSSET — Set student passwords from a file |

The standard menu imports of Part 1, in menu order:

- Data ▸ CSV import ▸ Rooms
- Data ▸ CSV import ▸ Tutors
- Data ▸ CSV import ▸ Tutors LDAP
- Data ▸ CSV import ▸ Programs
- Data ▸ CSV import ▸ Constraints
- Data ▸ CSV import ▸ Courses
- Data ▸ CSV import ▸ Reservations
- Data ▸ CSV import ▸ Students

- Data ▸ CSV import ▸ Exams
- Data ▸ CSV import ▸ Courses2
- Data ▸ CSV import ▸ Groups2
- Data ▸ CSV import ▸ Students2
- Exam data (button, not on the CSV import menu)

## Appendix C — Imports that can destroy data

Read the entry before running any of these.

| Code | What it removes |
|---|---|
| `DBBOOKINGSDEL` | **Every reservation in the database**, including manually entered ones, with all their group, room and lecturer links |
| `ENRLSTD` | The whole student list, before importing the file |
| `ENRLSD` | The whole student list — this only started working in this revision; before, the code did nothing |
| `ENRLC -f` | All existing branches, cascade-deleted, while processing line 1 |
| `FULLTT` / `FULLTTSTART` | All schedules and turns of each course/branch pair the file mentions |
| `FULLTTSTART` (second use in one session) | Re-cleans courses already imported earlier in the same session |
| `MODGRPA` / `AVAIL` | All existing availability reservations of every lecturer listed |
| `CRSHOURS` / `CRSHOURSBR` | A course's second course part, if it already had two |
| `ENRLE` | All turn parts of an existing exam turn with the same week range |
| `CLASSLABSCH` | Turns of the matching course part covering the same week range |
| `Data ▸ CSV import ▸ Courses` | Pre-existing turns whose week period overlaps a newly imported turn |
| `GRPSIZE` | Overwrites every group size with the counted number of students |
| `CONVERT60` | Rounds down every start slot and duration — lossy and not reversible |