Acceptance Test Cases

Based on “SRT Multi-tenant Requirement V1 – 2017-02-24.docx”

# A built-in OAGi admin developer account exists

Requirement document reference: Section 4.2

## Successful log in by the built-in OAGi admin developer

Test Steps:

1. A users clicks on the sign-in link.
2. The user attempts to log in with the user name “oagi” and valid password.
3. Verify that the user is successfully logged in and has the OAGi admin developer[[1]](#footnote-1) role.

# OAGi admin developer user management authorities

Requirement document reference: Section 5.1

Pre-condition: Pass “Test Suite 1 A built-in OAGi admin developer account exists”.

## OAGi admin developer successfully adds a user with minimum required information

Requirement document reference: Section 5.1.1

Pre-condition: No user with username, aNewUser1, exists in the system.

Test Steps:

1. A user with the OAGi admin developer role logs in.
2. The OAGi admin user somehow accesses the “Add a user” functionality.
3. The OAGi admin user fills in username=”aNewUser1”, a valid password, first name, last name, phone, email (omit the address and title fields) and create this new user.
4. Log out of the OAGi admin user account.
5. Check aNewUser1’s email for the system generated email account verification, which also tells aNewUser1 his initial password. Click on the verification link.
6. Log in as aNewUser1.
7. Verify that aNewUser1 has a free user role.

## OAGi admin developer successfully adds a user with all information filled

Requirement document reference: Section 5.1.1

Pre-condition: No user with username, aNewUser2, exists in the system.

Test Steps:

1. A user with the OAGi admin developer role logs in.
2. The OAGi admin user somehow accesses the “Add a user” functionality.
3. The OAGi admin user fills in username=”aNewUser2”, a valid password, first name, last name, phone, email, address and title fields and create this new user.
4. Log out of the OAGi admin user account.
5. Check aNewUser2’s email for the system generated email account verification, which also tells aNewUser2 his initial password. Click on the verification link. Log in as aNewUser2.
6. Verify that aNewUser2 has a free user role.

## Add user fails by OAGi admin developer due to invalid email address

Requirement document reference: Section 5.1.1

Pre-condition: No user with username, aNewUser3, exists in the system.

Test Steps:

1. A user with the OAGi admin developer role logs in.
2. The OAGi admin user somehow accesses the “Add a user” functionality.
3. The OAGi admin user fills in username=”aNewUser3”, a valid password, first name, last name, phone, email (assuming valid format but non-existing email address), address and title fields and create this new user.
4. Verify that the OAGi admin user receives an email from the system indicating that the system fails to send an email to aNewUser3’s email address.
5. OAGi admin user goes back to the system to correct the email address.
6. Check aNewUser3’s email for the system generated email account verification, which also tells aNewUser3 his initial password. Click on the verification link. Log in as aNewUser3.
7. Verify that aNewUser3 has a free user role.

## Add user fails by OAGi admin developer due to expired verification email

Requirement document reference: **Optional**, this is not clearly included in Section 5.1.1

Pre-condition: No user with username, aNewUser4, exists in the system.

Test Steps:

1. A user with the OAGi admin developer role logs in.
2. The OAGi admin user somehow accesses the “Add a user” functionality.
3. The OAGi admin user fills in username=”aNewUser4”, a valid password, first name, last name, phone, email, address and title fields and create this new user.
4. Verify that aNewUser4 has received the verification email.
5. Wait until the account verification email expires.
6. Verify that the OAGi admin user receives an email from the system indicating that aNewUser4 fails to verify his email address.
7. Verify that aNewUser4 cannot log in.
8. OAGi admin user goes back to the system to resend the verification email.
9. Wait until the account verification email expires.
10. OAGi admin user goes back to the system to remove all roles from aNewUser4 (note since the user cannot be removed from the system, only all roles can be removed).
11. Verify that aNewUser4 cannot log in.

## Add user fails by OAGi admin developer due to omitting username

Requirement document reference: Section 5.1.1

Test Steps:

1. A user with the OAGi admin developer role logs in.
2. The OAGi admin user somehow accesses the “Add a user” functionality.
3. The OAGi admin user omits the username, but fills in a valid password, first name, last name, phone, email, address and title fields and attempts to create this new user.
4. Verify that the system fails the attempt.

## Add user fails by OAGi admin developer due to omitting password

Requirement document reference: Section 5.1.1

Pre-condition: No user with username, aNewUser10, exists in the system.

Test Steps:

1. A user with the OAGi admin developer role logs in.
2. The OAGi admin user somehow accesses the “Add a user” functionality.
3. The OAGi admin user omits the password field but fills in username=”aNewUser10”, first name, last name, phone, email, address and title fields and attempts to create this new user.
4. Verify that the system fails the attempt or aNewUser10 does not exist in the system.

## Add user fails by OAGi admin developer due to non-compliance password policy

Requirement document reference: **Optional**, this is not clearly included in Section 5.1.1

Pre-condition: No user with username, aNewUser5, exists in the system.

Test Steps:

1. A user with the OAGi admin developer role logs in.
2. The OAGi admin user somehow accesses the “Add a user” functionality.
3. The OAGi admin user fill in a non-compliance password field but fills in username=”aNewUser10”, first name, last name, phone, email, address and title fields and attempts to create this new user.
4. Verify that the system fails the attempt or aNewUser10 does not exist in the system.

## Add user fails by OAGi admin developer due to omitting first name

Requirement document reference: Section 5.1.1

Pre-condition: No user with username, aNewUser3, exists in the system.

Test Steps:

1. A user with the OAGi admin developer role logs in.
2. The OAGi admin user somehow accesses the “Add a user” functionality.
3. The OAGi admin user omits the first name field but fills in username=”aNewUser10”, valid password, last name, phone, email, address and title fields and attempts to create this new user.
4. Verify that the system fails the attempt or aNewUser10 does not exist in the system.

## Add user fails by OAGi admin developer due to omitting last name

Requirement document reference: Section 5.1.1

Pre-condition: No user with username, aNewUser10, exists in the system.

Test Steps: Similar to above but omit the last name field.

## Add user fails by OAGi admin developer due to omitting phone

Requirement document reference: Section 5.1.1

Pre-condition: No user with username, aNewUser10, exists in the system.

Test Steps: Similar to above but omit the phone field.

## Add user fails by OAGi admin developer due to omitting email

Requirement document reference: Section 5.1.1

Pre-condition: No user with username, aNewUser10, exists in the system.

Test Steps: Similar to above but omit the email field.

## Add user fails by OAGi admin developer due to duplicate username and ensuring the username is case insensitive

Requirement document reference: Section 5.1.1

Pre-condition: A user with the username, aNewUser1, already exists (i.e., execute along with “Test Case 2.1 OAGi admin developer successfully adds a user with minimum required information”).

Test Steps:

1. A user with the OAGi admin developer role logs in.
2. The OAGi admin user somehow accesses the “Add a user” functionality.
3. The OAGi admin user fills in username=”ANEWUSER1”, a valid password, first name, last name, phone, email (omit the address and title fields) and attempt create this new user.
4. Verify that system reject the attempt indicating existing username.

## OAGi admin developer successfully assigns an OAGi developer role to a free user.

Requirement document reference: Section 5.1.2

Pre-condition: A user with the username, aNewUser1, already exists with only free use role (i.e., execute in the same session after “Test Case 2.1 OAGi admin developer successfully adds a user with minimum required information”).

Test Steps:

1. A user with the OAGi admin developer role logs in.
2. The OAGi admin user accesses the user management functionality.
3. The OAGi admin assigns the OAGi developer role to aNewUser1 user.
4. Verify that aNewUser1 now has the OAGi developer role.
5. Verify that emails about this change are sent to all admin developers and aNewUser1.

## OAGi admin developer successfully revoke an OAGi developer role to a user.

Requirement document reference: Section 5.1.2

Pre-condition: A user with the username, aNewUser1, has the OAGi developer role (i.e., execute in the same session after “Test Case 2.10 OAGi admin developer successfully assigns an OAGi developer role to a free user.”).

Test Steps:

1. A user with the OAGi admin developer role logs in.
2. The OAGi admin user accesses the user management functionality.
3. The OAGi admin revoke the OAGi developer role from aNewUser1 user.
4. Verify that aNewUser1 now do not have the OAGi developer role and have only the free user role.
5. Verify that emails about this change are sent to all admin developers and aNewUser1.

## OAGi admin developer successfully assigns an OAGi admin developer role to a free user.

Requirement document reference: Section 5.1.2

Pre-condition: A user with the username, aNewUser1, already exists with only free use role (i.e., execute in the same session after “Test Case 2.11 OAGi admin developer successfully revoke an OAGi developer role to a user.”).

Test Steps:

1. A user with the OAGi admin developer role logs in.
2. The OAGi admin user accesses the user management functionality.
3. The OAGi admin assigns the OAGi admin developer role to aNewUser1 user.
4. Verify that aNewUser1 now has the OAGi admin developer role.
5. Verify that emails about this change are sent to all admin developers and aNewUser1.

## OAGi admin developer cannot revoke himself from an OAGi admin developer role

Requirement document reference: Section 5.1.2

Pre-condition: A user with the username, aNewUser1, already exists the OAGi admin developer role (i.e., execute in the same session after “Test Case 2.12 OAGi admin developer successfully assigns an OAGi admin developer role to a free user.”).

Test Steps:

1. The user, aNewUser1, logs in.
2. The user accesses the user management functionality.
3. Verify that aNewUser1 cannot revoke his own OAGi admin developer role.
4. Log out.
5. Log in with the built-in OAGi admin developer account.
6. The built-in user access the user management functionality.
7. Verify that the built-in user cannot revoke his own OAGi admin developer role.

## OAGi admin developer successfully revoke an OAGi admin developer role to a user.

Requirement document reference: Section 5.1.2

Pre-condition: A user with the username, aNewUser1, has the OAGi admin developer role (i.e., execute in the same session after “Test Case 2.12 OAGi admin developer successfully assigns an OAGi admin developer role to a free user.”).

Test Steps:

1. A user, other than aNewUser1, with the OAGi admin developer role logs in.
2. The OAGi admin user accesses the user management functionality.
3. The OAGi admin revoke the OAGi admin developer role from aNewUser1 user.
4. Verify that aNewUser1 now do not have the OAGi admin developer role but simply a free user role.
5. Verify that emails about this change are sent to all OAGi admin developers and aNewUser1.

## Two OAGi admin developers cannot at the same time remove each other’s the admin developer role.

Requirement document reference: Section 5.1.2

Pre-condition: At least two users with the OAGi admin developer role exist in the system.

Test Steps:

1. Two OAGi admin developers log in to the system in two separate sessions.
2. Both OAGi admin developers accesses the user management functionality.
3. At the same time both admin developers attempt to revoke the OAGi admin developer roles from one another.
4. Verify that one of the admin developers fails to revoke.

## This is an alternate to Test Case 2.18. The built-in OAGi admin developer account can never be revoked by another admin developer.

Requirement document reference: Section 5.1.2

Pre-condition: A user with the username, aNewUser1, has the OAGi admin developer role

Test Steps:

1. ANewUser1 logs in.
2. ANewUser1 accesses the user management functionality.
3. Verify that it is not possible to revoke the OAGi admin developer role from the built-in OAGi admin developer account.

## A user cannot have both OAGi admin developer OAGi developer roles

Requirement document reference: Section 5.1.2

Pre-condition: There are at least two OAGi admin developers and one OAGi developer in the system.

Test Steps:

1. An OAGi admin developer log in.
2. The OAGi admin developer accesses the user management functionality.
3. Verify that it is not possible to add an OAGi developer role to another existing OAGi admin developer.
4. Verify that it is not possible to add an OAGi admin developer role to an existing OAGi developer.

## OAGi admin developer cannot manage roles of the root user

Requirement document reference: Section 3.2.6

Test Steps:

1. An OAGi admin developer logs in.
2. The OAGi admin user accesses the user management functionality.
3. Verify that it is not possible to for the OAGi admin developer to assign or revoke any role of the root user.

# OAGi admin developer onboarding an enterprise tenant

Requirement document reference: Section 4.1.1

## OAGi admin onboards a new enterprise tenant (for the first time)

Pre-condition: A user, OAGD1, with the OAGi developer role exist.

