Where Custom Checkout Field Data Is Stored (Order Meta Reference)

References · https://wpfactory.com/docs/woocommerce-checkout-field-order-meta/

When a customer submits an order, Custom Checkout Fields for WooCommerce saves each field value into the order as order meta. This reference explains the WooCommerce order meta keys it writes, the consolidated data array it also keeps, and the helper functions for reading and updating that data in code.

The per-field meta key

Each enabled field with a submitted value is stored under its own order meta key in this shape:

```
_<section>_<fields_id>_<field_number>
```

- **section** is the field's Section: `billing`, `shipping`, `account`, or `order`.
- **fields_id** is the value of the **Fields ID** option (`alg_wc_ccf_key`), default `alg_wc_checkout_field`.
- **field_number** is the field index, starting at 1.

So the first field, left in the Billing section with the default Fields ID, is stored under:

```
_billing_alg_wc_checkout_field_1
```

For a field set to **Duplicate** per product, each duplicate adds a suffix to the key so the copies do not collide.

The consolidated data array

Alongside the individual keys, the plugin stores one array under the meta key `_alg_wc_ccf_data`. This array holds one entry per saved field, and it is what the order, email, and admin displays read from. Each entry includes:

Key	Holds

`section`	The field section (billing, shipping, account, order).
`type`	The field type key, for example `text` or `select`.
`label`	The field label.
`_value`	The submitted value (an array for Multiselect).
`_value_meta_key`	The per-field meta key described above.
`_field_nr`	The field number.
`_key`	The Fields ID in force when the order was placed.
`_version`	The plugin version that saved the entry.

Duplicate fields also carry `_key_suffix` and `_label_suffix` so each copy keeps its own key and label.

Helper functions

Two functions give you safe access to the consolidated array. They work with both classic post-based orders and HPOS custom order tables, because they go through `wc_get_order()`.

Function	Returns / does
`alg_wc_ccf_get_order_fields_data( $order_id )`	Returns the `_alg_wc_ccf_data` array for an order.
`alg_wc_ccf_update_order_fields_data( $order_id, $fields_data )`	Writes a new array back to the order and saves it.

Read every custom field value on an order

```
$fields = alg_wc_ccf_get_order_fields_data( $order_id );
if ( ! empty( $fields ) ) {
    foreach ( $fields as $field ) {
        $value = is_array( $field['_value'] ) ? implode( ', ', $field['_value'] ) :
$field['_value'];
        echo esc_html( $field['label'] . ': ' . $value );
    }
```

```
}
```

Read a single value directly by its meta key

```
$order = wc_get_order( $order_id );
$value = $order->get_meta( '_billing_alg_wc_checkout_field_1' );
```



CUSTOM CHECKOUT FIELDS FOR WOOCOMMERCE

See every feature and the pricing on the Custom Checkout Fields for WooCommerce plugin page.

View plugin →

> **Note:** if you change the **Fields ID** option after orders exist, new orders use the new key while older orders keep the key they were saved with. Read historical orders through `alg_wc_ccf_get_order_fields_data()`, which carries each order's original key in `_value_meta_key`.

Need a hand?

Our team answers every ticket on the Custom Checkout Fields support forum.

Related articles

- Custom Checkout Fields in Emails and Order Pages: Templates and Placeholders
- Where to Position Your Custom Fields in WooCommerce Checkout
- Checkout Field Types Reference
- Additional Checkout Fields: Developer Hooks and Filters Reference
- Shortcodes Available in Custom Checkout Fields