# The Complete Guide to Using Build Agent Effectively

Everything you need to craft precise prompts, understand the development workflow, and build production-ready ServiceNow applications — all in one reference.

## 1 What Is the Build Agent?

The Build Agent is an AI-powered ServiceNow development assistant that creates and modifies **application metadata** using the **Now SDK Fluent DSL**. It can:

- **Create new applications** from scratch — tables, business rules, UI pages, and more
- **Edit existing metadata** — fetch records from your instance, modify them, and deploy changes
- **Build & install** — compile, validate, and push changes directly to your ServiceNow instance
- **Search your instance** — find tables, scripts, records, and metadata across all scopes

> ℹ️ Think of it as a developer colleague who knows the ServiceNow platform deeply and can write Fluent DSL code for you — but needs **clear, specific instructions** to produce the best results.

## 2 What It Can Build — 30+ Metadata Types

🗃️ **Tables & Columns**

Data models, fields, relationships, choices, indexes, auto-numbering

⚙️ **Business Rules**

Server-side logic on insert/update/delete/query — before, after, async, display

🖥️ **Client Scripts**

Browser-side form behavior — onLoad, onChange, onSubmit, onCellEdit

📜 **Script Includes**

Reusable server-side classes, GlideAjax client-callable scripts

🌐 **UI Pages**

Custom React-based web pages with ServiceNow components

🔌 **Scripted REST APIs**

Custom HTTP endpoints with versioning and resource definitions

🔄 **Flows (WFA)**

Workflow automation — triggers, actions, conditions, subflows

🔒 **Security**

ACLs, Roles, Security Attributes, Data Filters

📧 **Email Notifications**

Automated emails on record operations or custom events

⏰ **Scheduled Scripts**

Time-based background jobs and maintenance tasks

🛒 **Service Catalog**

Catalog items, record producers, variables, variable sets

🧪 **ATF Tests**

Automated Test Framework — server, form, REST, catalog test steps

📋 **Platform Views**

UI Policies, UI Actions, Lists, Views, Formatters

🏠 **Navigation**

Application Menus, Modules, Navigator entries

🌀 **Service Portal**

Portals, pages, widgets, themes, menus, Angular providers

🔧 **And More...**

Properties, Events, Data Sources, LDAP, Cross-Scope Privileges, Workspaces

## 3 The Development Workflow

Every interaction follows this structured pipeline. Understanding it helps you write better prompts:

### Creating New Metadata

STEP 1

**Plan**

Analyze requirements, pick metadata types

STEP 2

**Generate**

Write Fluent .now.ts files

STEP 3

**Validate**

Run diagnostics on files

STEP 4

**Build**

Compile the full app

STEP 5

**Install**

Deploy to instance

### Editing Existing Metadata

STEP 1

**Discover**

Search & find the record

STEP 2

**Fetch**

Add to working set

STEP 3

**Edit**

Modify the Fluent code

STEP 4

**Build**

Compile & validate

STEP 5

**Install**

Deploy changes

💡 You don't need to mention these steps in your prompt — the agent follows them automatically. But understanding them helps you provide the right information at the right time.

## 4 The 10 Golden Rules of Effective Prompts

| # | Rule | Why |
|---|---|---|
| 1 | **Be specific about the metadata type**<br>*"Create a **before business rule**" not "add some logic"* | Eliminates ambiguity about what to generate |
| 2 | **Name your tables with scope prefixes**<br>`x_snc_myapp_project` *not* `project` | Required for builds to succeed |
| 3 | **Specify timing / type / trigger**<br>*"before on insert", "onChange on priority field"* | The most critical property for rules and scripts |
| 4 | **Describe the exact behavior**<br>*"Set* `approval` *to 'requested'" not "handle approvals"* | Prevents guesswork about field names and values |
| 5 | **Include conditions and edge cases**<br>*"Only when amount > 5000 AND user lacks finance_admin role"* | Produces correct conditional logic upfront |
| 6 | **State field types explicitly**<br>*"* `budget` *— DecimalColumn" not "a field for money"* | Wrong type choices can't be easily fixed after data exists |
| 7 | **Mention related artifacts**<br>*"Also create the companion Script Include for GlideAjax"* | Many patterns require multiple cooperating metadata records |
| 8 | **Provide error messages and user feedback**<br>*Show message: "Expenses over $5K require approval"* | Creates a polished user experience |
| 9 | **One concern per prompt (usually)**<br>*Create the table first, then add business rules in the next prompt* | Reduces complexity and error rate — but complex prompts work too |

| # | Rule | Why |
|---|---|---|
| 10 | **Trust the agent's workflow**<br>*You don't need to say "create a .now.ts file" or "run the build"* | The agent handles file creation, building, and deployment automatically |

## 5 Anatomy of a Great Prompt — Universal Template

Regardless of metadata type, every great prompt follows this structure:

```
[ACTION] a [METADATA TYPE] on the [TABLE] table
  that [TRIGGER/TIMING]
  when [CONDITION]
  does [BEHAVIOR]
  with [CONFIGURATION OPTIONS]
```

### Filled Example

> **Create** a **before business rule** on the `x_snc_myapp_expense` table
> that runs on **insert and update**
> when `amount` is greater than 5000
> sets `approval` to "requested" and `approver` to the user's manager
> with order **200**, message *"Expenses over $5K require approval"*, and put logic in a separate module file.