Test steps:

1. A user, OAGADx, logs in as an OAGi admin.
2. OAGADx creates a new enterprise tenant, ENTa, filling in all the information except the title field with a valid email address. (Assertion #1)
3. Verify that the verification email is sent out to the indicated email address. (Assertion #2)
4. OAGADx creates a new enterprise tenant, ENTb, filling in all the information except the email address. Verify that the system does not allow that. (Assertion #3)
5. OAGADx goes back and fill in the email address field but with an invalid format. Verify that the system does not allow that. (Assertion #4)
6. OAGADx updates some information about ENTa. (Assertion #5).
7. OAGADx creates a new user, ENTaADMIN.
8. Verify that OAGADx can assign an ENTa admin role to ENTaADMIN. (Assertion #6)
9. OAGADx logs out.
10. ENTaADMIN logs in as the ENTa admin. Verify that ENTaADMIN cannot create a new enterprise tenant. (Assertion #7.1)
11. ENTaADMIN logs out.
12. A user, ENTUSER1, log in as an enterprise end user. Verify that ENTUSER1 cannot create a new enterprise tenant. (Assertion #7.2)
13. ENTUSER1 logs out.
14. A user, OAGD1, logs in as an OAGi developer. Verify that he cannot create a new enterprise tenant. (Assertion #7.3)
15. OAGD1 logs out.

Test assertions:

1. OAGi admin can create a new enterprise tenant with the optional title element left off.
2. The email address verification email shall be sent out. (Optional)
3. The email address of the enterprise tenant cannot be left out. (Optional)
4. The system does not allow an invalid email address format.
5. OAGi admin can update enterprise tenant information.
6. OAGi admin can assign the admin role of the new enterprise tenant to a user.
7. No other user roles can create a new enterprise.
   1. An enterprise admin user cannot.
   2. An enterprise end user cannot.
   3. An OAGi developer cannot.

## OAGi admin deactivate an enterprise tenant and activate again

Pre-condition: An enterprise tenant, ENTa, exists with some admins and end users.

Test steps:

1. A user, OAGADx, logs in as an OAGi admin.
2. OAGADx deactivate ENTa. (Maybe an email should be sent out to the email address on file). (Assertion #1)
3. OAGADx logs out.
4. A user with an ENTa admin role tries to log in with that role. Verify that the system does not allow. (Assertion #2)
5. A user with an ENTa end user roles tries to log in with that role. Verify that the system does not allow. (Assertion #2)
6. OAGADx logs in as an OAGi admin.
7. OAGADx reactivate ENTa tenant. (Assertion #4)
8. OAGADx logs out.
9. Verify that a user with an ENTa admin role can log in with that role. (Assertion #5)
10. Verify that a user with an ENTa end user role can log in with that role. (Assertion #5)

Test assertions:

1. OAGi admin can deactivate an enterprise tenant.
2. Users with the deactivated enterprise tenant’s role cannot log in with that role.
3. No user can be assigned a role in the deactivated enterprise tenant (optional) – not in the test step.
4. A deactivated enterprise tenant can be reactivated.
5. Users with the reactivated enterprise tenant’s role can log in with that role.

# OAGi developer’s user management authorities

Requirement document reference: Section 5.2.

## OAGi developer can change the contact info but not username

Requirement document reference: Section 5.1.2

Pre-condition: A user with the OAGi developer role exists

Test Steps:

1. An OAGi deverloper logs in.
2. The OAGi developer accesses the user management functionality to update his info.
3. Verify that the username cannot be changed.
4. The OAGi developer changes his first name, last name, phone, address, title.
5. Verify that the information was properly updated in the system.

## OAGi developer can change his email address

Requirement document reference: Section 5.1.2

Pre-condition: A user with the OAGi developer role exists

Test Steps:

1. An OAGi deverloper logs in.
2. The OAGi developer accesses the user management functionality to update his info.
3. The OAGi developer changes his email address.
4. Verify that the information was properly updated in the system.

Note: There is no verification email generated or required. If we assume that the user wants to change his email address because he no longer has access to his previous email address there is no way to notify him of an invalid new email address anyway. In the future, the system can notify the admin developer but this is still not absolutely reliable anyway b/c the admin developer may change his email to an invalid one at the same time period.

## OAGi developer can change his password

Requirement document reference: Section 5.1.2

Pre-condition: A user with the OAGi developer role exists

Test Steps:

1. An OAGi deverloper logs in.
2. The OAGi developer accesses the user management functionality to update his password.
3. The OAGi developer changes his password to a new compliance password.
4. Verify that the OAGi developer can log in with his new password.

## OAGi developer cannot change his password to a non-compliance password

Requirement document reference: **Optional**. Section 5.1.2

Pre-condition: A user with the OAGi developer role exists

Test Steps:

1. An OAGi deverloper logs in.
2. The OAGi developer accesses the user management functionality to update his password.
3. The OAGi developer changes his password to a non-compliance password.
4. Verify that the system does not allow the password change.

## OAGi developer cannot manage roles and access info of any other user

Requirement document reference: Section 3.2.6, 5.1.2

Pre-condition: A user with the OAGi developer role exists. Users with all other roles exist.

Test Steps:

1. An OAGi developer logs in.
2. Verify that it is not possible to for the OAGi developer to manage other users.

# Enterprise tenant admin user’s user management authorities

Requirement document reference: Section 5.3

## Enterprise tenant admin user successfully adds a user with minimum required information

Requirement document reference: Section 5.3.1

Pre-condition: No user with username, aEntUser1, exists in the system.

Test Steps:

1. A user logs in as Enterprise tenant admin.
2. The Enterprise tenant admin somehow accesses the “Add a user” functionality.
3. The Enterprise tenant admin user fills in username=”aEntUser1”, a valid password, first name, last name, phone, email (omit the address and title fields) and create this new user.
4. Log out of the Enterprise tenant admin user account.
5. Check aEntUser1’s email for the system generated email account verification, which also tells aEntUser1 his initial password. Click on the verification link.
6. Log in as aEntUser1.
7. Verify that aEntUser1 has a free user role.

## Enterprise tenant admin user successfully adds a user with all information filled

Requirement document reference: Section 5.3.1

Pre-condition: No user with username, aEntUser2, exists in the system.

Test Steps:

1. A user logs in as Enterprise tenant admin.
2. The Enterprise tenant admin user somehow accesses the “Add a user” functionality.
3. The Enterprise tenant admin user fills in username=” aEntUser2”, a valid password, first name, last name, phone, email, address and title fields and create this new user.
4. Log out of the Enterprise tenant admin user account.
5. Check aEntUser2’s email for the system generated email account verification, which also tells aEntUser2 his initial password. Click on the verification link. Log in as aEntUser2.
6. Verify that aEntUser2 has a free user role.

## Add user fails by Enterprise tenant admin user due to invalid email address

Requirement document reference: Section 5.3.1

Pre-condition: No user with username, aEntUser3, exists in the system.

Test Steps:

1. A user logs in as Enterprise tenant admin..
2. The Enterprise tenant admin user somehow accesses the “Add a user” functionality.
3. The Enterprise tenant admin user fills in username=” aEntUser3”, a valid password, first name, last name, phone, email (assuming valid format but non-existing email address), address and title fields and create this new user.
4. Verify that the Enterprise tenant admin user receives an email from the system indicating that the system fails to send an email to aEntUser3’s email address.
5. Enterprise tenant admin user goes back to the system to correct the email address.
6. Check aEntUser3’s email for the system generated email account verification, which also tells aEntUser3 his initial password. Click on the verification link. Log in as aEntUser3.
7. Verify that aEntUser3 has a free user role.

## Add user fails by Enterprise tenant admin user due to expired verification email

Requirement document reference: **Optional**, this is not clearly included in Section 5.3.1

Pre-condition: No user with username, aEntUser4, exists in the system.

Test Steps:

1. A user logs in as Enterprise tenant admin.
2. The Enterprise tenant admin user somehow accesses the “Add a user” functionality.
3. The Enterprise tenant admin user fills in username=”aEntUser4”, a valid password, first name, last name, phone, email, address and title fields and create this new user.
4. Verify that aEntUser4 has received the verification email.
5. Wait until the account verification email expires.
6. Verify that the Enterprise tenant admin user receives an email from the system indicating that aEntUser4 fails to verify his email address.
7. Verify that aEntUser4 cannot log in.
8. Enterprise tenant admin goes back to the system to resend the verification email.
9. Wait until the account verification email expires.
10. Enterprise tenant admin goes back to the system to remove all roles from aEntUser4 (note since the user cannot be removed from the system, only all roles can be removed).
11. Verify that aEntUser4 cannot log in.

## Add user fails by Enterprise tenant admin user due to omitting username

Requirement document reference: Section 5.3.1

Test Steps:

1. A user logs in as Enterprise tenant admin.
2. The Enterprise tenant admin user somehow accesses the “Add a user” functionality.
3. The Enterprise tenant admin omits the username, but fills in a valid password, first name, last name, phone, email, address and title fields and attempts to create this new user.
4. Verify that the system fails the attempt.

## Add user fails by Enterprise tenant admin user due to omitting password

Requirement document reference: Section 5.3.1

Pre-condition: No user with username, aEntUser10, exists in the system.

Test Steps:

1. A user logs in as Enterprise tenant admin.
2. The Enterprise tenant admin user somehow accesses the “Add a user” functionality.
3. The Enterprise tenant admin user omits the password field but fills in username=”aEntUser10”, first name, last name, phone, email, address and title fields and attempts to create this new user.
4. Verify that the system fails the attempt or aEntUser10 does not exist in the system.

## Add user fails by Enterprise tenant admin user due to non-compliance password policy

Requirement document reference: **Optional**, this is not clearly included in Section 5.3.1

Pre-condition: No user with username, aEntUser5, exists in the system.

Test Steps:

1. A user logs in as Enterprise tenant admin.
2. The Enterprise tenant admin user somehow accesses the “Add a user” functionality.
3. The Enterprise tenant admin user fill in a non-compliance password field but fills in username=”aEntUser10”, first name, last name, phone, email, address and title fields and attempts to create this new user.
4. Verify that the system fails the attempt or aEntUser10 does not exist in the system.

## Add user fails by Enterprise tenant admin user due to omitting first name

Requirement document reference: Section 5.3.1

Pre-condition: No user with username, aEntUser3, exists in the system.

Test Steps:

1. A user logs in as Enterprise tenant admin.
2. The Enterprise tenant admin user somehow accesses the “Add a user” functionality.
3. The Enterprise tenant admin user omits the first name field but fills in username=”aEntUser3”, valid password, last name, phone, email, address and title fields and attempts to create this new user.
4. Verify that the system fails the attempt or aEntUser3 does not exist in the system.

## Add user fails by Enterprise tenant admin user due to omitting last name

Requirement document reference: Section 5.3.1

Pre-condition: No user with username, aEntUser11, exists in the system.

Test Steps: Similar to above but omit the last name field.

## Add user fails by Enterprise tenant admin user due to omitting phone

Requirement document reference: Section 5.3.1

Pre-condition: No user with username, aEntUser12, exists in the system.

Test Steps: Similar to above but omit the phone field.

## Add user fails by Enterprise tenant admin user due to omitting email

Requirement document reference: Section 5.3.1

Pre-condition: No user with username, aEntUser13, exists in the system.

Test Steps: Similar to above but omit the email field.

## Add user fails by Enterprise tenant admin user due to duplicate username and ensuring the username is case insensitive

Requirement document reference: Section 5.3.1 (**Optional**)

Pre-condition: A user with the username, aEntUser1, already exists.

Test Steps:

1. A user logs in as Enterprise tenant admin.
2. The Enterprise tenant admin user somehow accesses the “Add a user” functionality.
3. The Enterprise tenant admin user fills in username=”AENTUSER1”, a valid password, first name, last name, phone, email (omit the address and title fields) and attempt to create this new user.
4. Verify that system reject the attempt.

## Enterprise tenant admin successfully assigns an Enterprise tenant end user role to a free user

Requirement document reference: Section 5.3.2

Pre-condition: A user with the username, aEntUser1, already exists with only free use role. There are at least 2 users with the enterprise tenant admin user role in the same enterprise.

Test Steps:

1. A user logs in as Enterprise tenant admin.
2. The Enterprise tenant admin accesses the user management functionality.
3. The Enterprise tenant admin assigns the Enterprise tenant end user role to aEntUser1 user.
4. Verify that aEntUser1 now has the Enterprise tenant end user role.
5. Verify that emails about this change are sent to all enterprise tenant admin users and aEntUser1.

## Enterprise tenant admin successfully revoke an Enterprise tenant end user role from a user

Requirement document reference: Section 5.3.2

Pre-condition: A user with the username, aEntUser1, has only one Enterprise tenant end user. There are at least 2 users with the enterprise tenant admin user role in the same enterprise.

