DESTINATION
KNITTING

CSV Order Import

Administrator Guide

Destination Knitting
CSV Order Import Module

1. Overview

The CSV Order Import module allows logged-in customers to upload a CSV file listing product IDs and quantities, preview the matched items, and add them to their cart in a single operation. The feature is implemented entirely on the storefront - no Shopify app or external service is required. It ships alongside two related storefront enhancements: a mandatory PO Number field that gates checkout, and one-click 'Add all' bulk-add buttons on the product, collection, and search surfaces. The components are listed in section 2.

This document describes the installed components, the data flow at runtime, the configuration points that need to be maintained, and the operational procedures for diagnosing and resolving common issues.

2. Installed Components

snippets/cart-drawer.liquid	Cart drawer markup and the inline script that parses the CSV, resolves product information, and drives the Shopify Ajax Cart API. Modified to host the CSV import panel, the Empty Cart action, and the catalogue refresh control.
templates/collection.product-catalog.liquid	Alternate template on the virtual 'all' collection. Emits a compact JSON page of available variants with barcode, used by the front-end to build an optional UPC -> variant index when customers upload CSVs that reference barcodes instead of variant IDs.
snippets/po-number-field.liquid	Renders the required 'PO Number' field (label, input, and inline error). Bound to the cart form via the form attribute and submitted as the cart attribute attributes[PO Number]. Included in both cart-drawer.liquid and sections/main-cart-footer.liquid.
assets/dk-po-number.js	Makes the PO Number field mandatory: persists the value to the cart attribute via /cart/update.js, keeps the drawer and cart-page fields in sync, and gates every checkout entry point (standard button, dynamic checkout buttons, and form submit) while the field is empty.
assets/dk-bulk-add.js	Exposes window.dkBulkAdd and a delegated click handler for any [data-variant-ids] element. Adds variants to the cart in batches of 50 with per-item retry, then refreshes the drawer via the Section Rendering API. Loaded site-wide from layout/theme.liquid.
sections/main-product.liquid, sections/main-collection-product-grid.liquid, sections/predictive-search.liquid	Each emits an 'Add all'/'Add to cart' button carrying a data-variant-ids list of the available variants, wired to dk-bulk-add.js.
sections/main-cart-footer.liquid	Renders the PO Number field on the full cart page (form id 'cart').

.gitignore	Repository hygiene: excludes tooling directories unrelated to the theme.

3. Runtime Architecture

The end-to-end flow, from the customer clicking 'Upload CSV' to the cart being refreshed, is as follows:

1. The customer selects a local CSV file. The browser reads the file via the File API - the file itself is never uploaded to the server.
2. The inline parser normalises the header row and extracts the id_product, qty, name_product, upc, and price columns. It tolerates a variety of synonyms (id_producto, product_id, id, sku, variant_id, quantity, ...).
3. Each value is sanitised: wrapping such as ="12345" is removed, leading apostrophes are stripped, and Excel's scientific-notation truncation is detected and reported as an unrecoverable error.
4. Before rendering the preview, the front-end loads the product catalogue (see sections 4 and 5) and builds an in-memory index keyed by variantId. Every row is cross-checked against this index so the preview can mark rows whose id_product does not exist in the store.
5. The preview table is rendered inside the cart drawer, with a Status column and rows highlighted in red when the product is missing from the catalogue. Missing rows are shown to the customer but excluded from the batch that is submitted to the cart.
6. On confirmation, valid items are posted to /cart/add.js in a single batch. If any item is rejected by Shopify, the script falls back to a one-by-one retry so that each failure can be tied back to its original CSV row for the user-facing error report.
7. After all additions are complete, the cart and header bubble sections are re-fetched via Shopify's Section Rendering API so the UI reflects the new state without requiring a hard reload.

4. The Product Catalogue Endpoint

The alternate template at templates/collection.product-catalog.liquid is accessible via the URL:

```
/collections/all?view=product-catalog&page=N
```

It returns a minified JSON payload containing, for each available variant with a non-empty barcode, the following fields:

variantId	Numeric Shopify variant identifier.
upc	Value of variant.barcode, trimmed.
sku	Value of variant.sku.
title	Concatenation of product.title and variant.title.
price	Variant price, formatted with money_without_currency.
priceCents	Raw integer price in the store's subunit (cents).
available	Boolean availability flag.

The payload is paginated at 50 products per page. The front-end fetches page 1, reads the

total_pages field, and issues the remaining requests in parallel. Results are cached in sessionStorage under the key dk:productCatalog:v7 for 15 minutes.