### The Information Spectrum

Prompts exist on a spectrum from minimal to maximal detail. More detail = better results:

| Level | Example | Result Quality |
|---|---|---|
| 🔴 **Minimal** | *"Add validation to incidents"* | Agent must guess — high chance of back-and-forth |
| 🟡 **Moderate** | *"Create an onSubmit client script on incident that validates description is not empty"* | Works but may miss edge cases or UI details |
| 🟢 **Detailed** | *"Create an onSubmit client script on incident that validates description has ≥10 chars. Show field-level error via g_form.showFieldMsg. Block submission. Run on all UI types."* | First-pass production quality 🎯 |

## 6 Universal Pitfalls — Mistakes That Apply to All Types

| ❌ Don't | ✅ Do | Why |
|---|---|---|
| Use vague verbs like *"handle"*, *"manage"*, *"process"* | Use precise verbs: *"set"*, *"validate"*, *"hide"*, *"abort"*, *"query"* | Vague verbs could mean 10 different things |
| Forget the scope prefix on table names | Always use `x_snc_myapp_tablename` | Builds fail without proper scope prefixes |
| Ask for unsupported column types | Check the supported column type list first | Types like `CurrencyColumn` don't exist — use `GenericColumn` |
| Assume the agent knows your instance | Name specific tables, fields, and values | The agent searches your instance but needs starting points |
| Request multiple unrelated changes in one prompt | Group related changes; split unrelated ones | Related changes benefit from shared context; unrelated ones cause confusion |
| Skip error messages and user feedback | Always specify what the user should see | Silent failures are bad UX |
| Say *"create a .now.ts file"* or *"run the build"* | Describe **what** you want, not **how** to implement it | The agent handles file creation and builds automatically |

## 7 Creating New Metadata — Prompt Strategies

### Strategy A: Bottom-Up (Recommended for Complex Apps)

Build foundational layers first, then add behavior:

1. **Prompt 1:** Create the tables and columns (data model)
2. **Prompt 2:** Add business rules (server-side logic)
3. **Prompt 3:** Add client scripts (form behavior)
4. **Prompt 4:** Add security (ACLs, roles)
5. **Prompt 5:** Add navigation (menus, modules)

### Strategy B: Feature-Complete (For Simple Features)

Describe the entire feature in one prompt:

> Create an Equipment Request feature: a table extending task with equipment_type (dropdown), quantity (integer), justification (multi-line text), and requested_for (reference to sys_user). Add a before business rule that sets approval to "requested" when quantity > 5. Add an onChange client script that shows justification as mandatory when equipment_type is "Other". Add an admin role and read ACL.

### Strategy C: Application Kickstart

Let the agent create the entire app from a high-level description:

> Create a new application called "Vendor Management" that tracks vendor companies, their contacts, and contracts. Include tables for vendors, contacts, and contracts with appropriate relationships. Add auto-numbering for contracts (prefix VND). Enable audit tracking on all tables.

> 💡 **Strategy A** gives the most control. **Strategy C** is fastest for getting started. Use A for mission-critical apps and C for prototypes.

## 8 Editing Existing Metadata — Prompt Strategies

### Identifying What to Edit

The agent needs to find the record before editing it. Help it by providing:

| What You Know | How to Say It |
|---|---|
| Exact name | *"Edit the business rule named 'Validate Priority' on the incident table"* |
| Table + type | *"Edit the onChange client script on the incident table that watches the category field"* |
| App name | *"Show me all business rules in my Vendor Management app"* |
| General area | *"I want to edit a script include in my app"* → Agent will list options |

### Describing the Changes

> **Update** the existing business rule named `"Set Default Priority"` on the `incident` table to:
> • Also check that `description` is not empty (currently it only checks `short_description`)
> • Change the abort message from "Missing fields" to "Short description and description are both required"
> • Add the `itil` role condition so it only runs for ITIL users

> ⚠️ When editing, **describe what should change** — not the entire record. The agent fetches the current state and applies your modifications on top of it.