Test Steps:

1. A user logs in as Enterprise tenant admin.
2. The Enterprise tenant admin accesses the user management functionality.
3. The Enterprise tenant admin revoke the Enterprise tenant end user role from aEntUser1 user.
4. Verify that aEntUser1 now do not have the Enterprise tenant end user role and have only the free user role.
5. Verify that emails about this change are sent to all enterprise tenant admin users and aEntUser1.

## Enterprise tenant admin successfully assigns an Enterprise tenant end user role to an OAGi user.

Requirement document reference: Section 5.3.2, 3.3

Pre-condition: A user with the username, aNewUser1, already exists with OAGi developer role. There are at least 2 users with the enterprise tenant admin user role in the same enterprise.

Test Steps:

1. A user logs in as Enterprise tenant admin.
2. The Enterprise tenant admin user accesses the user management functionality.
3. The Enterprise tenant admin assigns the Enterprise tenant end user role to aNewUser1 user.
4. Verify that aNewUser1 now has the Enterprise tenant end user role.
5. Verify that emails about this change are sent to all enterprise admin users and aNewUser1.

## Enterprise tenant admin successfully revoke an Enterprise tenant end user role from an OAGi user.

Requirement document reference: Section 5.3.2, 3.3

Pre-condition: A user with the username, aNewUser1, has the OAGi developer role and the Enterprise tenant end user role. There are at least 2 users with the enterprise tenant admin user role in the same enterprise.

Test Steps:

1. A user logs in as Enterprise tenant admin.
2. The Enterprise tenant admin user accesses the user management functionality.
3. The Enterprise tenant admin revoke the Enterprise tenant end user role from aNewUser1 user.
4. Verify that aNewUser1 now do not have the Enterprise tenant end user.
5. Verify that emails about this change are sent to all enterprise admin users and aNewUser1.
6. Verify that aNewUser1 can still log in as an OAGi developer.

## Enterprise tenant admin successfully assigns an Enterprise tenant admin user role to a free user.

Requirement document reference: Section 5.3.2

Pre-condition: A user with the username, aEntUser1, already exists with only free use role. There are at least 2 users with the enterprise tenant admin user role in the same enterprise.

Test Steps:

1. A user logs in as an enterprise admin user.
2. The enterprise admin user accesses the user management functionality.
3. The enterprise admin user assigns the enterprise admin user role to aEntUser1 user.
4. Verify that aEntUser1 now has the Enterprise tenant admin user role.
5. Verify that emails about this change are sent to all enterprise admin users and aEntUser1.

## Enterprise tenant admin can assign an enterprise admin user role to an OAGi developer.

Requirement document reference: Section 5.3.2

Pre-condition: A user with the username, OAGD1, already exists with only OAGi developer role. There are at least 2 users with the enterprise tenant admin user role in the same enterprise.

Test Steps:

1. A user logs in as an enterprise admin user.
2. The enterprise admin user accesses the user management functionality.
3. The enterprise admin user assigns the enterprise admin user role to OAGD1.
4. Verify that OAGD1 now has the Enterprise tenant admin user role.
5. Verify that emails about this change are sent to all enterprise admin users and aEntUser1.

## Enterprise tenant admin can assign an Enterprise tenant admin user role to an enterprise end user.

Requirement document reference: Section 5.3.2

Pre-condition: A user with the username, aEntUser1, already exists with only free use role. There are at least 2 users with the enterprise tenant admin user role.

Test Steps:

1. A user logs in as an enterprise admin user.
2. The enterprise admin user accesses the user management functionality.
3. The enterprise admin user assigns the enterprise admin user role to aEntUser1 user.
4. Verify that aEntUser1 now has the Enterprise tenant admin user role and not the enterprise end user role.
5. Verify that emails about this change are sent to all enterprise admin users and aEntUser1.

## Enterprise tenant admin cannot revoke himself from an Enterprise tenant admin user role

Requirement document reference: Section 5.3.2

Pre-condition: A user with the username, aEntUser1, already exists the Enterprise tenant admin user role.

Test Steps:

1. The user, aEntUser1, logs in.
2. The user accesses the user management functionality.
3. Verify that aEntUser1 cannot revoke his own Enterprise tenant admin role.

## Enterprise tenant admin successfully revoke an Enterprise tenant admin user role from a user with only one enterprise admin user role.

Requirement document reference: Section 5.3.2

Pre-condition: A user with the username, aEntUser1, has only one Enterprise tenant admin user role exists. There are at least two other admin users of that enterprise.

Test Steps:

1. A user, other than aEntUser1, with the Enterprise tenant admin role logs in.
2. The Enterprise tenant admin user accesses the user management functionality.
3. The Enterprise tenant admin revoke the Enterprise tenant admin role from aEntUser1 user.
4. Verify that aEntUser1 now only have the free user role.
5. Verify that emails about this change are sent to all Enterprise tenant admin users and aEntUser1.

## Two Enterprise tenant admins cannot at the same time remove each other’s the admin user role.

Requirement document reference: Section 5.3.2

Pre-condition: At least two users with the Enterprise tenant admin role exist in the system.

Test Steps:

1. Two Enterprise tenant admins log in to the system in two separate sessions.
2. Both Enterprise tenant admins accesses the user management functionality.
3. At the same time both admin users attempt to revoke the admin user roles from one another.
4. Verify that one of the admin users fails to revoke.

## Enterprise tenant admin cannot manage roles of the root user

Requirement document reference: Section 3.2.6

Test Steps:

1. An Enterprise tenant admin user logs in.
2. The Enterprise tenant admin accesses the user management functionality.
3. Verify that it is not possible to for the Enterprise tenant admin user to assign or revoke any role of the root user.

## Enterprise tenant admin cannot view/edit info of any other user

Requirement document reference: Section 5.3.3

Pre-condition: A user with the Enterprise tenant admin role exists. Users with all other roles exist.

Test Steps:

1. An Enterprise tenant admin logs in.
2. Verify that it is not possible to for the Enterprise tenant admin to view and edit info of other users.

## An enterprise end user can still log in when one of his enterprise end user roles is revoked.

# Enterprise tenant end user’s user management authorities

Requirement document reference: Section 5.4

## Enterprise tenant end user can change the contact info but not username

Requirement document reference: Section 5.4

Pre-condition: A user with the Enterprise tenant end user role exists.

Test Steps:

1. An Enterprise tenant end user logs in.
2. The Enterprise tenant end user accesses the user management functionality to update his info.
3. Verify that the username cannot be changed.
4. The Enterprise tenant end user changes his first name, last name, phone, address, title.
5. Verify that the information was properly updated in the system.

## Enterprise tenant end user can change his email address

Requirement document reference: Section 5.4

Pre-condition: A user with the Enterprise tenant end user role exists

Test Steps:

1. An Enterprise tenant end user logs in.
2. The Enterprise tenant end user accesses the user management functionality to update his info.
3. The Enterprise tenant end user changes his email address.
4. Verify that the information was properly updated in the system.

Note: There is no verification email generated or required. If we assume that the user wants to change his email address because he no longer has access to his previous email address there is no way to notify him of an invalid new email address anyway. In the future, the system can notify the admin developer but this is still not absolutely reliable anyway b/c the admin developer may change his email to an invalid one at the same time period.

## Enterprise tenant end user cannot manage roles and access info of any other user

Requirement document reference: Section 5.3.3

Pre-condition: A user with the Enterprise tenant end user role exists. Users with all other roles exist.

Test Steps:

1. An Enterprise tenant end user logs in.
2. Verify that it is not possible to for the Enterprise tenant end user to manage other users.

## Enterprise tenant end user cannot change the contact info due to omitting required information

Requirement document reference: Section 5.4

Pre-condition: A user with the Enterprise tenant end user role exists.

Test Steps:

1. An Enterprise tenant end user logs in.
2. The Enterprise tenant end user accesses the user management functionality to update his info.
3. The Enterprise tenant end user changes user information while omitting some required data (first name, last name, phone, address, title).
4. Verify that system rejects attempt.

# A user can never be removed

Requirement document reference: Section 5.5

## A user cannot be removed

## Free user role can be removed from a user and the user can’t then log in

## A user with no role can be reassigned with a free user role and then log in

# Public tenant signs up for a free user account

Requirement document reference: Section 3.4.3.4, 5.1.1, 5.6

## Successful creation of the Free User Account with minimal required information

Pre-condition: The application is in a new session, not logged in. There is no username, “serm”, existing in the database.

Test Steps:

1. A user open up the SRT home page.
2. The user click on the link to create a login account.
3. The user fill in username=”serm”, password = “sermserm”, first name, last name, phone, email (omit the address and title fields)
4. The user click the create account button.

Post-condition: User account “serm” exist with the role Free User.

## Successful creation of the Free User Account with all information filled

Pre-condition: The application is in a new session, not logged in.

Test Steps:

1. A user open up the SRT home page.
2. The user click on the link to create a login account.
3. The user fill in username=”kbserm”, password=”sermserm”, first name, last name, phone, email, omit the address and title fields.
4. The user click create account button.

Post-condition: Verify that the user account is created as a free user role.

## Successful log-in by a free user

Pre-condition: User “serm” with password “sermserm” exists with at least the free user role.

Recommendation: Executed right after Test Case 6.1 or Test Case 6.2

Test Steps:

1. A users clicks on the sign-in link.
2. The user enters the user name “serm” and password=”sermserm”.
3. The user clicks on the log-in link.

Post-condition: The user is taken to a user dashboard page.

## Failed log-in by a free user

Pre-condition: A user with only free user role whose user name is “serm” and password “sermserm” exists (i.e., executed after “Test Case 6.1 Successful creation of the Free User Account with minimal required information” or “Test Case 6.2 Successful creation of the Free User Account with all information filled”)

Test Steps:

1. A users clicks on the sign-in link.
2. The user enters the user name “serm” and password=”SermSerm”.
3. The user clicks on the log-in link.
4. Verify that the user is presented with an invalid log-in result.

## Fail creation of the free user account due to the username omission

Pre-condition: The application is in a new session, not logged in.

Test Steps:

1. A user open up the SRT home page.
2. The user click on the link to create a login account.
3. The user omits the username, but fill in password, first name, last name, phone, email, the address and title fields.
4. The user click create account button.

Post-condition: Verify that the system fails the user account creation.

## Fail creation of the free user account due to the password omission.

Pre-condition: The application is in a new session, not logged in.

Test Steps:

1. A user open up the SRT home page.
2. The user click on the link to create a login account.
3. The user fill in the username, first name, last name, phone, email, omit the address and title fields but omit the password field.
4. The user click create account button.

Post-condition: Verify that the system fails the user account creation.

## Fail creation of the free user account due to the first name omission.

Test Steps: Similar to above but the user omits the first name.

## Fail creation of the free user account due to the last name omission.

Test Steps: Similar to above but the user omits the last name.

## Fail creation of the free user account due to the phone omission.

Test Steps: Similar to above but the user omits the first name.

## Fail creation of the free user account due to the email omission.

Test Steps: Similar to above but the user omits the email.

## Fail creation of the free user account due to a duplicate username (and ensuring the username is case insensitive).

Pre-condition: The application is in a new session, not logged in. There is already a username “serm” in the system.

Test Steps:

1. A user opens up the SRT home page.
2. The user clicks on the link to create a login account.
3. The user fills in the username = “Serm”, password, first name, last name, phone, email, omit the address and title fields.
4. The user clicks create account button.
5. Verify that the system fails the user account creation indicating that the user already exists.

# Free user access right to SRT core functions[[2]](#footnote-2)

Requirement document reference: Section 3.2.5, 3.4.3.3, 7 row 12

## Free user can search, view, and generate XML schema expression from free profile BOD and nothing else.

Pre-condition: A free user account exists. Free profile BODs published by the OAGi tenant exist in the system. Private and shared OAGi tenant profile BODs exist in the system. Private and shared enterprise tenants profile BODs exist in the system.

Test Steps:

1. Execute “Test Case 7.3 Successful log-in by a free user”.
2. Verify that the user is not presented with the ability to view context classification schemes.
3. Verify that the user is not presented with the ability to directly view business contexts (except under the purview of the profile BOD).
4. Verify that the user is not allowed to create any CC or BIE artifact.
5. The user searches for or picks a free profile BOD and makes a request to view a particular free profile BODs.
6. Verify that the user can view the content of the selected free profile BOD along with its business context information.
7. Verify that the user cannot see any private or shared profile BOD, nor the core component.
8. The user goes to the expression generation page.
9. Verify that the user can see the list of free profile BODs.
10. Verify that the user cannot see any private or shared profile BOD.
11. The user select one of the free profile BOD.
12. The user click to generate the profile BOD in XML schema expression.

