ChatGPT

Frappe & ERPNext v14/v15 Developer Cheatsheet

Server-Side

Document Lifecycle Hooks

Hook Name	When Triggered	Notes (v14 vs v15)
`before_insert`	Before a new document is inserted (before DB commit) [1].	
`before_naming`	Before naming the document (before setting `doc.name`) [2].	
`autoname`	During naming; custom naming rules (set `doc.name`) [1].	
`before_validate`	Before validation runs (before mandatory checks) [3].	
`validate`	After `before_validate`, before save (usually custom checks) [4].	
`before_save`	Before document save (before update/insert) [5].	
`on_update`	After any update (before submit) [6].	
`before_submit`	Before submitting a document (docstatus 0→1) [7].	
`on_submit`	After submission (docstatus becomes 1) [7].	
`before_cancel`	Before cancelling (docstatus 0/1→2) [8].	
`on_cancel`	After cancelling (docstatus=2) [8].	
`before_update_after_submit`	Before changing a submitted document (after on_submit) [9].	
`on_update_after_submit`	After modifying a submitted document (post-`before_update_after_submit`) [9].	
`on_change`	Whenever the document is saved (after any change) [8].	
`after_insert`	After insert into DB (on any document) [10].	

Hook Name	When Triggered	Notes (v14 vs v15)
`before_rename` / `after_rename`	Before/after renaming a document (changing `name`).	
`on_trash` / `after_delete`	Before/after deleting a document (when *Submit* isn't used, usually for namesakes) [11].	

(Frappe v15 added Type Annotations for query reports/controllers (see v15 Dev Notes) [12].)

Core Document Attributes

Attribute	Description
`name`	Unique ID (usually primary key or naming series) [13].
`owner`	The user who created the document [13].
`creation`	Timestamp when created (auto-set) [14].
`modified`	Timestamp of last modification (auto-updated) [13].
`modified_by`	User who last modified (auto-updated) [14].
`docstatus`	Workflow status: 0 = Draft, 1 = Submitted, 2 = Cancelled [15].
`workflow_state`	Current state in a Workflow (if one is applied) (string).

Note: Fields like *creation, owner, modified, modified_by* are added by default [14]. Use `doc.workflow_state` if the doctype has a Workflow.

Common Document Methods

Method	Usage / Description
`doc.save()`	Save or update document (runs validations). Triggers `validate` → save → `on_update`. [16]
`doc.insert()`	Insert new document (alias of save when `name` is new).
`doc.submit()`	Submit a DocType (if `is_submittable=1`), changing `docstatus` to 1.
`doc.cancel()`	Cancel a submitted doc (`docstatus` to 2).
`doc.reload()`	Reload document from DB (refresh fields) [17].

Method	Usage / Description
`doc.db_set(field, value, ...)`	Update a field directly in DB (no validation, updates `modified`) [18].
`doc.delete()`	Delete document (if allowed).
`doc.check_permission(perm)`	Check if current user has a permission.
`doc.append(child_table, vals)`	Append a new row to a child table.
`doc.get_url()`	Get URL for this document (desk or web) [19].
`doc.add_comment(...)`	Add a comment to document.
`doc.run_method(method, args)`	Call any controller method (runs hooks/ permissions).
`frappe.db.set_value(doctype, name, field, value)`	Set field in DB (no validation; alias of `update`) [20].
`frappe.db.get_value(doctype, name/ filters, field)`	Get field value(s) from DB [21].

Hooks in `hooks.py`

Hook	Purpose / Usage
`doc_events`	Map DocType to event handlers (e.g. `"MyDoctype": {"on_update": "app.module.method"}`) [22]. Allows hooking CRUD events for docs.
`override_whitelisted_methods`	Override core whitelisted (HTTP) methods (last override wins in v15) [23] [24]. E.g. override `frappe.client.get_count`.
`scheduler_events`	Define periodic tasks (hourly, daily, etc). Example: `"hourly": ["app.tasks.hourly_job"]` [25]. v15 also supports long-running queues (`"hourly_long"`, etc) [26].
`ignore_links_on_delete`	List doctypes to ignore when deleting (skip linking validation) [27].
`jinja`	Add custom Jinja methods/filters (for print/email templates) [28].

Hook	Purpose / Usage
`fixtures`	(v14 only) Sync Data like Custom Fields or Reports as JSON fixtures. In v15, `custom_scripts` fixture is removed (use multiple scripts instead) [29] .
`override_doctype_class`	Override a DocType's Python class (e.g. provide a custom Controller).
Other hooks: `templates/includes`, `doc_events`, `permission_query_conditions`, `notification_config`, etc. (See Frappe Hooks docs) [22] .	

