Reference Manual

Prices Changer for Magento 2.x — Extmag_Pricer

2026-05-20

1 Service contracts (PHP API)

The module ships PHP-only service contracts. There is no `webapi.xml`, so the APIs below are not exposed over REST / SOAP — consume them via Magento DI from your own modules or scripts.

1.1 `Extmag\Pricer\Api\ScheduleRepositoryInterface`

CRUD for Schedule entities.

```php
public function save(ScheduleInterface $schedule): ScheduleInterface;
public function getById(int $scheduleId): ScheduleInterface;
public function getList(SearchCriteriaInterface $criteria): ScheduleSearchResultsInterface;
public function delete(ScheduleInterface $schedule): bool;
public function deleteById(int $scheduleId): bool;
```

1.2 `Extmag\Pricer\Api\Data\ScheduleInterface`

Schedule entity accessors.

```php
public function getTitle(): ?string;
public function isEnabled(): bool;
public function getScheduleType(): ?string;     // one_time | yearly | monthly | weekly | custom_cron
public function getApplyDate(): ?string;
public function getApplyTime(): ?string;
public function getRollbackDate(): ?string;
public function getRollbackTime(): ?string;
public function getTimezone(): ?string;
```

Each getter has a matching setter (`setTitle`, `setIsEnabled`, etc.). Cron-mode schedules use `getCronExpression()` / `setCronExpression()` and the matching `...RollbackCronExpression()` pair.

1.3 `Extmag\Pricer\Api\Data\ScheduleLinkInterface`

Per-pair execution state for the (Set, Schedule) M:N junction.

Fields: `set_id`, `schedule_id`, `status` (`pending` / `active` / `completed` / `failed` / `cancelled`), `last_applied_at`, `last_rollback_at`, `last_error`, `attempts`.

1.4 `Extmag\Pricer\Api\Data\SetsInterface`

Pricer Set entity. Accessors are paired per price type — each one has a *flag* column (`getPrice` / `setPrice`) controlling whether the type is touched at all, and a *value* column (`getPriceValue` / `setPriceValue`) carrying the formula. Covered price types: `Price`, `SpecialPrice`, `Cost`, `Msrp`, `CustomOptionsFixed`, `CustomOptionsPercent`, `Tier`, `CustomPrices`. Common fields: `status`, `stores`, `is_price_global`, `actions_serialized`, `created_time`, `update_time`.

2 System configuration reference

`Stores → Configuration → Extmag → Prices Changer` (`section_id = extmag_pricer`).

2.1 General (`extmag_pricer/general/...`)

Field	Default	Source model
`do_reindex`	Yes	`Magento\Config\Model\Config\Source\Yesno`
`indexes`	empty	`Extmag\Pricer\Model\Source\Indexes` (multiselect)

2.2 Developer Settings (`extmag_pricer/developer/...`)

Reserved for engineers — wrong choice can hold table-level locks for hours on a large catalog or leak rule semantics.

Field	Default	Source model
`load_all_attributes`	No	`Yesno`
`apply_mode`	`atomic`	`Extmag\Pricer\Model\Source\ApplyMode` (`atomic` / `chunked`)

See Apply transaction modes for the trade-off.

2.3 Price Change Log (`extmag_pricer/price_log/...`)

Field	Default	Source model
`enabled`	No	`Yesno`
`tracked_price_types`	`scalar`, `tier`, `custom_option`, `custom_price`	`Extmag\Pricer\Model\Source\PriceLogPriceType` (multiselect)
`tracked_scalar_fields`	`price`, `special_price`, `cost`, `msrp`	`Extmag\Pricer\Model\Source\PriceLogScalarField` (multiselect)
`retention`	`Do not clean`	`Extmag\Pricer\Model\Source\PriceLogRetention` (1, 2, 3, 4, 5 yrs)

2.4 Schedules (`extmag_pricer/schedule/...`)

Field	Default	Source model
`enabled`	No	`Yesno`
`batch_size`	`50`	text (`validate-digits validate-greater-than-zero`)
`default_timezone`	inherit `general/locale/timezone`	`Extmag\Pricer\Model\Source\Timezone`
`notify_on_failure`	No	`Yesno`
`notification_emails`	empty	text (comma-separated)
`failure_email_template`	`extmag_pricer_schedule_failure`	`Magento\Config\Model\Config\Source\Email\Template`
`failure_email_sender`	`general`	`Magento\Config\Model\Config\Source\Email\Identity`

3 Cron jobs

Job	Schedule	Purpose
`extmag_pricer_run_schedules`	`* * * * *`	Per-minute Schedule driver. Apply + rollback pass.
`extmag_pricer_purge_price_log`	`0 3 * * *`	Daily retention prune of the Price Change Log table.