# OAGi developer access right to SRT core functions

Requirement document reference: Section 3.2.3, 3.4.3.2

## OAGi developer’s authorized management of context categories

Pre-condition: At least two users, say OAGD1 and OAGD2, with both OAGi developer and enterprise end user roles exist. There are context categories created by various enterprise users[[3]](#footnote-3), some are created by OAGD1 and OAGD2 in the enterprise end user capacity, and some of which have been shared.

Test Step:

1. An OAGi admin developer logs into the system.
2. The admin developer creates new context categories, says CAT0 and CAT1.
3. The admin developer shares CAT0.
4. The admin developer logs out.
5. An OAGi developer, says OAGD1, logs into the system.
6. OAGD1 creates two new context category, says CAT2 and CAT3. (Assertion #1)
7. OAGD1 edits CAT2. (Assertion #2)
8. OAGD1 cannot share CAT2. (Assertion #12)
9. OAGD1 log out.
10. Another OAGi developer, says OAGD2, logs into the system.
11. OAGD2 views the list of context categories.
12. Verify that enterprise users’ context categories, which have not been shared, are not visible to OAGD2 in the list. Make sure that the verification statements include context categories that are created by OAGD1 and OAGD2 in the enterprise end user capacity. (Assertion #3)
13. Verify that enterprise users’ context categories, which have been shared, are visible to OAGD2 in the list. (Assertion #4)
14. Verify that OAGD2 can see the details of those visible shared enterprise users’ context categories but cannot make any change (verify with one created by other users and another one created by OAGD2 in the enterprise end user capacity). (Assertion #5)
15. Verify that CAT0, CAT1, CAT2, and CAT3 are in the list. (Assertion #6)
16. Verify that OAGD2 can see the details of CAT0, CAT1, CAT2, and CAT3. (Assertion #7)
17. OAGD2 edits CAT1. (Assertion #11)
18. OAGD2 deletes CAT1 and CAT2. (Assertion #8)
19. Verify that OAGD2 cannot edit or delete CAT0. (Assertion #9)
20. OAGD2 edits CAT3. (Assertion #10)
21. Verify that OAGD2 cannot share CAT3. (Assertion #12)
22. OAGD2 logs out.
23. The admin developer logs in again.
24. The admin developer shares CAT3.
25. The admin developer logs out.
26. OAGD2 logs in as an OAGi developer.
27. Verify that OAGD2 cannot make change or delete CAT3. (Assertion #9)

Test assertions covered by this test case:

1. The OAGi developer can create a context category.
2. The OAGi developer can edit a context category created by himself.
3. The OAGi developer cannot see in the context category list, enterprise users’ private context categories.
4. The OAGi developer can see in the context category list, enterprise users’ shared context categories..
5. The OAGi developer can see the details of enterprise user’s shared context categories.
6. The OAGi developer can see in the list context categories in any status created by others in the OAGi tenant.
7. The OAGi developer can see details of context categories in any status created by others in the OAGi tenant.
8. The OAGi developer can delete context categories created by others in the OAGi tenant.
9. The OAGi developer can no longer change or delete a shared context category created by OAGi admin or OAGi developers.
10. The OAGi developer can edit a context category even if it is created by other OAGi developers.
11. The OAGi developer can edit a context category even if it is created by other OAGi admins.
12. The OAGi developer cannot share a context category.

## OAGi developer’s authorized management of context schemes

Pre-condition: At least two users, say OAGD1 and OAGD2, with both OAGi developer and enterprise end user roles, exist. There are context schemes created by various enterprise users, some are created by OAGD1 and OAGD2 in the enterprise end user capacity, and some of which have been shared.

Test Step:

1. An OAGi admin developer logs into the system.
2. The admin developer creates new context schemes (with some values), says CS0 and CS1.
3. The admin developer shares CS0.
4. The admin developer logs out.
5. An OAGi developer, says OAGD1, logs into the system.
6. OAGD1 creates two new context schemes, says CS2 and CS3. Verify that both context categories that are owned by OAGi tenant and that are shared by enterprise tenants are selectable. Verify that enterprise tenants’ private context categories are not selectable. (Assertion #1, #2)
7. OAGD1 edits CS2. (Assertion #3)
8. OAGD1 cannot share CS2. (Assertion #14)
9. OAGD1 log out.
10. Another OAGi developer, says OAGD2, logs into the system.
11. OAGD2 views the list of context schemes.
12. Verify that enterprise users’ context schemes, which have not been shared, are not visible to OAGD2 in the list. Make sure that the verification statements include context schemes that are created by OAGD1 and OAGD2 in the enterprise end user capacity. (Assertion #4)
13. Verify that enterprise users’ context schemes, which have been shared, are visible to OAGD2 in the list. (Assertion #5)
14. Verify that OAGD2 can see the details of those visible shared enterprise users’ context schemes but cannot make any change (verify with one created by other users and another one created by OAGD2 in the enterprise end user capacity). (Assertion #6)
15. Verify that CS0, CS1, CS2, and CS3 are in the list. (Assertion #7)
16. Verify that OAGD2 can see the details of CS0, CS1, CS2, and CS3. (Assertion #8)
17. OAGD2 edits CS1. (Assertion #13)
18. OAGD2 deletes CS1 and CS2. (Assertion #9)
19. Verify that OAGD2 cannot delete CS0. (Assertion #10)
20. OAGD2 edits CS3. (Assertion #12)
21. Verify that OAGD2 cannot share CS3. (Assertion #14)
22. OAGD2 logs out.
23. The admin developer logs in again.
24. The admin developer shares CS3.
25. The admin developer logs out.
26. OAGD2 logs in as an OAGi developer.
27. Verify that OAGD2 cannot make change or delete CS3. (Assertion #11)

Test assertions covered by this test case:

1. The OAGi developer can create a context scheme.
2. Context categories belonging to the OAGi tenant and those shared by enterprise tenants shall be selectable by OAGi developers for association with the context schemes and those private ones are not selectable.
3. The OAGi developer can edit a context scheme he created.
4. The OAGi developer cannot see in the context scheme list, enterprise users’ private context schemes.
5. The OAGi developer can see in the context scheme list, enterprise users’ shared context schemes.
6. The OAGi developer can see the details of enterprise user’s shared context schemes.
7. The OAGi developer can see in the list context schemes in any status created by others in the OAGi tenant.
8. The OAGi developer can see details of context schemes in any status created by others in the OAGi tenant.
9. The OAGi developer can delete context schemes created by others in the OAGi tenant.
10. The OAGi developer can no longer change or delete a shared context schemes created by OAGi admin.
11. The OAGi developer can no longer change or delete a shared context schemes created by OAGi developers.
12. The OAGi developer can edit a context scheme even if it is created by other OAGi developers.
13. The OAGi developer can edit a context scheme even if it is created by other OAGi admins.
14. The OAGi developer cannot share a context scheme.

## OAGi developer cannot delete context categories used by a context scheme

Pre-condition: At least a user, say OAGD1, with the OAGi developer role exists. There are private context schemes created by various OAGi tenant roles[[4]](#footnote-4). One context scheme uses a private context category, says CAT1, owned by OAGD1 and another context scheme uses a shared context category, says CAT2, owned by OAGD1.

Test Step:

1. OAGD1 logs into the system as an OAGi developer.
2. Verify that OAGD1 cannot delete CAT1.
3. Verify that OAGD2 cannot delete CAT2.
4. OAGD1 log out.

## OAGi developer cannot delete context schemes used by a business context

Pre-condition: At least a user, say OAGD1, with the OAGi developer role exists. There are business contexts created by various OAGi tenant roles. One business context, says BC1, uses a private context scheme, CS1, owned by OAGD1 and another business context uses a shared context scheme, says CS2, owned by OAGD1.

Test Step:

1. OAGD1, logs into the system.
2. Verify that OAGD1 cannot delete CS1.
3. Verify that OAGD1 cannot delete CS2.
4. OAGD1 log out.

## OAGi developer authorized management of business contexts

Pre-condition: At least two users, say OAGD1 and OAGD2, with both the OAGi developer and enterprise end user roles exist. There are business contexts created by various enterprise users, some are created by OAGD1 and OAGD2 in the enterprise end user capacity, and some of which have been shared.

Test Step:

1. An OAGi admin developer logs into the system.
2. The admin developer creates new business contexts (with some context scheme values), says BC0 and BC1.
3. The admin developer shares BC0.
4. The admin developer logs out.
5. An OAGi developer, says OAGD1, logs into the system.
6. OAGD1 creates two new business contexts, says BC2 and BC3. Verify that context categories and context schemes belonging to OAGi tenants and shared by enterprise tenants are selectable. Verify that context categories and context schemes private to enterprise tenants are not selectable. (Assertion #1, 2)
7. OAGD1 edits BC2. (Assertion #3)
8. OAGD1 cannot share BC2. (Assertion #14)
9. OAGD1 log out.
10. Another OAGi developer, says OAGD2, logs into the system.
11. OAGD2 views the list of business contexts.
12. Verify that enterprise users’ business contexts, which have not been shared, are not visible to OAGD2 in the list. Make sure that the verification statements include business context that are created by OAGD1 and OAGD2 in the enterprise end user capacity. (Assertion #4)
13. Verify that enterprise users’ business contexts, which have been shared, are visible to OAGD2 in the list. (Assertion #5)
14. Verify that OAGD2 can see the details of at least one of those visible shared enterprise users’ business contexts but cannot make any change (verify with one created by other users and another one created by OAGD2 in the enterprise end user capacity). (Assertion #6)
15. Verify that BC0, BC1, BC2, and BC3 are in the list. (Assertion #7)
16. Verify that OAGD2 can see the details of BC0, BC1, BC2, and BC3. (Assertion #8)
17. OAGD2 deletes BC1 and BC2. (Assertion #9)
18. Verify that OAGD2 cannot change and cannot delete BC0. (Assertion #10)
19. OAGD2 edits BC3. (Assertion #12)
20. Verify that OAGD2 cannot share BC3. (Assertion #14)
21. OAGD2 logs out.
22. The admin developer logs in again.
23. The admin developer shares BC3.
24. OAGD2 logs in as an OAGi developer.
25. Verify that OAGD2 cannot make change and cannot delete BC3. (Assertion #11)

Test assertions covered by this test case:

1. The OAGi developer can create a business context.
2. Context categories and context schemes belonging to the OAGi tenant and those shared by enterprise tenants shall be selectable by OAGi developers for specifying context values and those private ones are not selectable.
3. The OAGi developer can edit a business context he created.
4. The OAGi developer cannot see in the business context list, enterprise users’ private business context.
5. The OAGi developer can see in the business context list, enterprise users’ shared business context.
6. The OAGi developer can see the details of enterprise user’s shared business context.
7. The OAGi developer can see in the list business context in any status created by others in the OAGi tenant.
8. The OAGi developer can see details of business context in any status created by others in the OAGi tenant.
9. The OAGi developer can delete business context created by others in the OAGi tenant.
10. The OAGi developer can no longer change or delete a shared business context created by OAGi admin.
11. The OAGi developer can no longer change or delete a shared business context created by OAGi developers.
12. The OAGi developer can edit a business context even if it is created by other OAGi developers.
13. The OAGi developer can edit a business context even if it is created by other OAGi admins.
14. The OAGi developer cannot share a business context.

## OAGi developer cannot delete business context used by a profile BOD

Pre-condition: At least a user, say OAGD1, with the OAGi developer role exists. There are profile BODs created by various OAGi tenant roles. One profile BOD uses a private business context, says BC1, owned by OAGD1 and another profile BOD uses a shared business context, says BC2, owned by OAGD1.

Test Step:

1. An OAGi developer, says OAGD1, logs into the system.
2. Verify that OAGD1 cannot delete BC1.
3. Verify that OAGD2 cannot delete BC2.
4. OAGD1 log out.

## OAGi developer authorized management of profile BODs

Pre-condition: At least two users, say OAGD1 and OAGD2, with OAGi developer role and enterprise end user role exists. Free profile BODs published by the OAGi tenant exist in the system. Private and shared OAGi tenant profile BODs exist in the system that are created by different OAGi developers and are in varying states. Private and shared enterprise tenants profile BODs exist in the system that are in varying states, some of which are owned by OAGD1 and OAGD2 in their enterprise end user capacities. There OAGi tenant’s published code lists. There are shared enterprise tenants’ code lists.