Background Job Patterns

- `frappe.enqueue` – Enqueue a function to run in background (async). E.g.:

  ```
  frappe.enqueue('myapp.tasks.do_work', arg1=val, queue='long', timeout=300)
  ```

  This schedules a job on the worker queue (`queue='long'` for heavy tasks) [30] . Returns a job ID.
- `frappe.enqueue_doc` – Enqueue a method on a document: `frappe.enqueue_doc(doctype, name, method, job_name, timeout)`.
- `frappe.utils.background_jobs.long` – (v14) older "long job" mechanism. In v14+, use `enqueue(queue='long')` or scheduler's *_long events.
- **Result Handling:** Workers process jobs (see *background jobs* in docs [30]). Use `frappe.get_last_doc` etc.

REST API Endpoints

Standard REST resource endpoints (JSON via `/api/resource`) [31] [32] :

HTTP Method	Endpoint	Description / Example
GET	`/api/resource/<DocType>`	List records (optionally filter via query parameters) [31] . **Example:** `GET /api/resource/Task?fields=["name","subject"]` returns list of Tasks.
POST	`/api/resource/<DocType>`	Create new doc. Send JSON body (doc fields) [31] . **Example:** `POST /api/resource/Task` with JSON `{"subject":"New Task"}` creates a Task.

HTTP Method	Endpoint	Description / Example
GET	`/api/resource/<DocType>/<name>`	Fetch a document by name [32] . **Example:** `GET /api/resource/Task/TASK-0001`.
PUT	`/api/resource/<DocType>/<name>`	Update a document. JSON body of changed fields [33] . **Example:** `PUT /api/resource/Task/TASK-0001` with `{"status":"Closed"}`.
DELETE	`/api/resource/<DocType>/<name>`	Delete a document (if permitted) [34] . **Example:** `DELETE /api/resource/Task/TASK-0001`.
GET	`/api/resource/<DocType>?filters=...`	List with filters. (e.g. `?filters=[["status","=","Open"]]` returns open docs.)
Misc: GET `/api/method/<module>.<function>` to call whitelisted methods; GraphQL endpoints (`/api/graphql`) available in newer versions.		

CLI / Bench Commands

Command	Description
`bench init <bench>`	Initialize a new bench directory (sets up sites/ env).
`bench new-app <app>` [35]	Create a new Frappe app template.
`bench new-site <site>` [36]	Create a new site (database) under the bench.
`bench get-app <repo> [app-name]`	Download and install an app from a Git repo.
`bench install-app <app>`	Install an app on the current site (after `get-app`).
`bench update` [37] / `bench update --patch`	Update all apps & sites: pulls repos, applies patches, rebuilds assets.
`bench migrate` [38]	Run schema migrations and patches for all sites.
`bench start` [39]	Start the development server (runs all processes via Procfile).
`bench serve`	Run only the web server (without background workers).

Command	Description
`bench console` [40]	Open a Python console in the bench env (with `frappe` imported).
`bench --site <site> <command>`	Run commands for a specific site (e.g. `--site site1 migrate`).
`bench update --requirements`	Update Python/Node requirements.
`bench drop-site <site>`	Delete a site (removes DB & files) [41].
`bench set-config -g key value`	Set config in `common_site_config.json`.
`bench doctor`	Show status of background workers / scheduler.

(See [Bench Commands Cheatsheet] [42] [43] *for more.)*

Client-Side

Form Event Triggers

Event (Client-Script)	When Triggered
`refresh`	After form is loaded or refreshed [44].
`onload` / `before_load`	When form is initializing (before/after render) [44].
`validate`	Before saving the form (runs before server-side validate) [44].
`before_save`	Right before client saves (after `validate`) [45].
`before_submit`	Before submitting the doc (client-side) [46].
`on_submit`	After document is submitted [46].
`before_cancel`	Before cancel (client-side) [47].
`after_cancel`	After cancel.
`frm.custom_event`	Any custom event defined via `frm.trigger()`.
> *Also:* In child tables: `{fieldname}_add`, `_remove`, `_move` events (e.g. `items_add(frm, cdt, cdn)`) fire when rows change [48].	