> **Cache key versioning**
> The trailing :v7 segment is a deliberate invalidation marker. If you change the shape of the catalogue payload, increment the suffix to force all active customer sessions to discard stale caches on next load. (It was last bumped to v7 when the Accept: application/json header was removed from the template fetch, because some B2B setups routed that request through a JSON-API layer that 404s on theme views and forced the metafield-less products.json fallback.)

5. Fallback Data Source

If the alternate template is not yet deployed to the published theme, the fetch returns HTTP 404. The script then falls back automatically to Shopify's built-in /products.json endpoint, which is always available. The fallback is paginated 250 products per page and stops when an empty page is returned.

> **Operational recommendation**
> For optimal performance and payload size, ensure that the alternate template is included in every published or live theme. The fallback is functional but emits significantly more data and has fewer guarantees about which products are returned in B2B contexts.

6. Download Controls

Two download options are exposed to the customer inside the CSV import panel. Both are implemented client-side.

6.1 Download sample CSV (dynamic)

A button that builds a small illustrative file on-demand in the customer's browser. It draws on the same product catalogue as the full export (sections 4 and 5), picks roughly 15 real products - preferring available ones that have a UPC - and downloads them as 'products-template-sample.csv'. If the picked products are missing barcodes (for example when the products.json fallback is in use), the script enriches just those rows from /products/<handle>.js so the sample still shows realistic UPCs.

> **No admin maintenance required**
> The sample is generated client-side from live data, so there is no static file to host or keep in sync. (Earlier versions linked to a pre-built CSV on the Shopify CDN; that approach has been removed.)

6.2 Generate full catalog CSV (dynamic)

A button that builds the CSV on-demand in the customer's browser by fetching the product catalogue from the endpoints described in sections 4 and 5. This is the recommended, always-fresh option - no maintenance on the admin side is required.

The generated CSV contains seven columns:

id_product	The Shopify variant identifier, wrapped as ="..." to survive Excel.
name_product	The product title, with any trailing '- Default Title' stripped.
upc	The variant barcode, wrapped as ="...".
price	The variant price in the store's active currency.
suggested_retail_price	The product metafield custom.suggested_retail_price (Suggested Retail Price).
wholesale_bag_size	The product metafield custom.wholesale_bag_size.
qty	Left blank, for the customer to fill in.

Operational recommendation
Direct customers to the 'Generate full catalog CSV' button whenever possible. It always reflects the current state of the store, avoids the manual replace-on-CDN step, and produces a richer file (product name, UPC, and price) than the static sample.

7. Cart API Integration

Adds are performed against Shopify's Ajax Cart endpoints using standard POST requests:

POST /cart/add.js	Adds one or more items. Used by the CSV import, the 'Add all' buttons, and the per-row/per-item retry path.
POST /cart/clear.js	Invoked by the Empty Cart action. Removes every line item from the cart.
POST /cart/update.js	Used by dk-po-number.js to persist the 'PO Number' cart attribute (debounced on input, and immediately on blur).
GET /cart?sections=...	Used by the Section Rendering API to re-render the cart drawer and the cart icon bubble after any mutation.

8. PO Number Enforcement

A purchase-order number is mandatory on every order. The field is rendered by snippets/po-number-field.liquid in two locations - the cart drawer (form id 'CartDrawer-Form') and the cart page footer (form id 'cart') - and submitted as the cart attribute attributes[PO Number]. The behaviour is driven entirely by assets/dk-po-number.js:

- Persistence - the value is written to the cart attribute via /cart/update.js, debounced ~500 ms on input and flushed immediately on blur, so it survives page navigation and reaches the order even through dynamic checkout buttons.
- Synchronisation - the drawer field and the cart-page field are kept in lock-step, so the customer only fills it in once.
- Gating - a capture-phase click handler intercepts #checkout, #CartDrawer-Checkout, .cart__dynamic-checkout-buttons, and any [data-po-gate] element; a capture-phase submit handler intercepts the 'cart' and 'CartDrawer-Form' forms. While the field is empty the event is cancelled, the inline error is shown, and focus is moved to the field.

The gate is client-side only

dk-po-number.js blocks the storefront entry points, but it cannot stop a checkout reached by other means. If a hard server-side guarantee is required, enforce the presence of the 'PO Number' attribute with a Shopify Function (checkout validation) or a checkout UI extension as well.