1. An OAGi admin developer, OAGADx, logs into the system.
2. The admin developer creates new profile BODs and made some customizations, says PB0, PB1, PB2. The admin developer moves to publish PB0, but still leave PB1 in the Editing state and PB2 in the Candidate state.
3. The admin developer shares PB0.
4. The admin developer logs out.
5. An OAGi developer, OAGD1, logs into the system.
6. OAGD1 creates new profile BODs with some customizations (here, make sure that the local and global extension functions are invoked in the customizations), says PB3, PB4, PB5, and PB6. OAGD1 advances PB3 to the Published state while PB4 and PB6 are left in the Editing state and PB5 is in the Candidate state. Verify that business contexts that are owned by OAGi tenant and that are shared by enterprise tenants are selectable. (Assertion #1, 2, 3, 4, 5)
7. Verify that OAGD1 cannot make any change to PB3[[5]](#footnote-5). (Assertion #7)
8. Verify that OAGD1 cannot make any change to PB5 except changing the state back to Editing or deleting it (note keep PB5 at the Editing state, just check that button to change the state back is enabled). (Assertion #6)
9. Verify that OAGD1 can successfully delete PB6. (Assertion #8)
10. Verify that OAGD1 cannot share PB0, PB1, and PB2. (Assertion #9)
11. Verify that OAGD1 cannot make PB0, PB1, and PB2 free. (Assertion #9)
12. Verify that OAGD1 cannot share PB3, PB4, and PB5. (Assertion #11)
13. Verify that OAGD1 cannot make PB3, PB4, and PB5 free. (Assertion #12)
14. OAGD1 logs out.
15. Another OAGi developer, says OAGD2, logs into the system.
16. OAGD2 views the list of profile BODs.
17. Verify that enterprise users’ private profile BODs are not visible to OAGD2 in the list including those owned by OAGD2 in the enterprise end user capacity. (Assertion #13)
18. Verify that enterprise users’ shared profile BODs are visible to OAGD2 in the list. (Assertion #14)
19. Verify that PB0, PB1, PB2, PB3, PB4, and PB5 are in the list. (Assertion #15)
20. Verify that OAGD2 can see the details of at least one of those visible enterprise users’ shared profile BODs but cannot make any change. Verify that OAGD2 cannot drill down to make change to the associated business context and context categories or see and edit the whole context scheme (the user should be able to see the context scheme values used in the business context). (Assertion #16)
21. Verify that OAGD2 can see the details of PB0 but cannot make any change (because it is published even though it is owned by the admin developer). (Assertion #17)
22. Verify that OAGD2 cannot see, edit the detail, nor change state and ownership of PB1 (which is still in the Editing state and owned by the admin developer). (Assertion #18)
23. Verify that OAGD2 can see (review) the detail of PB2 (which is in the Candidate state) but cannot make any change. (Assertion #19)
24. Verify that OAGD2 can see the details of PB3 but cannot make any change (because it is published even though it is owned by the OAGD1). (Assertion #20)
25. Verify that OAGD2 cannot see the detail nor making any change to PB4 (because it is still in the Editing state and is owned by another developer, OAGD1). (Assertion #21)
26. Verify that OAGD2 can see (review) the detail of PB5 (which is in the Candidate state) but cannot make any change. (Assertion #22)
27. OAGD2 logs out.
28. OAGD1 logs in.
29. OAGD1 opens PB4 for edit.
30. OAGD1 assigns a primitive restriction using built-in types. Verify that this works. (Assertion #23)
31. OAGD1 assigns a code list to a field for primitive restriction. Verify that published code lists belonging to the OAGi tenant and shared code lists belonging to enterprise tenants are available for selection. (Assertion #24)
32. Verify that OAGD1 can successfully change the ownership of PB4 to OAGD2. (Assertion #25)
33. Verify that OAGD1 cannot change the ownership of PB5. (Assertion #26)
34. Verify that OAGD1 can successfully change the state of PB5 back to the Editing state. (Assertion #6)
35. Verify that OAGD1 can successfully change the ownership of PB5 to OAGADx. (Assertion #27)
36. OAGD1 logs out.
37. OAGD2 logs in as an OAGi developer.
38. Verify that OAGD2 can edit (be sure to invoke the local & global extension functions) and change the state of PB4 to the Published state. (Assertion #3, 5)
39. OAGD2 logs out.
40. The admin developer, OAGADx, logs in.
41. OAGADx transfer the ownership of PB5 to OAGD2.
42. OAGADx logs out.
43. OAGD2 logs in.
44. OAGD2 goes to the Copy Profile BOD page.
45. Verify that OAGD2 cannot see and choose any enterprise users’ private profile BODs for copying including those owned by OAGD2 in the enterprise end user tenant capacity. (Assertion #28)
46. Verify that OAGD2 can see and choose enterprise tenant’s shared profile BODs for copying. (Assertion #29)
47. Verify that OAGD2 can see and choose PB0, PB2, PB3, and PB4 for copying. (Assertion #30)
48. Verify that OAGD2 cannot see and choose PB1 for copying (it is ok to fail this b/c this is a known bug in the version given to JT). (Assertion #31)
49. Verify that OAGD2 cannot see and choose PB5 for copying (it is ok to fail this b/c this is a known bug in the version given to JT). (Assertion #32)
50. OAGD2 goes to the expression generation page.
51. Verify that OAGD2 cannot see and choose any enterprise users’ private profile BODs for generating an expression including those owned by OAGD2 in the enterprise end user’s capacity. (Assertion #33)
52. Verify that OAGD2 can see and choose enterprise tenant’s shared profile BODs for generating an expression. (Assertion #34)
53. Verify that OAGD2 can see and choose PB0, PB2, PB3, and PB4 for generating an expression. (Assertion #35)
54. Verify that OAGD2 cannot see and choose PB1 for generating an expression (it is ok to fail this b/c this is a known bug in the version given to JT). (Assertion #36)
55. Verify that OAGD2 cannot see and choose PB5 for generating an expression (it is ok to fail this b/c this is a known bug in the version given to JT). (Assertion #37)
56. OAGD2 logs out.

Assertions covered in this test case:

1. An OAGi developer can create a profile BOD.
2. Business contexts belonging to the OAGi tenant and those shared by enterprise tenants shall be selectable by OAGi admins for associating with the profile BOD during its creation.
3. An OAGi developer can edit a profile BOD it owns.
4. An OAGi developer can change the state of a profile BOD he owns from Editing to Candidate.
5. An OAGi developer can change the state of a profile BOD he owns from Candidate to Published.
6. An OAGi developer cannot make any change to his own profile BOD in Candidate state except changing the state from Candidate back to Editing or deleting it.
7. An OAGi developer cannot make any change to an owned profile BOD that is in the Published state.
8. An OAGi developer can delete his own profile BOD that is in the Editing state.
9. An OAGi developer cannot share an unowned profile BOD in any state.
10. An OAGi developer cannot make an unowned profile BOD in any state free.
11. An OAGi developer cannot share an owned profile BOD in any state.
12. An OAGi developer cannot make an owned profile BOD in any state free.
13. An OAGi developer cannot see a list of profile BODs that includes enterprise users’ private profile BODs in any state.
14. An OAGi developer can see a list of profile BODs that includes enterprise users’ shared profile BODs.
15. An OAGi developer can see a list of profile BODs that includes other OAGi tenant’s profile BODs in any state.
16. An OAGi developer can see the details of enterprise users’ shared profile BODs including its business context but cannot make any change to the profile BODs and any associated information, and cannot drill down to see the whole context scheme values.
17. An OAGi developer can see the details of published profile BODs owned by an admin developer, but cannot make any change.
18. An OAGi developer cannot see the details of or make any change to profile BODs in the Editing state owned by an another OAGi admin developer.
19. An OAGi developer can see the details of the candidate profile BODs owned by an admin developer, but cannot make any change.
20. An OAGi developer can see the details of published profile BODs owned by an another OAGi developer, but cannot make any change.
21. An OAGi developer cannot see the details of or make any change to profile BODs in the Editing state owned by an another OAGi developer.
22. An OAGi developer can see the details of the candidate profile BODs owned by an another OAGi developer, but cannot make any change.
23. Built-in type primitive restriction works for the OAGi developer.
24. OAGi tenant’s published code lists and enterprise tenants’ shared code lists are available for primitive restriction using a code list.
25. An OAGi developer can transfer the ownership of his own profile BODs in Editing state to another OAGi developer.
26. An OAGi developer cannot change the ownership of his own profile BODs in Candidate state.
27. An OAGi developer can transfer the ownership of his own profile BODs in Editing state to an OAGi admin developer.
28. An OAGi developer cannot copy an enterprise tenant’s private profile BOD.
29. An OAGi developer can copy an enterprise tenant’s shared profile BOD.
30. An OAGi developer can copy profile BODs that belong to other OAGi developers and admin developers and that are in candidate or published states.
31. An OAGi developer cannot copy profile BODs that belong to an OAGi admin developer and that are in editing states.
32. An OAGi developer cannot copy profile BODs that belong to another OAGi developer and that are in editing states.
33. An OAGi developer cannot generate expressions of an enterprise users’ private profile BODs.
34. An OAGi developer can generate expressions of an enterprise users’ shared profile BODs.
35. An OAGi developer can generate expressions of profile BODs that belong to other OAGi developers and admin developers and that are in candidate or published states.
36. An OAGi developer cannot generate expressions of profile BODs that belong to another OAGi developer and that are in editing states.
37. An OAGi developer cannot generate expressions of profile BODs that belong to an OAGi admin developer and that are in editing states.

## OAGi developer authorized access to CC management functions with localized extension

Pre-condition: At least two users, say OAGD1 has both OAGi developer role and enterprise end user role exists and OAGD2 has both OAGi developer role and enterprise admin user role. OAGD1 and OAGD2 have created profile BOD extensions in their enterprise user capacity and the extensions are in various states.

Test Steps:

