Access Policies (Attribute-Based Access Control)

Navigation: Admin → People & Permissions → Access Policies **Also accessible via:** Menu → User Management → Access Policies

---

What Are Access Policies?

Access Policies implement **Attribute-Based Access Control (ABAC)** — a layer of fine-grained permission logic that works alongside RBAC (Roles & Permissions) and Divisions.

Where RBAC answers "*does this role have this permission?*", ABAC answers "*given the attributes of this user, this resource, and this environment — should access be allowed or denied?*"

> ℹ ABAC does **not** replace Roles or Divisions. It augments them. Authorization requires **both** ABAC policy evaluation and standard permission checks to pass.

Enable ABAC First

Before any access policies take effect, you must enable the feature at the org level:

Admin → Organization Settings → [toggle] Enable Attribute-Based Access Control

Policies can be created and saved before enabling, but they will not be enforced until the org setting is on.

Core Concepts

The Three Attribute Types

Attribute Type	What It Describes
Subject	The user or entity requesting access (their roles, division, group membership, etc.)
Resource / Object	The thing being accessed (a user profile, a recording, a gamification scorecard, etc.)
Environment	Contextual factors (IP address, time of day, etc.)

Policy Structure

Each policy contains:

Element	Description
Name	Identifier for the policy
Description	Purpose and context
Target	The API calls this policy applies to — written as `domain:entity:action` (e.g., `authorization:grant:add`)
Subject	Who the policy applies to — user, group, client, or all
Effect	`ALLOW` or `DENY`
Conditions	Boolean logic (Any = OR, All = AND) using `TypedAttribute` key-value pairs

Default-Deny Logic

> ⚠ **DENY overrides ALLOW.** If a subject matches both an ALLOW and a DENY policy, access is denied.

Access is granted only when **all three** are true:

1. Subject satisfies an ALLOW policy's conditions
2. Subject has the required traditional permission
3. Subject does not match any DENY policy

The Three Built-In Policy Types

Genesys Cloud ships three pre-built policy templates covering the most common ABAC use cases:

1. Cannot Grant New Roles

Purpose: Prevents non-admin users from granting roles they do not themselves hold.

Use case: Stops a supervisor from escalating privileges by assigning a role they don't have to another user — even if they technically have the `authorization:grant:add` permission.

How it works:

- Checks if the subject is an admin
- Checks if the subject already has the role being granted
- If neither condition is true → DENY

---

2. Cannot Update Certain User Profile Fields

Purpose: Prevents specified profile fields from being modified by regular users — while still allowing supervisors or admins to edit them.

Use case: Lock fields like employee ID, HR data, or contact info so agents cannot self-edit them, but managers can still update them.

How it works:

- Uses `profile.fields` condition — case-insensitive check against a list of restricted field names
- If the field being modified is in the restricted list AND the subject does not have a manager-level role → DENY

Restricted field values (configured in policy JSON as `preset attributes`):

Category	Example Field Values
Name	`name`, `name.firstName`, `name.lastName`
Contact Info	`contactInfo`, `contactInfo.phone_work`, `contactInfo.phone_work_2`
HR Fields	`hr` (entire section)

Category	Example Field Values
Images	`images`, `images.profile`
Biography	`biography`
Location	`location`
Relationships	`relationships`

> ℹ Using a section name (e.g., `contactInfo`) restricts the entire section — equivalent to `contactInfo.*`. Add specific sub-fields only when partial restriction is needed.

Full list: **Admin → People & Permissions → Access Policies → Templates → Cannot Update Certain User Profile Fields** — or see the Genesys Resource Center Restricted fields value list.

---

3. Access Control for Gamification Scorecard and Insights

Purpose: Limits supervisors to viewing gamification data only for agents within their reporting hierarchy, while admins retain full access.

How it works (hierarchy tiers):

Role	Access Scope
Supervisor	Direct reports only
Manager of managers	Up to 3 levels deep in hierarchy
Admin (designated role)	Full access to all gamification data

Use case: Prevents supervisors from browsing gamification performance data for agents outside their team.

Creating an Access Policy

Option A — From a Template

1. Admin → People & Permissions → **Access Policies**
2. Click **Templates**
3. Select the desired template
4. Modify the `domain`, `entity`, and `action` fields in the target if needed
5. Edit the `subject`, `effect`, and `condition` sections in the Policy JSON
6. Optionally toggle **Enable Policy** to activate immediately
7. Click **Save**

Option B — From Scratch

1. Admin → People & Permissions → **Access Policies**
2. Click **Create Policy**
3. Write the full Policy JSON manually
4. Use **Validate Syntax** tab to check for errors before saving

---

Policy JSON Syntax

```
{
  "name": "Policy Name",
  "description": "What this policy does",
  "targets": [
    {
      "domain": "authorization",
      "entity": "grant",
      "action": "add"
    }
  ],
  "subject": {
    "type": "user"
  },
  "effect": "DENY",
  "conditions": {
    "all": [
      {
        "attribute": "subject.role.names",
        "operator": "notContains",
        "value": "Admin"
      }
```

```
    ]
  }
}
```

Condition operators: `equals`, `notEquals`, `contains`, `notContains`, `startsWith`, `in`, `notIn`

Logical operators:

- `any` → OR (at least one condition must be true)
- `all` → AND (every condition must be true)
- These can be nested for complex logic

Validation and Testing

Before deploying a policy:

1. Click **Validate Syntax** tab — checks for:
   - Missing mandatory fields
   - Invalid attributes
   - Incorrect data type comparisons
   - Preset attribute conflicts
2. Click **Test Policy** tab — paste sample subject/resource/environment data and run a simulated evaluation to confirm expected ALLOW/DENY result before enabling

---

Enabling and Managing Policies

Action	How
Enable a policy	Toggle **Enable Policy** on the policy detail page
Disable a policy	Toggle off — policy is retained but not enforced
Edit a policy	Admin → Access Policies → select policy → Edit
Delete a policy	Admin → Access Policies → select policy → Delete
View all policies	Admin → Access Policies → table view with status

---

Permissions Required

Action	Permission
Create / edit / delete policies	Authorization > Policy > Add, Edit, Delete
View policies	Authorization > Policy > View
Enable ABAC org setting	Organization Settings admin access

> ⚠ The ABAC feature itself must be enabled in **Organization Settings** before any policies are enforced, regardless of permissions.

Important Behaviours and Gotchas

- **DENY always wins** — if a user matches both an ALLOW and a DENY policy, they are denied
- **ABAC + RBAC are additive** — a user needs both the RBAC permission AND to satisfy the ABAC ALLOW policy to gain access
- **Policies apply immediately** after enabling — test in non-production if possible
- **Preset attribute names** must use only alphanumeric characters, periods, or underscores — no spaces or special characters
- `profile.fields` **condition is case-insensitive** — field name matching does not require exact case

---

See Also

- **Roles & Permissions** — the RBAC layer that ABAC works alongside
- **Divisions & Access Control** — division scoping, which also intersects with ABAC subject conditions
- **Organization Settings → Security & Compliance** — where ABAC is enabled at the org level
- **User Profile Management** — for context on which profile fields can be restricted

---

Revision #1
Created 12 March 2026 22:31:51 by Cesar Gzz
Updated 13 March 2026 00:20:17 by Cesar Gzz