9. One-Click Bulk-Add Buttons

assets/dk-bulk-add.js (loaded site-wide from layout/theme.liquid) exposes window.dkBulkAdd(variantIds, btn, opts) and a delegated click handler. Any element carrying a data-variant-ids attribute (a comma-separated list of variant IDs) becomes a trigger, so server-rendered and dynamically rendered markup both work. Three call sites ship with the theme:

sections/main-product.liquid	'Add all N variants to cart' below the buy buttons, shown when the product has more than one available variant.
sections/main-collection-product-grid.liquid	'Add all N SKUs in this collection to cart' at the top of the collection grid.
sections/predictive-search.liquid	A per-result 'Add to cart' button in the search dropdown.

Items are posted to /cart/add.js in batches of 50. If a batch is rejected the script retries that batch one item at a time, so a single bad variant does not block the rest. After the additions complete it refreshes the drawer through the Section Rendering API (cart-drawer, cart-icon-bubble) and clears the host's is-empty class, falling back to a full reload only if the section render is unavailable.

10. Configuration & Maintenance Checklist

- Confirm that every saleable product has a populated Barcode field on its variants if you want customers to be able to upload CSVs that use UPCs rather than variant IDs.
- Publish the alternate template on every live theme so the catalogue endpoint is served.
- The sample and full-catalogue CSVs are both generated client-side from live data - there is no static template file to maintain.
- If you deploy significant changes to the module, bump the dk:productCatalog cache key suffix in cart-drawer.liquid so customer sessions discard outdated caches on their next visit.
- PO Number is enforced on the storefront only. If you need a hard guarantee, add a checkout validation Function or UI extension that requires the 'PO Number' attribute.
- Review the browser console of a real customer account when testing. Both the success and the error paths log enough information to reason about the cause.

11. Troubleshooting

11.1 All rows are marked 'Not found' in the preview

Either the catalogue cache is stale or the customer's CSV uses a different identifier scheme than the one expected. Ask the customer to click 'Refresh catalog' (or clear their browser storage) and retry. If the problem persists, confirm that the alternate template is published in the live theme - when the

template 404s the fallback reads from /products.json, which does not expose variant barcodes and will only find matches against variant IDs.

11.2 'Missing columns' error

The CSV header row does not contain any of the recognised column names. If the listed columns look like a Shopify product export (Handle, Variant SKU, ...) the user uploaded the wrong file - direct them to the Generate full catalog CSV button instead. Otherwise confirm the file was saved as CSV (not XLSX) and that the first line contains a header.

11.3 CSV opens corrupted in Excel / customer asks about ="..."

Excel coerces long numbers to scientific notation, which silently drops the last digits when the file is saved. To prevent this, every id_product and upc cell in the generated CSV is wrapped as ="..." - this is a standard spreadsheet directive that forces the cell to be treated as text. Customers occasionally ask whether this is a bug; it is not. The wrapper is removed automatically by Excel when the file is saved, and the import parser also strips ="..." and any leading apostrophe before validating the row, so files survive a round trip without manual intervention.

If a customer reports IDs that have been truncated, it is because they removed the wrapper, opened the file in an environment that ignored it, or pasted IDs into cells that were not pre-formatted as text. Direct them to re-download the catalogue CSV, edit only the qty column, and re-upload. The Customer Guide (section 5) explains the wrapper in plain English.

11.4 Cart does not refresh after upload

After a successful batch the script re-renders the drawer through the Section Rendering API and only falls back to a full location.reload() if that fails. If the cart drawer continues to show stale contents, inspect the network tab for failed calls to /cart/add.js or /cart?sections=.... A change to the cart drawer section ID would break the Section Rendering API request.

11.5 Customer cannot check out / checkout button does nothing

This is almost always the PO Number gate (section 8): the field is empty, so dk-po-number.js is cancelling the checkout click or form submit. Confirm the inline 'Please enter a PO number to continue' error is showing. If the gate fires even when a value is present, check that the input still carries the data-po-number-input attribute and that /cart/update.js calls are succeeding in the network tab.

12. Security & Privacy Considerations

- CSV parsing is performed entirely in the customer's browser. The file itself is never transmitted.
- Only the extracted product IDs and quantities are sent to Shopify, through the standard authenticated customer session.
- The /cart/add.js and /cart/clear.js endpoints enforce the same quantity rules, customer segmentation, and pricing logic as the standard PDP. The module does not introduce any additional trust surface.