1. An OAGi admin developer, OAGADx, logs into the system.
2. OAGADx creates a new profile BOD, PB0.
3. OAGADx makes a localized extension to PB0. This results in, says, User Extension Group ACC, UEGACC0, and corresponding ASCCP0 and ASCC0 (these two are supposed to be hidden on the CC Management page by default logic).
4. OAGADx adds some ASCCs and BCCs to the extension, i.e., to UEGACC0.
5. OAGADx advances the PB0 extension to the Published state.
6. OAGADx repeats step 2 to 4 two times resulting PB1, PB2, and corresponding UEGACC1, ASCCP1, ASCC1, and UEGACC2, ASCCP2, ASCC2, respectively. Use a commonly used component like ApplicationArea’s Extension for PB1.
7. OAGADx advances PB2 extension to the Candidate state.
8. OAGADx logs out.
9. OAGD1 logs into the system as an OAGi developer.
10. OAGD1 creates new profile BODs, PB3, PB4, PB5, and PB6.
11. OAGD1 makes some localized extensions to PB3, PB4, PB5 and PB6 that results in UEGACC3, ASCCP3, ASCC3, UEGACC4, ASCCP4, ASCC4, UEGACC5, ASCCP5, ASCC5, and UEGACC6, ASCCP6, ASCC6. Some ASCCs and BCCs are added to these extensions. Use a commonly used component like Party’s Extension for PB5. Verify that no ASCCP belonging to enterprise tenants ever shows up for selection when adding an ASCC (use particularly the ASCCP that belongs to OAGD1 in the enterprise end user capacity in the verification). Also verify that user extension group ASCCPs are not available for selection. (Assertion #1, #2 #2.1, #2.2)
12. OAGD1 removes an ASCC from the PB3 extension. (Assertion #2.3)
13. OAGD1 removes a BCC from the PB3 extension. (Assertion #2.4)
14. OAGD1 advances the PB5 extension to the Candidate state. (Assertion #2.5)
15. OAGD1 advances the PB3 extension to the Published state. (Assertion #2.6)
16. Verify that OAGD1 cannot make any change to the PB5 extension except changing the state back to Editing (note keep the PB5 extension in the Candidate state after this step). (Assertion #2.7)
17. OAGD1 creates a profile BOD PB7. Makes an extension to PB7 to the same component as PB5. Here actually no user extension group created (so there is no UEGACC7) (Assertion #2.8)
18. OAGD1 opens PB4. Try to make an extension to the same component as PB1. Verify that the application does not allow this. (Assertion #2.9.1)
19. OAGD1 logs out.
20. OAGD2 logs in as an OAGi developer.
21. OAGD2 creates a profile BOD, PB8.
22. OAGD2 tries to make an extension in PB8 to the same component as PB5. Verify that the application does not allow this. (Assertion #2.9.2)
23. OAGD2 logs out.
24. OAGD1 logs in as an OAGi developer.
25. OAGD1 opens the CC Management page.
26. Verify that OAGD1 cannot see in the list any user extension group ACC belonging to OAGD1 in the enterprise end user capacity. (Assertion #3.1)
27. Verify that OAGD1 can see in the list UEGACC1. (Assertion #3.2.1)
28. OAGD1 logs out.
29. OAGD2 logs in as an OAGi developer.
30. OAGD2 goes to the CC Management page.
31. Verify that OAGD2 can see in the list UEGACC5. (Assertion #3.2.2)
32. Verify that OAGD2 can see the details of UEGACC2 but cannot make any change. (Assertion #3.3.1)
33. Verify that OAGD2 can see the details of UEGACC3 but cannot make any change. (Assertion #3.3.2)
34. Verify that OAGD2 cannot see the details of the UEGACC6. (Assertion #3.3.3)
35. OAGD2 logs out.
36. OAGD1 logs in as an OAGi developer and goes to the CC Management page.
37. OAGD1 open UEGACC4 to add a new ASCC. Verify that no ASCCP belonging to enterprise tenants ever shows up for selection when adding an ASCC (use particularly the ASCCP that belongs to OAGD1 in the enterprise end user capacity in the verification). Also, verify that user extension group ASCCPs are not available for selection. (Assertion #3.4)
38. OAGD1 adds a new BCC to UEGACC4. (Assertion #3.5)
39. OAGD1 removes an ASCC from UEGACC4. (Assertion #3.6)
40. OAGD1 removes a BCC from UEGACC4. (Assertion #3.7)
41. OAGD1 moves UEGACC4 to the Candidate state. (Assertion #3.8)
42. OAGD1 moves UEGACC4 back to the Editing state. (Assertion #3.9)
43. OAGD1 moves UEGACC4 to the Candidate and then Published state. (Assertion #3.10)
44. Verify that there is no sharing capability in the CC Management. (Assertion #4)
45. OAGD1 transfers the ownership of UEACC6 to OAGD2. Verify no user without an OAGi tenant role shows up for selection and verify that the transfer is successful. (Assertion #5.1)
46. Verify that OAGD1 cannot change the ownership of UEGACC5. (Assertion #5.2)
47. OAGD1 changes the state of UEGACC5 to the Editing state.
48. Verify that OAGD1 can successfully change the ownership of UEGACC5 to OAGADx. (Assertion #5.3)
49. Verify that transfer of ownership is not allow for the user extension group ACC he does not own. (Assertion #5.4)
50. OAGD1 logs out.
51. OAGD2 logs in as an OAGi developer.
52. OAGD2 change the state of UEGACC5 to the Published state.
53. OAGD2 logs out.
54. OAGD1 logs in as an OAGi developer.
55. OAGD1 opens PB7.
56. Verify that OAGD1 gets a notification to up take the extension when opening the PB7 for edit. (note this is because PB7 uses the same extension as PB5). (Assertion #6.1)
57. OAGD1 opens PB3.
58. Verify that OAGD1 gets a notification to up take the extension when opening the PB3 for edit. (Assertion #6.2)
59. OAGD1 logs out.

Assertions covered in this test case:

1. OAGi developer can creates a localized profile BOD extension.
2. OAGi developer can manage a localized profile BOD extension (through the localized profile BOD extension menu).
   1. An OAGi developer can add a new ASCC to an owned profile BOD extension in the Editing state using ASCCP belonging to the OAGi tenant but not ASCCP belonging to enterprise tenants in any state. Also, user extension group ASCCPs shall not be available for selection.
   2. An OAGi developer can add a new BCC to an owned profile BOD extension in the Editing state.
   3. An OAGi developer can remove an ASCC from an owned profile BOD extension in the Editing state.
   4. An OAGi developer can remove an BCC from an owned profile BOD extension in the Editing state.
   5. An OAGi developer can change the state of an owned profile BOD extension from Editing to Candidate.
   6. An OAGi developer can change the state of an owned profile BOD extension from Candidate to Published.
   7. An OAGi developer can change the state of an owned profile BOD extension from Candidate back to Editing.
   8. An OAGi developer can open up the extension to the same component that is still in the Editing state through another profile BOD.
   9. The system shall not allow an OAGi developer to create an extension to a component being extended by another OAGi tenant’s user.
      1. The system shall not allow an OAGi developer to create an extension to a component being extended by an OAGi admin in the Editing state.
      2. The system shall not allow an OAGi developer to create an extension to a component being extended by another OAGi developer in the Candidate state.
3. OAGi developer can manage user extension group ACCs on the CC Management page.
   1. An OAGi developer cannot see in the CC list any user extension group ACC belonging to enterprise tenants (test particularly on those ACCs owned by the OAGi developer in the enterprise tenant’s capacity).
   2. An OAGi developer can see in the CC list user extension group ACCs belonging to others in the OAGi tenant. The ACCs can be in any state.
      1. An OAGi developer can see in the list user extension group ACCs in the Editing state belonging to an OAGi admin.
      2. An OAGi developer can see in the list user extension group ACCs in the Candidate state belonging to another OAGi developer.
   3. An OAGi developer can see the detail of user extension group ACCs belonging to others in the OAGi tenant only in the Candidate or Published state but cannot make any change.
      1. An OAGi developer can see the detail of a user extension group ACC belonging to another OAGi admin in the Candidate state but cannot make any change.
      2. An OAGi developer can see the detail of a user extension group ACC belonging to another OAGi developer in the Published state but cannot make any change.
      3. An OAGi developer cannot see the detail of a user extension group ACC belonging to another OAGi developer in the Editing state.
   4. An OAGi admin can add a new ASCC to an owned ACC in the Editing state using ASCCP belonging to the OAGi tenant but not ASCCP belonging to enterprise tenants in any state. User extension group ASCCPs shall not be available for selection.
   5. An OAGi admin can add a new BCC to an owned ACC in the Editing state.
   6. An OAGi admin can remove an ASCC from an owned ACC in the Editing state.
   7. An OAGi admin can remove an BCC from an owned ACC in the Editing state.
   8. An OAGi admin can change the state of an ACC it owns from Editing to Candidate.
   9. An OAGi admin can change the state of an ACC it owns from Candidate back to Editing.
   10. An OAGi admin can change the state of an ACC it owns from Candidate to Published.
4. There is no sharing capability in the CC Management.
5. An OAGi developer can transfer the ownership of the user extension group ACC he owns. The ACC has to be in the Editing state, and the new owner can must have an OAGi tenant role.
   1. An OAGi developer owner of a user extension group ACC in the Editing state can transfer its ownership to another OAGi developer and not a user without an OAGi tenant role.
   2. An OAGi developer owner of the user extension group ACC in the Candidate state can’t transfer its ownership.
   3. An OAGi developer owner of a user extension group ACC in the Editing state can transfer its ownership to an OAGi admin.
   4. OAGi developer can’t change the ownership of the user extension group ACC belonging to another user in the OAGi or enterprise tenant.
6. The system shall make a notification to the user while opening up a profile BOD, to which a related user extension group ACC has been published.
   1. The system shall notify the OAGi admin when another OAGi developer published a user extension group ACC relevant to his profile BOD.
   2. The system shall notify the OAGi admin while opening a profile BOD when his user extension group ACC relevant to the profile BOD has been published.

## OAGi developer authorized access to CC management functions with the global extension

Repeat Test Case 9.8 but with the global extension.

## OAGi developer authorized access to code list management functions.

Pre-condition: A user, say OAGD1, with OAGi developer role and enterprise end user role exists. Another user, say OAGD2, with OAGi developer role and enterprise admin user role exists. OAGi tenant code lists exist in the system that are created by different OAGi developers and are in varying states. Private and shared enterprise tenants’ code lists exist in the system that are in varying states, some of which are created by OAGD1 and OAGD2 in their enterprise end user capacities.

Test steps:

1. An OAGi admin developer, OAGADx, logs into the system.
2. The admin developer creates new code list and made some customizations, says CL0, CL1, CL2. Make CL0 extensible. The admin developer moves to publish CL0, but still leave CL1 and CL2 in the Editing state.
3. The admin developer logs out.
4. An OAGi developer, OAGD1, logs into the system.
5. OAGD1 creates new code lists with some values, says CL3, CL4, CL5, and CL6. OAGD1 advances CL3 to the Published state while CL4, CL5, and CL6 are left in the Editing state. Make CL3 extends CL0, verify that this is possible. Make CL4 extends an enterprise tenant’s shared code list, verify that this is possible. Also, verify that extensible, private enterprise tenants’ code list cannot be used as a base (used particularly the enterprise tenant in which OAGD1 is a member to verify). (Assertion #1, 2, 3)
6. Verify that OAGD1 cannot make any change to CL3. (Assertion #4)
7. Verify that OAGD1 can successfully delete CL6. (Assertion #8)
8. Verify that OAGD1 cannot share CL0, CL1, and CL2. (Assertion #9)
9. Verify that OAGD1 cannot make CL0, CL1, and CL2 free. (Assertion #9)
10. Verify that OAGD1 cannot share CL3, CL4, and CL5. (Assertion #9)
11. Verify that OAGD1 cannot make CL3, CL4, and CL5 free. (Assertion #9)
12. OAGD1 logs out.
13. Another OAGi developer, says OAGD2, logs into the system.
14. OAGD2 views the list of code lists.
15. Verify that enterprise users’ private code lists are not visible to OAGD2 in the list including those owned by OAGD2 in the enterprise end user capacity. (Assertion #8)
16. Verify that enterprise users’ shared code lists are visible to OAGD2 in the list. (Assertion #9)
17. Verify that CL0, CL1, CL2, CL3, CL4, and CL5 are in the list. (Assertion #10)
18. Verify that OAGD2 can see the details of at least one of those visible enterprise users’ shared code lists but cannot make any change. (Assertion #11)
19. Verify that OAGD2 can see the details of CL0 but cannot make any change (because it is published even though it is created by the admin developer). (Assertion #12)
20. Verify that OAGD2 can edit CL1. (Assertion #13)
21. Verify that OAGD2 can see the details of CL3 but cannot make any change. (Assertion #14)
22. Verify that OAGD2 can edit CL4. (Assertion #15)
23. OAGD2 logs out.

Assertions covered in this test case:

1. An OAGi developer can create a code list.
2. An OAGi developer can edit a code list he created.
3. An OAGi developer can change the state of a code list he created from Edting to Published.
4. Published, extensible OAGi tenant’s code list can be used as a based code list by an OAGi developer.
5. Shared, extensible enterprise tenants’ code lists can be used as a based code list by an OAGi developer.
6. Private enterprise tenants’ code lists, even though extensible, cannot be used as a based code list by an OAGi developer.
7. An OAGi developer cannot make any change to a code list he created that is in the Published state.
8. An OAGi developer can delete the code list he created that is in the Editing state.
9. OAGi developer (OAGi tenant in general) cannot share code list.
10. OAGi developer (OAGi tenant in general) cannot make code list free.
11. An OAGi developer cannot see a list of code lists that includes enterprise users’ private code lists in any state.
12. An OAGi developer can see a list of code lists that includes enterprise users’ shared code lists.
13. An OAGi developer can see a list of code lists that includes other OAGi tenant’s code lists in any state.
14. An OAGi developer can see the details of enterprise users’ shared code lists.
15. An OAGi developer can see the details of published code lists created by an admin developer, but cannot make any change.
16. An OAGi developer can edit code lists in the Editing state even though it is created by an another OAGi admin developer.
17. An OAGi developer can see the details of published code lists created by an another OAGi developer, but cannot make any change.
18. An OAGi developer can edit code lists in the Editing even though it was created by an another OAGi developer.

# Ensure that there is no conflict across tenants related to shared resources

## Multiple tenants can make extensions to the same component without mixing up their extensions.

Pre-condition: OAGD1 user exists and has an OAGi developer role. An enterprise tenant A exists with an enterprise end user ENTA. An enterprise tenant B exists with an enterprise admin user ENTB.