## 9 Quick Ref: Tables & Columns

| Dimension | What to Specify | Example |
|---|---|---|
| **Name** | Full scoped name | `x_snc_myapp_project` |
| **Label** | Human-readable | *"Project Request"* |
| **Extends** | Base table (if any) | `task`, `cmdb_ci` |
| **Each Column** | Type + label + constraints | *`budget`* *— Decimal, mandatory* |
| **Display** | Which field shows in references | `name` |
| **Auto-number** | Prefix, start, digits | *PRJ, 1000, 7 digits* |
| **Access** | Public or private + web services | *Public, enable web services* |

💡 **Key column types:** `StringColumn`, `IntegerColumn`, `DecimalColumn`, `BooleanColumn`, `ReferenceColumn`, `DateColumn`, `DateTimeColumn`, `MultiLineTextColumn`, `ChoiceColumn`, `EmailColumn`, `UrlColumn`, `JsonColumn`.
For dropdowns, use `StringColumn` + `choices` + `dropdown`.

## 10 Quick Ref: Business Rules

| Dimension | What to Specify | Options |
|---|---|---|
| **Table** | Which table | System name: `incident`, `x_snc_myapp_task` |
| **Timing** | When it runs | `before` · `after` · `async` · `display` |
| **Action** | On which operations | `insert` · `update` · `delete` · `query` |
| **Condition** | When to fire | Filter condition or script condition |
| **Logic** | What it does | Set fields, validate, abort, update related records |
| **Extras** | Optional details | Order, role, message, module file, description |

⚠️ **Before** = can modify current record & abort. **After** = record already saved, update related records only. **Async** = non-blocking background. **Display** = calculated form values.

## 11 Quick Ref: Client Scripts

| Dimension | What to Specify | Options |
|---|---|---|
| **Table** | Which form | System table name |
| **Type** | When it fires | `onLoad` · `onChange` · `onSubmit` · `onCellEdit` |
| **Field** | Trigger field | Required for onChange & onCellEdit |
| **Behavior** | What it does | Use `g_form` methods: setValue, setVisible, setMandatory, etc. |
| **UI Target** | Which interface | `desktop` · `mobile_or_service_portal` · `all` |
| **Server Data** | Need to call server? | Mention **GlideAjax + Script Include** |

🚫 **Never use GlideRecord in a client script.** For server lookups, use GlideAjax + a client-callable Script Include. Always mention both.

## 12 Quick Ref: All Other Metadata Types

| Type | Key Dimensions to Specify |
|---|---|
| **Script Includes** | Name, class or function pattern, client-callable (yes/no), methods, which APIs used |
| **UI Pages** | Page name, what it displays (list/form/dashboard), data sources (Table API), interactions |
| **REST APIs** | API name, version, resources, HTTP methods, request/response format, authentication |
| **Flows** | Trigger type (record/scheduled/app), conditions, actions, branching logic |
| **ACLs** | Table, operation (read/write/create/delete), role, condition, script |
| **Roles** | Role name, description, contains roles (inheritance) |
| **Email Notifications** | Table, trigger event, recipients, subject, body template, conditions |
| **Scheduled Scripts** | Name, schedule (cron), script logic, active/inactive |
| **Catalog Items** | Name, category, variables, workflow/flow, UI policies, client scripts |
| **UI Policies** | Table, conditions, field actions (visible/mandatory/read-only), reverse |
| **UI Actions** | Table, action name, form/list/both, conditions, script, client/server |
| **Properties** | Name, type, value, description, category |
| **Menus & Modules** | Menu name, module type (list/url/separator), table, filter, order |
| **ATF Tests** | Test name, steps in order, assertions, test data setup |
| **Service Portal** | Widget name, template (HTML), client script, server script, CSS, dependencies |

## 13 Multi-Artifact Prompts — Building Complete Features

You can request multiple related metadata in a single prompt. The agent will create a plan and build them in order:

### Pattern: Table + Behavior + Security + Navigation

> Build a complete **Equipment Checkout** feature:
>
> **Table:** `x_snc_myapp_checkout` extending `task` with fields: `equipment` (Reference to cmdb_ci, mandatory), `checkout_date` (DateTime), `return_date` (DateTime), `condition_notes` (MultiLineText 2000). Auto-number: prefix CHK, start 1000, 7 digits.
>
> **Business Rule:** Before insert — set `checkout_date` to now and `state` to "Open".
>
> **Client Script:** onChange on `equipment` — use GlideAjax to look up the equipment's location and auto-fill the `location` field. Create the companion Script Include.
>
> **Security:** Create a `checkout_user` role. Add read ACL for all authenticated users, write ACL for checkout_user role only.
>
> **Navigation:** Add an "Equipment Checkout" module under the app menu linking to the checkout table list.

> 💡 Multi-artifact prompts work best when the artifacts are **related** (same table/feature). For unrelated changes, use separate prompts.