Both jobs live in the Magento `default` cron group. The Schedule driver is gated by `extmag_pricer/schedule/enabled`; the purge job is gated by `extmag_pricer/price_log/retention != none`.

Lock acquisition for the Schedule driver is managed by `LockManagerInterface` under the lock name `extmag_pricer_schedule_runner`. Concurrent cron nodes will not double-fire.

4 Schedule types and cron expressions

4.1 Types

Type	Triggers on
`one_time`	Single firing at `apply_date apply_time` in the schedule timezone.
`yearly`	Same `MM-DD HH:MM:SS` every year.
`monthly`	`day_of_month` every month — clamped to the last day of months that don't reach the requested day.
`weekly`	`day_of_week` (0 = Sunday, 6 = Saturday) every week.
`custom_cron`	5-field cron expression evaluated against the schedule timezone.

4.2 Cron expressions

`custom_cron` schedules are parsed by `dragonmantank/cron-expression ^3.3`. Apply and rollback have **separate** expressions — leaving rollback empty keeps the prices applied indefinitely.

```
0 9 * * 1       every Monday at 09:00
*/15 * * * *    every 15 minutes
0 0 24 11 *     at midnight on November 24
@daily          shorthand for 0 0 * * *
```

4.2.1 Rollback gating for `* * * * *`

A `custom_cron` schedule with apply expression `* * * * *` and **no** rollback expression is supported. The driver gates the auto-reset on `hasRollbackConfigured`, so a no-rollback recurring schedule applies once per pair and stays `completed`, avoiding an infinite re-apply loop.

4.3 Timezone and DST

`timezone` is an IANA identifier per schedule. `apply_date` / `apply_time` are interpreted as local times in that timezone, then converted to UTC for the cron index.

DST transitions are delegated to PHP `DateTimeZone`. At spring-forward / fall-back boundaries the firing time may shift by ±1 hour. Avoid scheduling between 02:00 and 03:00 in DST-observing timezones when exact timing is critical.

5 Apply transaction modes

`extmag_pricer/developer/apply_mode`.

Mode	Transaction	Crash recovery	Frontend visibility
`atomic` (default)	Single `beginTransaction` / `commit` wraps the whole apply	Full rollback — Set stays `pending`, no partial writes	Frontend never sees mid-apply state
`chunked`	Each page (~500 products) commits independently	Earlier pages committed; resume cursor on `applied_progress_page`	Frontend sees mixed old / new prices during the apply

5.1 When to switch to `chunked`

- Catalog > 100k products.
- Apply pass takes more than a few minutes wall-clock.
- Long row locks on `catalog_product_entity_decimal` block admin product saves and REST writes during apply.

Trade-off: the all-or-nothing guarantee is gone. A purchaser browsing during a chunked apply can see a half-applied catalog. Acceptable for mass repricing, not acceptable when timing is contractually critical.

5.2 Resuming a partial apply

```
SELECT entity_id, title, status, applied_progress_page
FROM extmag_pricer_sets
WHERE applied_progress_page IS NOT NULL;
```

Non-NULL cursor + `status = 'pending'` = partial apply. Options:

- **Cron-driven** — the link state machine retries on the next firing; the cursor skips already-committed pages.
- **Manual** — operator Apply / Reset action. Same skip-and-resume.
- **Discard and restart** — Set rollback reverses every dump row written so far, clears `applied_progress_page`, sets status back to `pending`. Next apply starts at page 1.

6 Price Change Log schema

Table `extmag_pricer_price_log` — the only schema column reference you should need to query the audit log directly from BI / reporting / integration code. Other module tables are considered internal and are documented in source code.