(These can be set via `frappe.ui.form.on('Doctype', { refresh: function(frm){...}, validate: function(frm){...}, on_submit: function(frm){...} })`*.)*

Field-Level Events

- `fieldname_onchange` – A handler for a specific field. E.g.:

```
frappe.ui.form.on('Task', {
  status(frm) {
    // runs whenever status field changes
    console.log('Status changed:', frm.doc.status);
  }
});
```

  (Frappe auto-sets `frm` and the changed field.) [49] .
- **Custom triggers:** Any method defined in `frappe.ui.form.on` can be triggered manually via `frm.trigger('method_name')`.

Common Client API / Helper Functions

Function	Description / Example
`frappe.call(options)` [50]	Async server call (AJAX). Provide `{method:'app.mod.func', args:{}, callback:func}` [51] . Returns a Promise.
`frm.set_value(field, value)` [52]	Set a field's value on the form (client-side); triggers change event. Supports object for multiple fields.
`frm.toggle_display(fieldnames, condition)` [53]	Show/hide fields: e.g. `frm.toggle_display(['priority','due_date'], frm.doc.status==='Open')` [53] .
`frm.refresh_field(fieldname)`	Rerender a field (and its dependencies) on the form.
`frappe.db.get_value(doctype, filters, field, callback)`	Get a field value from DB (async). e.g. `frappe.db.get_value('User', {'email':email}, 'full_name', r=>{ ... })`.
`frappe.show_alert(message)`	Show a temporary alert/toast message.
`frappe.msgprint(message)`	Show a popup message.
`frappe.confirm(msg, yes, no)`	Confirmation dialog.
`frappe.set_route()`	Navigate (set route) in Desk UI.
`cur_frm.add_fetch(link_field, src_field, target_field)`	Auto-fetch value: when `link_field` changes, copy `src_field` into `target_field`.

Function	Description / Example
Form Methods: `frm.reload_doc()`, `frm.save()`, `frm.dirty()`, `frm.is_new()`, `frm.add_custom_button()`, etc. (See [Form API] [54] [55].)	

Desk UI Scripting (Form/List/Report)

- **Form Scripts:** Use `frappe.ui.form.on('Doctype', {...})` or Client Script doctype (UI) for form-level logic (with events above).
- **List Scripts:** To customize List View, create `<doctype>_list.js` or use Client Script (Type=List). In code:

```
frappe.listview_settings['Note'] = {
  add_fields: ['title','public'],
  filters: [['public','=',1]],
  onload(listview) { ... },
  get_indicator(doc) { ... }
}
```

  (See example in docs [56].)
- **Report Scripts:** For custom Query/Script reports, you can include client-side scripts in a `{report}.js` file to set up filters or buttons. (In script reports, filters defined in doc appear as `filters` in context.)
- **Other scripts:** Page scripts (`frappe.pages`), Dashboard/chart configs, and **Server Scripts** (new in v14) allow injecting Python logic at various hooks.

Customization Options

Feature	Purpose / Effect
Custom Field	Adds a site-specific field to a DocType (stored in `tabCustom Field`). Persists after upgrades [57].
Property Setter	Override a property of a DocType or field (e.g. label, default, mandatory) without changing code. Stored in `tabProperty Setter` [57].
Customize Form	GUI tool to add Custom Fields and Property Setters in one place; creates above records behind the scenes [58].
Client Script	(formerly Custom Script) Custom JS for a DocType or global context (Form/List/Report). Executes in browser. Introduced as “Client Script” in v13 [59].

Feature	Purpose / Effect
Server Script	Custom Python logic triggered by DocType events or APIs (available in UI since v13). **Note:** In v15, Server Scripts are **disabled by default** (enable via `bench set-config -g server_script_enabled 1`) [60].
Property Setter	(see above)

Note: These customizations are stored in the database and override core metadata at runtime (see Customize Form docs) [58].

Jinja Template Context (Print/Email)

In Jinja-based templates (Print Formats, Email Templates), the following are available:

- `doc` – The current document object. Use `{{ doc.fieldname }}` to show field values.
- `frappe` – Frappe module with helper functions (e.g. `frappe.format`, `frappe.get_doc`, `frappe.db.get_value`) [61] [62]. For example:

```
{{ frappe.get_doc('Task', doc.name).title }}   {# fetch another doc #}
{{ frappe.db.get_value('User', doc.owner, 'full_name') }}
{{ frappe.format_date(doc.date_field) }}
```

- `filters` – A dict of current filter values (mainly in report print contexts). E.g. in custom report print templates you can use `filters.some_filter_name` [63].
- **Other:** Jinja built-in filters (e.g. `| date:"%d-%m-%Y"`) and translations (`_()` macro) work. In print formats you can also use **custom Jinja methods/filters** defined via hooks [28].