## 14 Handling Build Errors

Sometimes builds fail. Here's how to respond effectively:

### If the Agent Reports a Build Error

- **Don't panic** — build errors are normal during development
- **Read the error message** — the agent will show you exactly what went wrong
- **The agent will attempt to fix it** — usually it can resolve errors automatically
- **If it can't fix it**, provide more context or simplify your request

### Common Error Causes & How to Help

| Error Type | Common Cause | How to Help |
|---|---|---|
| Missing import | Column type used but not imported | Agent fixes this automatically |
| Name mismatch | Export variable ≠ table name | Agent fixes this automatically |
| Invalid reference | Referenced table doesn't exist | Confirm the exact table name on your instance |
| Scope prefix missing | Table name lacks scope prefix | Always use full scoped names in prompts |
| Unsupported column type | Type like `CurrencyColumn` doesn't exist | Use `GenericColumn` with `internal_type` |

## 15 Iterating Effectively

### Follow-Up Prompt Patterns

| Scenario | Good Follow-Up Prompt |
|---|---|
| Add to what was just created | *"Now add a before business rule to the table we just created that sets state to 'Open' on insert."* |
| Modify what was just created | *"Change the* `quantity` *field to be mandatory and add a max value of 100."* |
| Fix something wrong | *"The dropdown should be* `dropdown_without_none` *instead of* `dropdown_with_none`*. Also change the default to 'Laptop'."* |
| Add related behavior | *"Add an onSubmit client script to the same table that validates quantity is between 1 and 100."* |
| Request different approach | *"Instead of a business rule, use a UI Policy to make those fields mandatory when state is Resolved."* |

> 💡 The agent **remembers context within a conversation**. You can say "the table we just created" or "that business rule" without re-specifying names.

## 16 Master Prompt Checklist

### Before Every Prompt, Ask Yourself:

- ☐ **What metadata type?** — Table, Business Rule, Client Script, Flow, ACL, etc.
- ☐ **Which table?** — Full scoped name (or "the table we just created")
- ☐ **What triggers it?** — Timing, event type, action, field
- ☐ **Under what conditions?** — When should it fire/apply?
- ☐ **What's the behavior?** — Exact fields, values, logic
- ☐ **What should the user see?** — Messages, visibility changes, validation errors
- ☐ **Any related artifacts needed?** — Script Include for GlideAjax, Role for ACL, etc.

### Quality Boosters (Optional but Helpful)

- ☐ **Execution order** — For business rules that interact with others
- ☐ **Role restrictions** — Who should this affect?
- ☐ **UI target** — Desktop only? Mobile? All?
- ☐ **Description** — For future maintainers
- ☐ **Edge cases** — What happens with empty values? Null references?
- ☐ **Performance** — Should it be async? Does it need an index?

## 17 Gold-Standard Examples

### Example 1: Creating a Complete Table

> Create a table called `x_snc_myapp_vendor` with label **"Vendor"**. Fields:
> • `name` — String, 100 chars, mandatory, display column
> • `category` — String dropdown (dropdown_with_none): "Technology", "Consulting", "Supplies". Default: none.
> • `primary_contact` — Reference to `sys_user`, cascade: clear
> • `contract_value` — Decimal, label "Contract Value (USD)"
> • `active` — Boolean, default true
> • `website` — URL column
> • `notes` — Multi-line text, 4000 chars
>
> Auto-number: prefix VND, start 1000, 7 digits. Enable audit and web service access. Set as extensible, public access.

### Example 2: Editing a Business Rule

> Update the existing before business rule named `"Validate Assignment"` on the `incident` table to also check that `assignment_group` is not empty when `priority` is "1 - Critical". If validation fails, abort with message: *"Critical incidents must have an assignment group."* Keep all existing logic unchanged.

### Example 3: Multi-Artifact Feature

> Build an expense approval workflow for the `x_snc_myapp_expense` table:
>
> 1. **Before business rule** on insert/update: When `amount` > 5000 and user lacks `finance_admin` role, set `approval` to "requested" and `approver` to user's manager. Show message: "High-value expenses require manager approval." Order 200. Put logic in a module file.
>
> 2. **onChange client script** on `amount`: When amount changes to > 5000, show an info message via `g_form.showFieldMsg`: "This amount requires manager approval." When ≤ 5000, hide the message. Include isLoading guard. Run on all UI types.

3. **onSubmit client script**: Validate that `description` has at least 20 chars and `expense_date` is not in the future. Show field-level errors. Block submission on failure.

4. **Email notification**: When `approval` changes to "requested", send email to the `approver` with subject "Expense Approval Required: ${number}" and body including the amount, description, and a link to the record.

💡 These gold-standard prompts cover every relevant dimension for their metadata types. Notice how each specifies **what** (the metadata), **when** (triggers/timing), **where** (table/fields), **why** (conditions), and **how** (exact behavior + messages). 🎯