Column	Type	Notes
`entity_id`	bigint unsigned, AI	Primary key. Hidden in the grid by default.
`product_id`	int unsigned	Resolved by SKU when missing (import path).
`sku`	varchar(64)	
`store_id`	smallint unsigned	0 = admin / all stores.
`price_type`	varchar(32)	`scalar`, `tier`, `custom_option`, `custom_price`.
`price_field`	varchar(64)	Field key inside the type (e.g. `price`, attribute code, tier op).
`price_meta`	text	JSON with the qualifying tuple. Shape varies per price type — see below.
`old_value`	decimal(20,4)	NULL when unknown (import / API).
`new_value`	decimal(20,4)	NULL on tier / custom_option deletion.
`currency`	varchar(3)	Base currency of the row's store.
`source`	varchar(32)	See Source attribution tags.
`source_ref`	varchar(128)	Set id, plugin tag, or null.
`admin_user_id`	int unsigned	NULL when no backend session.
`admin_username`	varchar(64)	Snapshot at write time — readable after the user is deleted.
`created_at`	timestamp	Defaults to `CURRENT_TIMESTAMP`.

Indexes: `created_at`, `(product_id, created_at)`, `sku`, `admin_user_id`, `source`, `price_type`.

6.1 `price_meta` JSON shape

`price_type`	Keys present in `price_meta`
`scalar`	none (the column on the product is identified by `price_field`)
`tier`	`customer_group_id`, `qty`, `website_id`, `price_type` (`fixed` / `percent`)
`custom_option`	`option_id`, `option_type_id`, `price_type` (Pricer Set only)
`custom_price`	`price_field` holds the attribute code (Pricer Set only)

7 Source attribution tags

Used in the `source` column of `extmag_pricer_price_log`. Filter the grid or your BI query by these tags to slice by origin.

Tag	Trigger
`manual`	Admin product edit, or admin *Update Attributes* mass action.
`pricer`	Pricer Set apply (Schedule or operator-driven). `source_ref` = Set id.
`pricer_rollback`	Pricer Set rollback. `source_ref` = Set id.
`import`	CSV product import through the native Magento ImportExport flow.
`api`	REST tier-price storage (bulk and single), `ProductRepository::save` from API.
`cli`	CLI / cron via service contracts, async REST consumer running from CLI.
`system`	Unidentified context (no admin session, no HTTP request, no CLI flag).

8 Performance and scale notes

- Price Log writers append rows to a request-scoped in-memory buffer. One batched `INSERT MULTIPLE` runs at request shutdown — no per-save round trip to the database. When the buffer reaches 500 rows it flushes early and keeps buffering, bounding memory on very large mass actions.
- Config reads (`PriceLogConfig`) are memoised per-request so hot paths don't pay for repeated `ScopeConfig` round trips.
- The Schedule cron driver scans a denormalised `next_apply_at_utc` / `next_rollback_at_utc` index — the per-minute pass stays under a few milliseconds regardless of how many schedules exist.
- `RulesApplier` streams products in chunks of *Batch Size* (`extmag_pricer/schedule/batch_size`, default 50). Lower this on memory-constrained environments.

8.1 Approximate grid count

The Price Log grid pagination footer uses `information_schema.TABLES.TABLE_ROWS` for unfiltered queries — InnoDB does not maintain an exact row counter and a bare `COUNT(*)` on a million-row audit log can take seconds. Estimate is accurate to ~80%, fine for *page X of N* math. Any active filter falls through to the parent implementation so paging stays correct.

9 Adobe Commerce vs Open Source

The Price Log grid joins the admin-scope product name from `catalog_product_entity_varchar`. Adobe Commerce keys that table on `row_id` (staging support); Open Source uses `entity_id`. The collection auto-resolves the link field through `MetadataPool` and routes the join through `catalog_product_entity` accordingly — same code works on both editions without conditional configuration.

If the Price Log grid 500s with a SQL error referencing an unknown column, clear `generated/` and re-run `setup:di:compile` — the MetadataPool entry is built at compile time.

10 Known limitations

- **Direct SQL writes** by third-party modules, raw migrations or install / upgrade scripts that bypass both ORM events and EAV service contracts are invisible to the log.
- **Catalog Pricing Rules** (`Magento_CatalogRule`) compute discounts at runtime and write to `catalogrule_product_price`. They do not modify the product's own `price` / `special_price` attributes, so they are not — and shouldn't be — logged here.

- **Old value during import** is not captured. The CSV importer surfaces only the new values per bunch row; pre-fetching all SKUs would defeat responsiveness.
- **Old value during REST tier-price storage** likewise is not pre-fetched. Prior values can be reconstructed from earlier log rows.
- **PHP fatal mid-burst** can lose up to 500 buffered rows (one batch). Acceptable for an audit log because the surrounding save itself may not have committed cleanly either.
- **GraphQL mutations** do not exist for catalog prices in CE / EE, so the log path does not cover them.