(For example, Frappe provides `get_serial_or_batch_nos(bundle_id)` *in v15 for printing serials from Serial/Batch Bundle* [64] *.)*

Version Differences (v14 vs v15)

- **Type Annotations (v15):** Query reports and controllers can now use type hints (e.g. annotate `execute(filters: dict)`), introduced in Frappe v15 [12].
- **Hooks Resolution:** The `override_whitelisted_methods` hook now uses **last-installed** override instead of first [24]. E.g. the last app defining an override wins in v15.
- **Server Scripts:** In Frappe v15, Server Scripts are **disabled by default** for security [60]. Enable with `bench set-config -g server_script_enabled 1`.
- **Custom Scripts Fixtures:** The old single-script fixture (`fixtures/custom_scripts`) is removed in v15 [29]. Use multiple client scripts (UI) or fixtures normally.
- **Bench Defaults:** `bench use` now replaces `currentsite.txt` for default site (v15 removed `currentsite.txt`) [65].
- **get_installed_apps:** In v15, `frappe.get_installed_apps()` no longer accepts sorting args (the hook order changed) [66].

- **SocketIO:** v15 uses site-based namespaces for Socket.IO (frontend clients now auto-connect with `io(url + '/' + frappe.local.site)`) [67].
- `search_link` **/** `search_widget` **output:** In v15 their AJAX responses use `message` instead of custom keys (update client code accordingly) [68].
- **Deprecated:** `setup.py` is dropped (use `pyproject.toml`) [69]; old window globals (e.g. `user`, `get_today`) are removed [70].
- **Workflow/DocStatus:** (no change) DocStatus values remain 0/1/2 [15]; `workflow_state` still stores the custom workflow state in both versions.
- **Print APIs:** GraphQL support (introduced in v14) is stable; new Jinja helpers like `get_serial_or_batch_nos()` were added for v15 (for Serial/Batch Bundle) [64].

Each section above includes v14/v15 notes where relevant. (For complete migration notes, see Frappe v15 migration docs [60] [24].)

Sources: Official Frappe/ERPNext docs and developer guides [1] [22] [44] [51] [56] [57] [63] [60] [24].

---

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] Controllers
https://docs.frappe.io/framework/user/en/basics/doctypes/controllers

[15] Workflows
https://docs.frappe.io/erpnext/user/manual/en/workflows

[16] [17] [18] [19] Document API
https://docs.frappe.io/framework/user/en/api/document

[20] [21] Database API
https://docs.frappe.io/framework/user/en/api/database

[22] [23] [25] [26] [27] [28] Hooks
https://docs.frappe.io/framework/user/en/python-api/hooks

[24] [29] [60] [65] [66] [67] [68] [69] [70] Migrating to version 15 · frappe/frappe Wiki · GitHub
https://github.com/frappe/frappe/wiki/Migrating-to-version-15

[30] Background Jobs
https://docs.frappe.io/framework/user/en/api/background_jobs

[31] [32] [33] [34] REST API
https://docs.frappe.io/framework/user/en/api/rest

[35] [36] [38] [40] [41] [42] [43] Bench Commands Cheatsheet
https://docs.frappe.io/framework/user/en/bench/resources/bench-commands-cheatsheet

[37] [39] Bench Commands
https://docs.frappe.io/framework/user/en/bench/bench-commands

[44] [45] [46] [47] [48] [49] [52] [53] [54] [55] Form Scripts
https://docs.frappe.io/framework/user/en/api/form

[50] [51] Frappe Ajax Call
https://docs.frappe.io/framework/user/en/guides/basics/frappe_ajax_call

[56] List
https://docs.frappe.io/framework/user/en/api/list

[57] [58] Customizing DocTypes
https://docs.frappe.io/framework/user/en/basics/doctypes/customize

[59] Client Script
https://docs.frappe.io/framework/user/en/desk/scripting/client-script

[61] [62] Jinja API
https://docs.frappe.io/framework/user/en/api/jinja

[63] Printing
https://docs.frappe.io/framework/user/en/desk/printing

[64] Migration Guide to ERPNext version 15 · frappe/erpnext Wiki · GitHub
https://github.com/frappe/erpnext/wiki/Migration-Guide-to-ERPNext-version-15