/\*\*

\* V7.7.7 Data Manager (V4.31 logic, single module, root folder)

\* - Loads 17 CSV files from GitHub Raw with local fallback

\* - Parses CSV with quotes/CRLF handling

\* - Builds full V4.31 relationships + computed fields

\* - Honors Mold <-> Cutter relation via MoldDesignID first, fallback to old mapping

\* - Exposes window.DataManager for other modules (search/ui/app)

\* - No bundler required, can run from file:// or http(s) origins

\*/

(function () {

'use strict';

const GITHUB\_BASE\_URL = 'https://raw.githubusercontent.com/toanysd/MoldCutterSearch/main/Data/';

const REMOTE\_TIMEOUT\_MS = 12000;

// Registry of CSV files (14 legacy + 3 new)

const CSV\_FILES = [

{ key: 'molds', file: 'molds.csv', required: true },

{ key: 'cutters', file: 'cutters.csv', required: true },

{ key: 'customers', file: 'customers.csv', required: false },

{ key: 'molddesign', file: 'molddesign.csv', required: false },

{ key: 'moldcutter', file: 'moldcutter.csv', required: false },

{ key: 'racklayers', file: 'racklayers.csv', required: false },

{ key: 'racks', file: 'racks.csv', required: false },

{ key: 'companies', file: 'companies.csv', required: false },

{ key: 'shiplog', file: 'shiplog.csv', required: false },

{ key: 'locationlog', file: 'locationlog.csv', required: false },

{ key: 'usercomments', file: 'usercomments.csv', required: false },

{ key: 'employees', file: 'employees.csv', required: false },

{ key: 'jobs', file: 'jobs.csv', required: false },

{ key: 'processingitems', file: 'processingitems.csv', required: false },

// New tables for V7.7.7

{ key: 'CAV', file: 'CAV.csv', required: false },

{ key: 'destinations', file: 'destinations.csv', required: false },

{ key: 'statuslogs', file: 'statuslogs.csv', required: false },

];

// Internal state

const state = {

allData: {

molds: [],

cutters: [],

customers: [],

molddesign: [],

moldcutter: [],

racklayers: [],

racks: [],

companies: [],

shiplog: [],

locationlog: [],

usercomments: [],

employees: [],

jobs: [],

processingitems: [],

CAV: [],

destinations: [],

statuslogs: [],

},

maps: {},

loaded: false,

};

// Public API

const DataManager = {

get data() {

return state.allData;

},

get loaded() {

return state.loaded;

},

// Main entry

async loadAllData() {

console.time('V7.7.7 loadAllData');

const results = await Promise.all(

CSV\_FILES.map(def => fetchCSVWithFallback(def.file, def.required).then(text => {

const data = text ? parseCSV(text) : [];

console.log(`✅ ${def.file}: ${data.length} rows`);

state.allData[def.key] = data;

return true;

}))

);

if (!results.every(Boolean)) {

throw new Error('One or more required CSV files failed to load.');

}

processDataRelationships();

state.loaded = true;

// Expose for debugging

window.ALL\_DATA = state.allData;

console.timeEnd('V7.7.7 loadAllData');

// Fire event for other modules

document.dispatchEvent(new CustomEvent('data-manager:ready'));

},

// Convenience: all items merged for searching

getAllItems() {

return [...state.allData.molds, ...state.allData.cutters];

},

// Utility getters

getMaps() {

return state.maps;

},

// Re-process if external modules mutate source tables

recompute() {

processDataRelationships();

document.dispatchEvent(new CustomEvent('data-manager:updated'));

},

};

// ------------- Fetch helpers -------------

async function fetchCSVWithFallback(filename, required) {

// 1) Try GitHub Raw

try {

const controller = new AbortController();

const id = setTimeout(() => controller.abort(), REMOTE\_TIMEOUT\_MS);

const res = await fetch(GITHUB\_BASE\_URL + filename, {

// When opening from file://, CORS is typically fine for GET;

// keep it simple to ensure text() is accessible.

method: 'GET',

cache: 'no-cache',

signal: controller.signal,

});

clearTimeout(id);

if (res.ok) {

return await res.text();

}

console.warn(`GitHub fetch not OK for ${filename}: ${res.status}`);

} catch (e) {

console.warn(`GitHub fetch failed for ${filename}: ${e.message}`);

}

// 2) Fallback to same-folder relative path

try {

const resLocal = await fetch('./' + filename, { method: 'GET', cache: 'no-cache' });

if (resLocal.ok) {

return await resLocal.text();

}

console.warn(`Local fetch not OK for ${filename}: ${resLocal.status}`);

} catch (e) {

console.warn(`Local fetch failed for ${filename}: ${e.message}`);

}

if (required) {

throw new Error(`Required CSV not available: ${filename}`);

}

return '';