Test Steps:

1. A user, OAGD1, logs in as an OAGi developer.
2. OAGD1 creates a profile BOD, PBOAG.
3. OAGD1 creates a global extension.
4. OAGD1 adds a BCC, OAGBCC, to the global extension.
5. OAGD1 logs out.
6. A user, ENTA, logs in as an enterprise A’s end user.
7. ENTA creates a profile BOD, PBENTA.
8. Verify that ENTA can create the global extension and add an ASCC. (Assertion #1)
9. Verify that OAGBCC does not appear in the global extension created by ENTA. (Assertion #1)
10. ENTA creates a local extension to the Extension in Application Area.
11. ENTA adds a BCC to the Application Area extension, ENTABCC.
12. ENTA logs out.
13. A user, ENTB, logs in as an enterprise B’s admin user.
14. ENTB creates a profile BOD, PBENTB.
15. Verify that ENTB can create the local extension to the Application Area and add an ASCC. (Assertion #2).
16. Verify that ENTABCC does not appear in the Application Area extension.

Test assertions covered by this test case:

1. Multiple tenants can make global extensions to the same component without mixing up their extensions.
2. Multiple tenants can make local extensions to the same component without mixing up their extensions.

# OAGi admin developer access right to SRT core functions

Requirement document reference: Section 3.2.3, 3.4.3.2

## OAGi admin developer’s authorized management of context categories

Pre-condition: At least two OAGi admins, say OAGADx and OAGADy, with both OAGi admin and enterprise end user roles exist. There are context categories created by various enterprise users, some are created by OAGDADx and OAGADy in the enterprise end user capacity, and some of which have been shared. There are at least two OAGi developers in the system.

Test Step:

1. An OAGi admin developer, OAGADx, logs into the system.
2. OAGADx creates new context categories, says CAT0, CAT1, and CAT4. (Assertion #1)
3. OAGADx edits CAT1. (Assertion #2)
4. The admin developer logs out.
5. An OAGi developer, says OAGD1, logs into the system.
6. OAGD1 creates two new context categories, says CAT2 and CAT3.
7. OAGD1 log out.
8. Another OAGi admin, says OAGADy, logs into the system.
9. OAGADy views the list of context categories.
10. Verify that enterprise users’ context categories, which have not been shared, are not visible to OAGADy in the list. Make sure that the verification statements include context categories that are created by OAGADx and OAGADy in the enterprise end user capacity. (Assertion #3)
11. Verify that enterprise users’ context categories, which have been shared, are visible to OAGADy in the list. (Assertion #4)
12. Verify that OAGADy can see the details of those visible shared enterprise users’ context categories but cannot make any change (verify with one created by other users and another one created by OAGADy in the enterprise end user capacity). (Assertion #5)
13. Verify that CAT0, CAT1, CAT2, and CAT3 are in the list. (Assertion #6)
14. Verify that OAGADy can see the details of CAT0, CAT1, CAT2, and CAT3. (Assertion #7)
15. OAGADy deletes CAT1 and CAT2. (Assertion #8)
16. OAGADy edits CAT3. (Assertion #9)
17. OAGADy edits CAT4. (Assertion #10)
18. OAGADy shares CAT0, CAT3, CAT4. (Assertion #11)
19. OAGADy creates CAT5.
20. OAGADy logs out.
21. OAGADx logs in again as an OAGi admin.
22. OAGADx shares CAT5. (Assertion #11)
23. OAGADx logs out.
24. OAGADy logs in as an OAGi amin.
25. Verify that OAGADy cannot edit or delete CAT3. (Assertion #12)
26. Verify that OAGADy cannot make change or delete CAT4 and CAT5. (Assertion #13)

Test Assertions covered by this test case:

1. The OAGi admin can create a context category.
2. The OAGi admin can edit a context category he created.
3. The OAGi admin cannot see in the context category list, enterprise users’ private context categories.
4. The OAGi admin can see in the context category list, enterprise users’ private context categories.
5. The OAGi admin can see the details of enterprise user’s shared context categories.
6. The OAGi admin can see in the list context categories in any status created by others in the OAGi tenant
7. The OAGi admin can see details of context categories in any status created by others in the OAGi tenant.
8. The OAGi admin can delete context categories created by others in the OAGi tenant.
9. The OAGi admin can edit a context category even if it is created by another OAGi developer.
10. The OAGi admin can edit a context category even if it is created by another OAGi admin.
11. The OAGi admin can share a context category created by himself or others in the OAGi tenant.
12. The OAGi admin can no longer change or delete a shared context category created by other OAGi developers.
13. The OAGi admin can no longer change or delete a shared context category created by other OAGi admins.

## OAGi admin developer’s authorized management of context schemes

Pre-condition: At least two OAGi admins, say OAGADx and OAGADy, with both OAGi admin and enterprise end user roles exist. There are context categories created by various enterprise users, some are created by OAGDADx and OAGADy in the enterprise end user capacity, and some of which have been shared. There are at least two OAGi developers in the system.

Test Step:

1. An OAGi admin developer, OAGADx, logs into the system.
2. The admin developer creates new context schemes (with some values), says CS0, CS1, and CS4. Verify that both context categories that are owned by OAGi tenant and that are shared by enterprise tenants are selectable for associating with the context schemes during the creation. Verify that enterprise tenants’ private context categories are not selectable. (Assertion #1, #2)
3. OAGADx edits CS1. (Assertion #3)
4. The admin developer logs out.
5. A user, says OAGD1, logs into the system as an OAGi developer.
6. OAGD1 creates two new context schemes, says CS2 and CS3.
7. OAGD1 logs out.
8. Another user, OAGADy, logs into the system as an OAGi admin.
9. OAGDy views the list of context schemes.
10. Verify that enterprise users’ context schemes, which have not been shared, are not visible to OAGADy in the list. Make sure that the verification statements include context schemes that are created by OAGADx and OAGADy in the enterprise end user capacity. (Assertion #4)
11. Verify that enterprise users’ context schemes, which have been shared, are visible to OAGADy in the list. (Assertion #5)
12. Verify that OAGADy can see the details of those visible shared enterprise users’ context schemes but cannot make any change (verify with one created by other users and another one created by OAGADy in the enterprise end user capacity). (Assertion #6)
13. Verify that CS0, CS1, CS2, and CS3 are in the list. (Assertion #7)
14. Verify that OAGADy can see the details of CS0, CS1, CS2, and CS3. (Assertion #8)
15. OAGADy deletes CS1 and CS2. (Assertion #9)
16. OAGADy edits CS3. (Assertion #10)
17. OAGADy edits CS4. (Assertion #11)
18. OAGADy shares CS0, CS3, CS4. (Assertion #12)
19. OAGADy creates CS5.
20. OAGADy shares CS5. (Assertion #12)
21. OAGADy logs out.
22. OAGADx logs in as an OAGi admin.
23. Verify that OAGADx cannot edit or delete CS3. (Assertion #13)
24. Verify that OAGADx cannot make change or delete CS4 and CS5. (Assertion #14)
25. Verify that OAGDADx cannot delete CS0. (Assertion #14)

Test assertions covered by this test case:

1. The OAGi admin developer can create a context scheme.
2. Context categories belonging to the OAGi tenant and those shared by enterprise tenants shall be selectable by OAGi admins for association with the context schemes and those private ones are not selectable.
3. The OAGi admin developer can edit a context scheme he created.
4. The OAGi admin developer cannot see in the context scheme list, enterprise users’ private context schemes.
5. The OAGi admin developer can see in the context scheme list, enterprise users’ shared context schemes.
6. The OAGi admin developer can see the details of enterprise user’s shared context schemes.
7. The OAGi admin developer can see in the list context schemes in any status created by others in the OAGi tenant.
8. The OAGi admin developer can see details of context schemes in any status created by others in the OAGi tenant.
9. The OAGi admin developer can delete context schemes created by others in the OAGi tenant.
10. The OAGi admin developer can edit a context scheme even if it is created by other OAGi developers.
11. The OAGi admin developer can edit a context scheme even if it is created by other OAGi admins.
12. The OAGi admin developer can share a context scheme created by himself or others in the OAGi tenant.
13. The OAGi admin developer can no longer change or delete a shared context schemes created by OAGi developers.
14. The OAGi admin developer can no longer change or delete a shared context schemes created by another OAGi admin.

## OAGi admin developer cannot delete context categories used by a context scheme

Pre-condition: At least a user, say OAGADx, with the OAGi admin developer role exists. There are private context schemes created by various OAGi tenant roles. One context scheme uses a private context category, says CAT1, owned by OAGD1 and another context scheme uses a shared context category, says CAT2, owned by OAGADx.

Test Step:

1. OAGADx logs into the system as an OAGi admin.
2. Verify that OAGADx cannot delete CAT1.
3. Verify that OAGADx cannot delete CAT2.
4. OAGADx log out.

## OAGi admin developer cannot delete context schemes used by a business context

Pre-condition: At least a user, say OAGADx, with the OAGi admin developer role exists. There are business contexts created by various OAGi tenant roles. One business context, says BC1, uses a private context scheme, CS1, owned by OAGADx and another business context uses a shared context scheme, says CS2, owned by OAGADx.

Test Step:

1. OAGADx, logs into the system.
2. Verify that OAGADx cannot delete CS1.
3. Verify that OAGADx cannot delete CS2.
4. OAGADx log out.

## OAGi developer authorized management of business contexts

Pre-condition: At least two OAGi admins, say OAGADx and OAGADy, with both OAGi admin and enterprise end user roles exist. There are context categories created by various enterprise users, some are created by OAGDADx and OAGADy in the enterprise end user capacity, and some of which have been shared. There are at least two OAGi developers in the system.

Test Step:

1. An OAGi admin developer, OAGADx, logs into the system.
2. The admin developer creates new business contexts (with some context scheme values), says BC0, BC1, and BC4. Verify that context categories and context schemes that are owned by OAGi tenant and that are shared by enterprise tenants are selectable for context values while creating the business contexts. Verify that enterprise tenants’ private context categories and context schemes are not selectable. (Assertion #1, #2)
3. OAGADx edits BC1. (Assertion #3)
4. The admin developer logs out.
5. An OAGi developer, says OAGD1, logs into the system.
6. OAGD1 creates two new business contexts, says BC2 and BC3.
7. OAGD1 logs out.
8. Another user, OAGADy, logs into the system as an OAGi admin.
9. OAGDy views the list of business contexts.
10. Verify that enterprise users’ business contexts, which have not been shared, are not visible to OAGADy in the list. Make sure that the verification statements include business context that are created by OAGADx and OAGADy in the enterprise end user capacity. (Assertion #4)
11. Verify that enterprise users’ business contexts, which have been shared, are visible to OAGADy in the list. (Assertion #5)
12. Verify that OAGADy can see the details of at least one of those visible shared enterprise users’ business contexts but cannot make any change (verify with one created by other users and another one created by OAGADy in the enterprise end user capacity). (Assertion #6)
13. Verify that BC0, BC1, BC2, and BC3 are in the list. (Assertion #7)
14. Verify that OAGADy can see the details of BC0, BC1, BC2, and BC3. (Assertion #8)
15. OAGADy deletes BC1 and BC2. (Assertion #9)
16. OAGADy edits BC3. (Assertion #10)
17. OAGADy edits BC4. (Assertion #11)
18. OAGADy shares BC0, BC3, BC4. (Assertion #12)
19. OAGADy creates BC5.
20. OAGADy shares CS5. (Assertion #12)
21. OAGADy logs out.
22. OAGADx logs in as an OAGi admin.
23. Verify that OAGADx cannot make change and cannot delete BC3. (Assertion #13)
24. Verify that OAGADx cannot make change or delete BC4 and BC5. (Assertion #14)
25. Verify that OAGADx cannot change and cannot delete BC0. (Assertion #14)

Test assertions covered by this test case:

1. The OAGi admin developer can create a business context.
2. Context categories and context schemes belonging to the OAGi tenant and those shared by enterprise tenants shall be selectable by OAGi developers for specifying context values, and those private ones are not selectable.
3. The OAGi admin developer can edit a business context he created.
4. The OAGi admin developer cannot see in the business context list, enterprise users’ private business context.
5. The OAGi admin developer can see in the business context list, enterprise users’ shared business context.
6. The OAGi admin developer can see the details of enterprise user’s shared business context.
7. The OAGi admin developer can see in the list business context in any status created by others in the OAGi tenant.
8. The OAGi admin developer can see details of business context in any status created by others in the OAGi tenant.
9. The OAGi admin developer can delete business context created by others in the OAGi tenant.
10. The OAGi admin developer can edit a business context even if it is created by other OAGi developers.
11. The OAGi admin developer can edit a business context even if it is created by other OAGi admins.
12. The OAGi admin can share a business context created by himself or others in the OAGi tenant.
13. The OAGi admin developer can no longer change or delete a shared business context created by OAGi developers.
14. The OAGi admin developer can no longer change or delete a shared business context created by other OAGi admin.

