**Frappe Document Events: Execution Order, Uses & Statuses**

In **Frappe Framework**, different **document events** trigger at various stages of a document’s lifecycle. Here’s a breakdown of these events in the order they execute, along with their uses and the status of the document during each event.

**1. Events Before Document Creation (Naming & Validation)**

These events trigger **before the document is saved to the database**.

**1.1 before\_naming**

**When it Triggers?**

* Before the document’s name is set.

**Use Case:**

* To set a custom naming convention before saving the document.

**Status of Document:**

* The document exists **only in memory**; it is not yet in the database.

**1.2 autoname *(Optional)***

**When it Triggers?**

* If defined, this method will execute to customize the name property **before saving the document**.

**Use Case:**

* To set custom unique IDs for documents, such as STU-0001 instead of a system-generated number.

**Status of Document:**

* Still **not saved** to the database.

**2. Events During Document Validation & Insertion**

**2.1 validate**

**When it Triggers?**

* Before the document is **saved for the first time or updated**.

**Use Case:**

* To check if all required fields are filled.
* To prevent duplicate entries.
* To apply custom validation rules.

**Status of Document:**

* Exists only in memory, **not yet saved**.

**2.2 before\_insert**

**When it Triggers?**

* Right **before inserting** a new document into the database.

**Use Case:**

* To modify fields before they are saved.
* To set default values.

**Status of Document:**

* Still in memory, **not inserted yet**.

**2.3 after\_insert**

**When it Triggers?**

* Right **after the document is inserted into the database**.

**Use Case:**

* To trigger related actions like notifications.
* To create related documents.

**Status of Document:**

* **Saved in the database** for the first time.

**3. Events Before & After Saving**

**3.1 before\_save**

**When it Triggers?**

* Before the document is saved (new or existing).

**Use Case:**

* To make last-minute modifications before saving.

**Status of Document:**

* Not saved yet.

**3.2 on\_change**

**When it Triggers?**

* If any field values in the document **have changed**.

**Use Case:**

* To track changes and log modifications.

**Status of Document:**

* The document is being updated.

**3.3 on\_update**

**When it Triggers?**

* After any update to an **existing document**.

**Use Case:**

* To execute actions after data is changed.

**Status of Document:**

* Already updated.

**4. Events Related to Submission Workflow**

**4.1 before\_submit**

**When it Triggers?**

* Before a document is **submitted**.

**Use Case:**

* To validate business rules before approval.

**Status of Document:**

* Not submitted yet.

**4.2 on\_submit**

**When it Triggers?**

* When a document is **submitted**.

**Use Case:**

* To trigger notifications or create related records.

**Status of Document:**

* **Submitted**.

**4.3 before\_update\_after\_submit**

**When it Triggers?**

* Before updating a **submitted** document.

**Use Case:**

* To prevent changes after submission if not allowed.

**Status of Document:**

* **Submitted & being updated**.

**4.4 on\_update\_after\_submit**

**When it Triggers?**

* After updating a **submitted** document.

**Use Case:**

* To trigger actions after an update.

**Status of Document:**

* **Submitted & updated**.

**5. Events Related to Cancellation**

**5.1 before\_cancel**

**When it Triggers?**

* Before a document is **cancelled**.

**Use Case:**

* To prevent cancellation if dependencies exist.

**Status of Document:**

* **Submitted & about to be cancelled**.

**5.2 on\_cancel**

**When it Triggers?**

* When a document is **cancelled**.

**Use Case:**

* To notify users or reverse transactions.

**Status of Document:**

* **Cancelled**.

**6. Events Related to Renaming**

**6.1 before\_rename**

**When it Triggers?**

* Before renaming a document.

**Use Case:**

* To validate if renaming is allowed.

**Status of Document:**

* **Existing, but not yet renamed**.

**6.2 after\_rename**

**When it Triggers?**

* After renaming a document.

**Use Case:**

* To update references in related documents.

**Status of Document:**

* **Renamed**.

**7. Events Related to Deletion**

**7.1 on\_trash**

**When it Triggers?**

* Before a document is deleted.

**Use Case:**

* To prevent deletion if dependencies exist.

**Status of Document:**

* **Exists but about to be deleted**.

**7.2 after\_delete**

**When it Triggers?**

* After a document is deleted.

**Use Case:**

* To clean up related records.

**Status of Document:**

* **Deleted**.

**8. Summary of Event Execution Order**

* **For New Document:** before\_naming → autoname → validate → before\_insert → after\_insert → before\_save → on\_change → on\_update
* **For Updating Document:** validate → before\_save → on\_change → on\_update
* **For Submission:** before\_submit → on\_submit
* **For Cancelling:** before\_cancel → on\_cancel
* **For Deletion:** on\_trash → after\_delete

This guide helps in choosing the right event based on your use case! 🚀

**🔹 Events Execution Order (From Creation to Deletion)**

Below is the order in which these events are triggered:

**🚀 1. Before Document Creation**

| **Event** | **When is it Triggered?** | **Purpose** | **Document Status** |
| --- | --- | --- | --- |
| before\_naming | Before the document name is generated. | Modify the document’s name before it is set. | Not saved yet |
| autoname | Custom name generation (only if defined). | Define a custom naming scheme. | Not saved yet |
| before\_insert | Before inserting into the database. | Modify default values before saving. | Not saved yet |
| validate | Before saving, to validate data. | Ensure required fields, prevent invalid data. | Not saved yet |
| after\_insert | After insertion but before saving. | Perform actions after the document is created. | Saved but not submitted |