}

// Robust CSV parser with quotes/CRLF handling (V4.31-compatible)

function parseCSV(csvText) {

if (!csvText || !csvText.trim()) return [];

const lines = csvText.replace(/\r\n/g, '\n').replace(/\r/g, '\n').split('\n').filter(l => l.trim() !== '');

if (lines.length === 0) return [];

const headers = splitCSVLine(lines[0]).map(h => stripQuotes(h.trim()));

const rows = [];

for (let i = 1; i < lines.length; i++) {

const parts = splitCSVLine(lines[i]).map(v => stripQuotes(v));

const obj = {};

headers.forEach((h, idx) => {

obj[h] = parts[idx] !== undefined ? parts[idx] : '';

});

rows.push(obj);

}

return rows;

}

function splitCSVLine(line) {

const out = [];

let cur = '';

let inQuotes = false;

for (let i = 0; i < line.length; i++) {

const ch = line[i];

if (ch === '"' && line[i - 1] !== '\\') {

inQuotes = !inQuotes;

} else if (ch === ',' && !inQuotes) {

out.push(cur);

cur = '';

} else {

cur += ch;

}

}

out.push(cur);

return out;

}

function stripQuotes(s) {

if (s == null) return '';

return s.replace(/^"(.\*)"$/s, '$1');

}

// ------------- Relationship processing (V4.31) -------------

function processDataRelationships() {

const d = state.allData;

// Build maps

const moldDesignMap = mapBy(d.molddesign, 'MoldDesignID');

const customerMap = mapBy(d.customers, 'CustomerID');

const companyMap = mapBy(d.companies, 'CompanyID');

const rackMap = mapBy(d.racks, 'RackID');

const rackLayerMap = mapBy(d.racklayers, 'RackLayerID');

const jobByDesignMap = mapBy(d.jobs, 'MoldDesignID');

const processingItemMap = mapBy(d.processingitems, 'ProcessingItemID');

const destinationsMap = mapBy(d.destinations, 'DestinationID');

// Relation: MoldDesignID → CutterIDs (preferred), fallback to moldcutter.MoldID

const cuttersByDesign = new Map();

const moldsByDesign = new Map();

if (Array.isArray(d.moldcutter) && d.moldcutter.length > 0) {

const hasDesignKey = d.moldcutter.some(mc => 'MoldDesignID' in mc && mc.MoldDesignID);

if (hasDesignKey) {

// Build via MoldDesignID (preferred)

d.moldcutter.forEach(mc => {

const desId = mc.MoldDesignID;

if (!desId) return;

if (mc.CutterID) {

const list = cuttersByDesign.get(desId) || [];

list.push(mc.CutterID);

cuttersByDesign.set(desId, list);

}

if (mc.MoldID) {

const listM = moldsByDesign.get(desId) || [];

listM.push(mc.MoldID);

moldsByDesign.set(desId, listM);

}

});

} else {

// Fallback legacy mapping via direct ids

d.moldcutter.forEach(mc => {

if (mc.MoldID && mc.CutterID) {

const mold = d.molds.find(m => m.MoldID === mc.MoldID);

const designId = mold?.MoldDesignID;

if (designId) {

const list = cuttersByDesign.get(designId) || [];

list.push(mc.CutterID);

cuttersByDesign.set(designId, list);

}

}

});

}

}

// Helper to get latest status log for an item

function getLatestStatusLog(kind, id) {

const logs = d.statuslogs.filter(log => {

if (kind === 'MOLD') return String(log.MoldID || '').trim() === String(id || '').trim();

if (kind === 'CUTTER') return String(log.CutterID || '').trim() === String(id || '').trim();

return false;

});

if (logs.length === 0) return null;

// Prefer ISO Timestamp when present; otherwise last row order

logs.sort((a, b) => {

const ta = Date.parse(a.Timestamp || '') || 0;

const tb = Date.parse(b.Timestamp || '') || 0;

return tb - ta;

});

const latest = logs[0];

return {

...latest,

destinationInfo: destinationsMap.get(latest.DestinationID) || null,

};

}

// ------------- Process molds -------------

d.molds = d.molds.map(mold => {

const design = moldDesignMap.get(mold.MoldDesignID) || {};

const customer = customerMap.get(mold.CustomerID) || {};

const company = companyMap.get(customer.CompanyID) || {};

const rackLayer = rackLayerMap.get(mold.RackLayerID) || {};

const rack = rackLayer.RackID ? (rackMap.get(rackLayer.RackID) || {}) : {};

const storageCompany = companyMap.get(mold.storage\_company) || {};

const job = jobByDesignMap.get(mold.MoldDesignID) || {};

const processingItem = processingItemMap.get(job.ProcessingItemID) || {};

// Related cutters by MoldDesignID

const relatedCutterIDs = cuttersByDesign.get(mold.MoldDesignID) || [];

const relatedCutters = relatedCutterIDs

.map(cid => d.cutters.find(c => c.CutterID === cid))

