Xurrent Personal Access Required Token Permissions or Scopes

Last Modified on 12/09/2025 4:59 am EST

This document details all API permissions required by the Exalate Xurrent connector, organized by feature and use case.

Overview

Xurrent uses a **record type + operation** permission model for API access. When creating a Personal Access Token or OAuth application, you must explicitly grant permissions for each record type your integration needs to access.

Key Points:

- Personal Access Tokens **deny all access by default**
- You must add scopes for each record type needed
- Operations include: `Read`, `Create`, `Update`, `Delete`, `All`
- Some operations require specific user roles in addition to API scopes

---

Permission Format

Xurrent permissions follow the format:

```
<record_type>:<operation>
```

Available Operations:

Operation	Description
`Read`	Query and retrieve data
`Create`	Create new records
`Update`	Modify existing records
`Delete`	Delete records
`All`	All operations (Read + Create + Update + Delete)

Examples:

- `Request:Read` - Can read requests
- `Request:Create` - Can create requests
- `Person:Read` - Can read people/users
- `Team:Read` - Can read teams

Minimum Required Permissions

These are the **absolute minimum** permissions required for basic Exalate sync functionality:

Record Type	Operations	Purpose
Request	Read, Create, Update	Core sync functionality

With only these permissions, you can:

- Create new requests from incoming sync
- Update existing requests
- Read request details for outgoing sync

Limitations with minimum permissions:

- Cannot sync comments (need Note permissions)
- Cannot look up users by email (need Person permissions)
- Cannot look up teams (need Team permissions)
- Cannot use request templates (need Request Template permissions for custom fields type)
- Cannot use custom fields with labels (need UI Extension permissions)

Full Feature Permissions

For **full Exalate functionality**, grant these permissions:

Record Type	Operations	Purpose
Request	Read, Create, Update	Core request sync
Note	Read, Create	Comment sync
Person	Read	User lookup (reporter, assignee)
Team	Read	Team assignment (for member field)
Request Template	Read	Template-based requests
UI Extension	Read	Custom field labels/values
Service Instance	Read	Service assignment

Permission by Feature

1. Request Sync (Required)

Permissions Needed:

```
Request: Read, Create, Update
```

Script Usage:

```
// These work with Request permissions only
entity.summary = replica.summary
entity.category = "incident"
entity.status = replica.status
```

---

2. Comment/Note Sync

Permissions Needed:

```
Note: Read, Create
```

Script Usage:

```
// Sync comments
entity.comments = commentHelper.mergeComments(entity, replica)
```

Without Note:Read permission (Outgoing Sync):

- Comments will NOT be read from Xurrent
- Warning logged: `"Unauthorized (401) when fetching comments... API token may lack Note:Read permission"`
- Sync continues without comments (graceful degradation)
- No error thrown

Without Note:Create permission (Incoming Sync):

- Sync FAILS with clear error message
- Error: `"Unauthorized (401) when creating comment... Ensure your API token has Note:Create permission"`
- This ensures users know exactly what permission is missing

---

3. User/People Lookup

Permissions Needed:

```
Person: Read
```

Script Usage:

```
// Look up user by email
def user = nodeHelper.getUserByEmail("john@example.com")

entity.requestedFor = user

// Look up user by ID
def user = nodeHelper.getUserById("12345")
```

Without Person permissions:

- `getUserByEmail()` throws `IssueTrackerException` with 401 error
- Reporter/Assignee enrichment fails

---

4. Team Assignment

Permissions Needed:

```
Team: Read
```

Script Usage:

```
// Look up team by name
def teamId = nodeHelper.getTeamIdByName("Support Team")
entity.teamId = teamId
```

Without Team permissions:

- `getTeamIdByName()` throws `IssueTrackerException` with 401 error
- Team member validation is skipped (graceful degradation)
- Request creation may still work if team ID is known

---

5. Request Templates

Permissions Needed:

```
Request Template: Read
```

Script Usage:

```
// Look up template by name
def templateId = nodeHelper.getTemplateIdByName("IT Support Request")
entity.templateId = templateId
```

Without Request Template permissions:

- `getTemplateIdByName()` throws `IssueTrackerException` with 401 error
- Cannot dynamically look up templates
- Can still use template ID directly if known

6. Custom Fields (UI Extensions)

Permissions Needed:

```
Request Template: Read
UI Extension: Read
```

Script Usage:

```
// Get custom field with validation
def cf = nodeHelper.getCustomField(templateId, "My Field", "Value 1")
entity.customFields["My Field"] = cf

// Get field label
def label = nodeHelper.getCustomFieldLabel(templateId, "custom_field_123")

// Get value label (for dropdowns)
def valueLabel = nodeHelper.getCustomFieldValueLabel(templateId, "company", "value_1")

// Get all field definitions
def definitions = nodeHelper.getCustomFieldDefinitions(templateId)
```

Without UI Extension permissions:

- `getCustomField()` throws error about missing definitions
- `getCustomFieldLabel()` returns internal ID instead of label
- `getCustomFieldValueLabel()` returns internal value instead of label
- Custom fields still sync, but with internal IDs instead of labels

---

7. Service Assignment

Permissions Needed:

```
Service Instance: Read
```

Script Usage:

```
// Look up service by name
def serviceId = nodeHelper.getServiceIdByName("IT Service Desk")
entity.serviceId = serviceId
```

Without Service Instance permissions:

- `getServiceIdByName()` throws `IssueTrackerException` with 401 error
- Can still use service ID directly if known

---

8. Tags/Labels

Permissions Needed:

```
Request: Read, Update
```

Tags are managed through the Request API, no separate Tag permission needed.

Script Usage:

```
// Sync labels/tags
entity.labels = replica.labels
```

---

9. Attachments

Permissions Needed:

```
Request: Read, Update
Note: Create (for attachments on comments)
```

Attachments use the storage facility API which is accessible with Request permissions.

Script Usage:

```
// Sync attachments
entity.attachments = replica.attachments
```

Permission Errors

401 Unauthorized Errors

When a method lacks required permissions, you'll see errors like:

```
IssueTrackerException: Unauthorized: If you want to use this feature (getUserByEmail),
ensure your API token has proper permissions for the People endpoint.
```

Solution: Add the required record type with Read permission to your Personal Access Token.

Graceful Degradation

Some features gracefully degrade instead of throwing errors:

Feature	Behavior without Permission
Note:Read (outgoing)	Comments skipped - sync proceeds without comments

Feature	Behavior without Permission
Team member validation	Skipped - request proceeds
Custom field labels	Returns internal ID
Custom field value labels	Returns internal value

Strict Validation (Throws Exception)

These features throw exceptions with clear 401 messages:

Feature	Error Message
Note:Create (incoming)	"Unauthorized (401) when creating comment... Ensure your API token has Note:Create permission"
Person:Read	"Unauthorized (401) when fetching person. Check API permissions for people endpoint."
Team:Read (lookup)	"Unauthorized: If you want to use this feature (getTeamIdByName), ensure your API token has proper permissions for the Teams endpoint."

Additional Resources

- Xurrent REST API Documentation
- Personal Access Tokens
- OAuth Scopes

Next steps

- Set up your first sync connection
- Configure request synchronization

Have more questions? Ask the community