**✍️ 2. During Document Updates**

| **Event** | **When is it Triggered?** | **Purpose** | **Document Status** |
| --- | --- | --- | --- |
| before\_save | Before the document is saved (new or existing). | Modify or validate fields before saving. | Draft (or existing) |
| on\_change | When any field is modified. | Track field changes dynamically. | Draft (or existing) |
| on\_update | When an **existing** document is updated. | Run actions after updates (e.g., send notifications). | Draft (or existing) |

**✅ 3. During Submission**

| **Event** | **When is it Triggered?** | **Purpose** | **Document Status** |
| --- | --- | --- | --- |
| before\_submit | Before a document is submitted. | Prevent incorrect submissions. | Draft |
| on\_submit | When a document is submitted. | Execute logic after submission (e.g., create linked records). | Submitted (Finalized) |

**🔄 4. After Submission (Editing & Updates)**

| **Event** | **When is it Triggered?** | **Purpose** | **Document Status** |
| --- | --- | --- | --- |
| before\_update\_after\_submit | Before updating a **submitted** document. | Prevent invalid updates after submission. | Submitted |
| on\_update\_after\_submit | After updating a **submitted** document. | Track updates in a submitted state. | Submitted |

**❌ 5. Cancelling a Document**

| **Event** | **When is it Triggered?** | **Purpose** | **Document Status** |
| --- | --- | --- | --- |
| before\_cancel | Before a document is cancelled. | Prevent unwanted cancellations. | Submitted |
| on\_cancel | When a document is cancelled. | Run actions after cancellation (e.g., reverse transactions). | Cancelled |

**🔄 6. Renaming a Document**

| **Event** | **When is it Triggered?** | **Purpose** | **Document Status** |
| --- | --- | --- | --- |
| before\_rename | Before renaming a document. | Validate if renaming is allowed. | Any |
| after\_rename | After renaming a document. | Handle post-renaming actions. | Any |

**🗑️ 7. Deleting a Document**

| **Event** | **When is it Triggered?** | **Purpose** | **Document Status** |
| --- | --- | --- | --- |
| on\_trash | Before deleting a document. | Stop deletion if necessary. | Any |
| after\_delete | After a document is deleted. | Cleanup related records. | Deleted |

## ****🔹**** doc.insert()

✅ **Creates a new document in the database.**

* It **assigns a new name (ID)** to the document.
* It **must be used for new documents** that are not yet saved.

## ****🔹**** doc.save()

✅ **Saves changes to an existing document.**

* Can be used for both **new and existing documents**.
* If the document is new, it **calls insert() internally**.
* If the document exists, it **updates** it.
* If the document **already exists**, it **updates** it.

### ****🔹 Simple Difference****

| **Action** | **insert()** | **save()** |
| --- | --- | --- |
| Creates a new record? | ✅ Yes | ✅ Yes (if new) |
| Updates an existing record? | ❌ No | ✅ Yes |
| Assigns a new ID? | ✅ Yes | ❌ No |

 **insert() → Creates a new record** (Always new)

 **save() → Saves changes** (New or existing)

## ****🔹 When to Use**** @frappe.whitelist()****?****

✔ **If you want JavaScript (frappe.call) to access a Python function**  
✔ **If you want an API endpoint for external apps**  
✔ **If you want to expose a function publicly (allow\_guest=True)**

### ****🔹 When to Use?****

✔ **Use @frappe.whitelist()** when you want a **Python function to be accessible from JavaScript or API calls**.

💡 Key Differences

Feature frappe.db.delete() doc.delete()

Bypasses validation? ✅ Yes ❌ No

Deletes child tables? ❌ No ✅ Yes

Triggers before\_delete and after\_delete events? ❌ No ✅ Yes

Requires commit? ✅ Yes ✅ Yes

🚀 Which one should you use?

If you need a safe, proper deletion → Use doc.delete() ✅

If you are performing bulk deletions and don't need validation → Use frappe.db.delete() ⚡

Let me know if you need help implementing this! 🔥

**1. Frappe Framework**

* Frappe is a full-stack web application framework built with Python and JavaScript.
* It follows an MVC (Model-View-Controller) architecture.
* It provides the foundation for **ERPNext**, enabling developers to build custom applications.
* It includes built-in features like Doctypes, REST API, authentication, permissions, and workflow automation.

**2. Frappe Form (FRM) / Doctype Form**

* In **Frappe**, a **Form (FRM)** refers to a user interface to interact with a Doctype.
* Every Doctype in Frappe has a corresponding form for data entry and management.
* Forms are dynamically generated based on Doctype fields, allowing easy CRUD (Create, Read, Update, Delete) operations.
* You can customize forms using **Custom Scripts**, **Client Scripts**, and **Server Scripts**.

**Key Differences**

| **Feature** | **Frappe Framework** | **Frappe Form (FRM)** |
| --- | --- | --- |
| Purpose | Full-stack framework to build apps | UI for interacting with a Doctype |
| Role | Backend + Frontend + API | Frontend component |
| Scope | Entire application logic and workflow | Only a part of the application (UI for a Doctype) |
| Customization | Requires Python & JavaScript | Can be modified via Frappe UI, Custom Scripts |