.filter(Boolean);

// Cutline from design or CAV table (if any)

let cutlineSize = '';

if (design.CutlineX && design.CutlineY) {

cutlineSize = `${design.CutlineX}x${design.CutlineY}`;

}

if (!cutlineSize && d.CAV?.length) {

// Optional: if design.Serial or design.CAV matches CAV row, prefer CAV length/width

const key = (design.Serial || design.CAV || '').trim();

if (key) {

const cavRow = d.CAV.find(r => (r.Serial || r.CAV || '').trim() === key);

if (cavRow && cavRow.CAVlength && cavRow.CAVwidth) {

cutlineSize = `${cavRow.CAVlength}x${cavRow.CAVwidth}`;

}

}

}

// Display fields (V4.31 style)

const displayDimensions = createMoldDimensionString(mold, design);

const displayCustomer = getCustomerDisplayName(customer, company);

const storageBadge = getStorageCompanyDisplay(mold.storage\_company, companyMap);

// V4.31 status + latest statuslog (does not override text unless needed)

const v431Status = getCurrentStatus(mold);

const latestLog = getLatestStatusLog('MOLD', mold.MoldID);

return {

...mold,

// Joined

designInfo: design,

customerInfo: customer,

companyInfo: company,

rackLayerInfo: rackLayer,

rackInfo: rack,

storageCompanyInfo: storageCompany,

jobInfo: job,

processingItemInfo: processingItem,

relatedCutters,

// Computed

displayCode: mold.MoldCode || '',

displayName: mold.MoldName || mold.MoldCode || '',

displayDimensions,

displayLocation: mold.RackLayerID || '',

displayCustomer,

displayStorageCompany: storageBadge,

displayRackLocation: rack.RackLocation || '',

rackId: rackLayer.RackID || '',

drawingNumber: design.DrawingNumber || '',

equipmentCode: design.EquipmentCode || '',

plasticType: design.DesignForPlasticType || '',

moldSetupType: design.MoldSetupType || '',

pieceCount: design.PieceCount || '',

cutlineSize,

storageCompany: storageCompany.CompanyShortName || storageCompany.CompanyName || '',

storageCompanyId: mold.storage\_company || '',

moldStatus: v431Status?.status === 'available' ? 'Active' : (mold.MoldDisposing === 'TRUE' ? 'Disposed' : (mold.MoldReturning === 'TRUE' ? 'Returned' : 'In Use')),

itemType: 'mold',

// Logs

shipHistory: getShipHistory('MOLD', mold.MoldID),

locationHistory: getLocationHistory('MOLD', mold.MoldID),

currentStatus: v431Status,

latestStatusLog: latestLog,

};

});

// Index molds by design for cutter backlink

const moldIdsByDesign = new Map();

state.allData.molds.forEach(m => {

const list = moldIdsByDesign.get(m.MoldDesignID) || [];

list.push(m.MoldID);

moldIdsByDesign.set(m.MoldDesignID, list);

});

// ------------- Process cutters -------------

d.cutters = d.cutters.map(cutter => {

const customer = customerMap.get(cutter.CustomerID) || {};

const company = companyMap.get(customer.CompanyID) || {};

const rackLayer = rackLayerMap.get(cutter.RackLayerID) || {};

const rack = rackLayer.RackID ? (rackMap.get(rackLayer.RackID) || {}) : {};

const storageCompany = companyMap.get(cutter.storage\_company) || {};

// Cutline display for cutter

let cutlineSize = '';

if (cutter.CutlineLength && cutter.CutlineWidth) {

cutlineSize = `${cutter.CutlineLength}x${cutter.CutlineWidth}`;

if (cutter.CutterCorner) cutlineSize += `-${cutter.CutterCorner}`;

if (cutter.CutterChamfer) cutlineSize += `-${cutter.CutterChamfer}`;

}

// Related molds by MoldDesignID preference

let relatedMolds = [];

if (cutter.MoldDesignID && moldIdsByDesign.has(cutter.MoldDesignID)) {

const moldIds = moldIdsByDesign.get(cutter.MoldDesignID) || [];

relatedMolds = moldIds.map(mid => d.molds.find(mm => mm.MoldID === mid)).filter(Boolean);

} else if (Array.isArray(d.moldcutter) && d.moldcutter.length > 0) {

// Legacy fallback

const rels = d.moldcutter.filter(mc => mc.CutterID === cutter.CutterID);

relatedMolds = rels.map(r => d.molds.find(mm => mm.MoldID === r.MoldID)).filter(Boolean);

}

// Status and logs

const v431Status = getCurrentStatus(cutter);

const latestLog = getLatestStatusLog('CUTTER', cutter.CutterID);

// Display fields

const displayName = cutter.CutterName || cutter.CutterDesignName || '';

return {

...cutter,

// Joined

customerInfo: customer,

companyInfo: company,

rackLayerInfo: rackLayer,

rackInfo: rack,

storageCompanyInfo: storageCompany,

relatedMolds,

// Computed

displayCode: cutter.CutterNo || '',

displayName,

displayDimensions: cutlineSize,

displayLocation: cutter.RackLayerID || '',

displayCustomer: getCustomerDisplayName(customer, company),

displayStorageCompany: getStorageCompanyDisplay(cutter.storage\_company, companyMap),

displayRackLocation: rack.RackLocation || '',

rackId: rackLayer.RackID || '',

plasticCutType: cutter.PlasticCutType || '',

cutterType: cutter.CutterType || '',

bladeCount: cutter.BladeCount || '',

cutlineSize,

storageCompany: storageCompany.CompanyShortName || storageCompany.CompanyName || '',

storageCompanyId: cutter.storage\_company || '',

itemType: 'cutter',

// Logs

shipHistory: getShipHistory('CUTTER', cutter.CutterID),

locationHistory: getLocationHistory('CUTTER', cutter.CutterID),

currentStatus: v431Status,

latestStatusLog: latestLog,

};

});

// Persist maps for other modules

state.maps = {

moldDesignMap,

customerMap,

companyMap,

rackMap,

rackLayerMap,

jobByDesignMap,

processingItemMap,

destinationsMap,

};

console.log(`🔗 Processed ${state.allData.molds.length} molds & ${state.allData.cutters.length} cutters (V4.31 relationships ready)`);

}

// --- Helpers (V4.31 semantics) ---

function mapBy(arr, key) {

const m = new Map();

(arr || []).forEach(row => {

if (row && row[key] != null && row[key] !== '') m.set(row[key], row);

});

return m;

}

function createMoldDimensionString(mold, design) {

// Priority: design LxWxH -> design single field -> mold LxWxH

if (design?.MoldDesignLength && design?.MoldDesignWidth && design?.MoldDesignHeight) {

return `${design.MoldDesignLength}x${design.MoldDesignWidth}x${design.MoldDesignHeight}`;

}

if (design?.MoldDesignDim) return design.MoldDesignDim;

if (mold?.MoldLength && mold?.MoldWidth && mold?.MoldHeight) {

return `${mold.MoldLength}x${mold.MoldWidth}x${mold.MoldHeight}`;

}

return '';

}

function getCustomerDisplayName(customer, company) {

if (!customer || !customer.CustomerID) return '';

let name = customer.CustomerShortName || customer.CustomerName || customer.CustomerID;

if (company && company.CompanyShortName) {

name = `${company.CompanyShortName} - ${name}`;

}

return name;

}

function getStorageCompanyDisplay(storageCompanyId, companyMap) {

if (!storageCompanyId) return { text: 'N/A', class: 'unknown' };

const company = companyMap.get(storageCompanyId);

if (!company) return { text: 'N/A', class: 'unknown' };

const companyName = company.CompanyShortName || company.CompanyName || storageCompanyId;

// V4.31: storage\_company "2" is YSD (visual badge)

if (String(storageCompanyId) === '2') {

return { text: companyName, class: 'ysd' };

}

return { text: companyName, class: 'external' };

}

function getShipHistory(itemType, itemID) {

const d = state.allData;

if (!itemID) return [];

return d.shiplog

.filter(log => (itemType === 'MOLD' ? log.MoldID === itemID : log.CutterID === itemID))

.sort((a, b) => new Date(b.DateEntry || 0) - new Date(a.DateEntry || 0));

}

function getLocationHistory(itemType, itemID) {

const d = state.allData;

if (!itemID) return [];

return d.locationlog

.filter(log => (itemType === 'MOLD' ? log.MoldID === itemID : log.CutterID === itemID))

.sort((a, b) => new Date(b.DateEntry || 0) - new Date(a.DateEntry || 0));

}

function getCurrentStatus(item) {

if (item.MoldReturning === 'TRUE' || item.MoldReturning === true) {

return { status: 'returned', text: '返却済み', class: 'status-returned' };

}

if (item.MoldDisposing === 'TRUE' || item.MoldDisposing === true) {

return { status: 'disposed', text: '廃棄済み', class: 'status-disposed' };

}

// If last shiplog indicates shipped to external company (heuristic from V4.31)

const history = getShipHistory(item.MoldID ? 'MOLD' : 'CUTTER', item.MoldID || item.CutterID);

if (history.length > 0) {

const latest = history[0];

if (latest.ToCompanyID && latest.ToCompanyID !== 'YSD') {

return { status: 'shipped', text: '出荷済み', class: 'status-shipped' };

}

}

return { status: 'available', text: '利用可能', class: 'status-available' };

}

// Attach to window

window.DataManager = DataManager;

})();