## OAGi developer cannot delete business context used by a profile BOD

Pre-condition: At least a user, say OAGADx, with the OAGi developer role exists. There are profile BODs created by various OAGi tenant roles. One profile BOD uses a private business context, says BC1, owned by OAGADx and another profile BOD uses a shared business context, says BC2, owned by OAGADx.

Test Step:

1. OAGADx, logs into the system.
2. Verify that OAGADx cannot delete BC1.
3. Verify that OAGADx cannot delete BC2.
4. OAGADx log out.

## OAGi admin developer authorized management of profile BODs

Pre-condition: At least two OAGi admins, say OAGADx and OAGADy, with OAGi admin role and enterprise admin user role exists. Free profile BODs published by the OAGi tenant exist in the system. Private and shared OAGi tenant profile BODs exist in the system that are created by different OAGi developers and are in varying states. Private and shared enterprise tenants profile BODs exist in the system that are in varying states, some of which are owned by OAGADx and OAGADy in their enterprise admin user capacities. There OAGi tenant’s published code lists. There are shared enterprise tenants’ code lists.

1. An OAGi admin developer, OAGADx, logs into the system.
2. The admin developer creates new profile BODs and made some customizations, says PB0, PB1, and PB2. Verify that both business contexts that are owned by OAGi tenant and that are shared by enterprise tenants are selectable for associating with the profile BODs during the creation. Verify that enterprise tenants’ private business contexts are not selectable. (Assertion #1, #2)
3. OAGADx edits PB1. (Assertion #3)
4. OAGADx advances PB1 to the Candidate state. (Assertion #4)
5. Verify that OAGADx cannot edit PB1. (Assertion #5)
6. OAGADx moves PB1 back to the Editing state. (Assertion #6)
7. OAGADx deletes PB1. (Assertion #7)
8. OAGADx advances PB0 to the Candidate and then Published state. (Assertion #8)
9. Verify that OAGADx cannot edit or delete PB0. (Assertion #9)
10. OAGADx deletes PB2. (Assertion #10)
11. OAGADx creates PB4 and leave it in the Editing state, PB5 and advances it to the Candidate state, and PB6, PB7, and PB8 then advances them to the Published state.
12. OAGADx shares PB0. (Assertion #11)
13. Verify that OAGADx cannot share PB4 and PB5. (Assertion #11)
14. OAGADx makes PB0 free (can make free after shared). (Assertion #12)
15. OAGADx makes PB6 free (can make free even before sharing). (Assertion #12)
16. OAGADx logs out.
17. OAGADy logs in.
18. OAGADy shares PB7. (Assertion #13)
19. OAGADy cannot share PB4 and PB5. (Assertion #13)
20. OAGADy can make free PB7. (Assertion #14)
21. OAGADy cannot make free PB4 and PB5. (Assertion #14)
22. OAGADy goes to the profile BOD list page.
23. Verify that enterprise users’ private profile BODs are not visible to OAGADy in the list including those owned by OAGADy in the enterprise admin user capacity. (Assertion #15)
24. Verify that enterprise users’ shared profile BODs are visible to OAGADy in the list. (Assertion #16)
25. Verify that PB0, PB3, PB4, PB5, PB6, PB7 are in the list. (Assertion #17)
26. Verify that OAGADy can see the details of at least one of those visible enterprise users’ shared profile BODs but cannot make any change. Verify that OAGADy cannot drill down to make change to the associated business context and context categories or see and edit the whole context scheme (the user should be able to see the context scheme values used in the business context). (Assertion #18)
27. Verify that OAGADy can see the details of PB8 but cannot make any change (because it is published even though it is owned by the admin developer). (Assertion #19)
28. Verify that OAGDADy cannot see, edit the detail, nor change state and ownership of PB4 (which is still in the Editing state and owned by the admin developer). (Assertion #20)
29. Verify that OAGADy can see the detail of PB5 (which is in the Candidate state) but cannot make any change. (Assertion #21)
30. OAGADy logs out.
31. A user, OAGD1, logs in as an OAGi developer.
32. OAGD1 creates PB9, PB10, and PB11 and move PB10 to the Candidate state and PB11 to the Published state.
33. OAGD1 logs out.
34. OAGADy logs in as an OAGi admin.
35. Verify that OAGADy can see the details of PB11 but cannot make any change (because it is published even though it is owned by the OAGADx). (Assertion #22)
36. Verify that OAGADy cannot see the detail nor making any change to PB9 (because it is still in the Editing state and is owned by another admin developer, OAGADx). (Assertion #23)
37. Verify that OAGADy can see (review) the detail of PB10 (which is in the Candidate state) but cannot make any change. (Assertion #24)
38. OAGADy logs out.
39. OAGADx logs in as an OAGi admin.
40. OAGADx opens PB4 for editing.
41. OAGADx assigns a primitive restriction using built-in types. Verify that this works. (Assertion #24)
42. OAGADx assigns a code list to a field for primitive restriction. Verify that published code lists belonging to the OAGi tenant and shared code lists belonging to enterprise tenants are available for selection. (Assertion #25)
43. Verify that OAGADx can successfully change the ownership of PB4 to OAGD1. (Assertion #27)
44. Verify that OAGADx can successfully change the ownership of PB5 to OAGADy. (Assertion #28)
45. OAGADx logs out.
46. OAGD1 logs in as an OAGi developer.
47. OAGD1 transfer the ownership of PB4 to OAGADx.
48. OAGD1 logs out.
49. OAGADx logs in as an OAGi admin.
50. Verify that OAGADx can successfully transfer the ownership of PB4 to OAGADy. (Assertion #29)
51. Verify that OAGADx can successfully transfer the ownership of PB9 to OAGADy. (Assertion #30.1)
52. Verify that OAGADx can successfully transfer the ownership of PB5 to OAGD1. (Assertion #30.2)
53. OAGADx goes to the copy profile BOD page which brings up the list or profile BODs allowed for copying.
54. OAGADx logs out.
55. OAGD1 logs in as an OAGi developer.
56. OAGD1 transfers the ownership of PB5 to OAGADx.
57. OAGD1 logs out.
58. OAGADy logs in as an OAGi admin.
59. Verify that enterprise users’ private profile BODs are not visible to OAGADy in the list including those owned by OAGADy in the enterprise admin user capacity. (Assertion #31)
60. Verify that enterprise users’ shared profile BODs are visible to OAGADy in the list. (Assertion #32)
61. Verify that OAGADy can see and choose in the list PB5, PB8, PB10, and PB11 for copying. (Assertion #33)
62. Verify that OAGADy can see and choose in the list PB0, PB6, and PB7 for copying. (Assertion #34)
63. OAGADy transfers the ownership of PB9 to OAGD1 and PB4 to OAGADx.
64. Verify that OAGADy cannot see and choose PB9 for copying (it is ok to fail this b/c this is a known bug in the version given to JT). (Assertion #35)
65. Verify that OAGADy cannot see and choose PB4 for copying (it is ok to fail this b/c this is a known bug in the version given to JT). (Assertion #36)
66. OAGADy goes to the expression generation page.
67. Verify that OAGADy cannot see and choose any enterprise users’ private profile BODs for generating an expression including those owned by OAGADy in the enterprise admin user’s capacity. (Assertion #37)
68. Verify that OAGADy can see and choose enterprise tenant’s shared profile BODs for generating an expression. (Assertion #38)
69. Verify that OAGADy can see and choose PB5, PB8, PB10, and PB11 for generating an expression. (Assertion #39)
70. Verify that OAGADy can see and choose in the list PB0, PB6, and PB7 for generating an expression. (Assertion #40)
71. Verify that OAGADy cannot see and choose PB9 for generating an expression (it is ok to fail this b/c this is a known bug in the version given to JT). (Assertion #41)
72. Verify that OAGADy cannot see and choose PB4 for generating an expression (it is ok to fail this b/c this is a known bug in the version given to JT). (Assertion #42)
73. OAGADy logs out.

Assertions covered in this test case:

1. An OAGi admin can create a profile BOD.
2. Business contexts belonging to the OAGi tenant and those shared by enterprise tenants shall be selectable by OAGi admins for associating with the profile BOD during its creation.
3. An OAGi admin can edit a profile BOD it owns.
4. An OAGi admin can change the state of a profile BOD he owns from Editing to Candidate.
5. An OAGi admin cannot edit his own profile BOD in Candidate state.
6. An OAGi admin can change the state of his own profile BOD from Candidate back to Editing.
7. An OAGi admin can delete his own profile BOD in the Candidate state.
8. An OAGi admin can change the state of a profile BOD he owns from Candidate to Published.
9. An OAGi admin cannot make any change to an owned profile BOD that is in the Published state.
10. An OAGi admin can delete his own profile BOD that is in the Editing state.
11. An OAGi admin can share an owned profile BOD ONLY in the Published state.
12. An OAGi admin can make an owned profile BOD ONLY in the Published state free.
13. An OAGi admin can share an unowned profile BOD ONLY in the Published state.
14. An OAGi admin can make an unowned profile BOD ONLY in the Published state free.
15. An OAGi admin cannot see, in the list of profile BODs, enterprise users’ private profile BODs in any state including those owned by him in enterprise tenant roles.
16. An OAGi admin can see, in the list of profile BODs, enterprise users’ shared profile BODs.
17. An OAGi admin can see, in the list of profile BODs, OAGi tenant’s profile BODs in any state.
18. An OAGi admin can see the details of enterprise users’ shared profile BODs including its business context but cannot make any change to the profile BODs and any associated information, and cannot drill down to see the whole context scheme values.
19. An OAGi admin can see the details of published profile BODs owned by another admin developer, but cannot make any change.
20. An OAGi admin cannot see the details of or make any change to profile BODs in the Editing state owned by an another OAGi admin developer.
21. An OAGi admin can see the details of the candidate profile BODs owned by an admin developer, but cannot make any change.
22. An OAGi admin can see the details of published profile BODs owned by an another OAGi developer, but cannot make any change.
23. An OAGi admin cannot see the details of or make any change to profile BODs in the Editing state owned by an another OAGi developer.
24. Built-in type primitive restriction works for the OAGi admin.
25. OAGi tenant’s published code lists and enterprise tenants’ shared code lists are available for primitive restriction using a code list.
26. An OAGi admin can see the details of the candidate profile BODs owned by an another OAGi developer, but cannot make any change.
27. An OAGi admin can transfer the ownership of his own profile BODs in Editing state to another OAGi developer.
28. An OAGi admin can transfer the ownership of his own profile BODs in Candidate state to another OAGi admin.
29. An OAGi admin can transfer the ownership of his own profile BODs in Editing state to an OAGi admin developer.
30. An OAGi admin can transfer the ownership of a profile BOD in the Editing or Candidate state belonging to another user in the OAGi tenant to another user in the OAGi tenant.
    1. An OAGi admin can transfer the ownership of a profile BOD in the Editing state belonging to an OAGi developer to an OAGi admin.
    2. An OAGi admin can transfer the ownership of a profile BOD in the Candidate state belonging to another OAGi admin to an OAGi developer.
31. An OAGi admin cannot copy an enterprise tenant’s private profile BOD, particularly even the one he owns in an enterprise tenant role.
32. An OAGi admin can copy an enterprise tenant’s shared profile BOD.
33. An OAGi admin can copy profile BODs that belong to other OAGi developers and admin developers and that are in candidate or published states.
34. An OAGi admin can copy free profile BODs.
35. An OAGi admin cannot copy profile BODs that belong to another OAGi developer and that are in editing states.
36. An OAGi admin cannot copy profile BODs that belong to an OAGi admin developer and that are in editing states.
37. An OAGi admin cannot generate expressions of an enterprise users’ private profile BODs.
38. An OAGi admin can generate expressions of an enterprise users’ shared profile BODs.
39. An OAGi admin can generate expressions of profile BODs that belong to other OAGi developers and admin developers and that are in candidate or published states.
40. An OAGi admin can generate expressions from free profile BODs.
41. An OAGi admin cannot generate expressions of profile BODs that belong to another OAGi developer and that are in editing states.
42. An OAGi admin cannot generate expressions of profile BODs that belong to an OAGi admin developer and that are in editing states.

## OAGi admin authorized access to CC management functions with localized extension

Pre-condition: At least two OAGi admins, say OAGADx and OAGADy, with OAGi admin role and enterprise end user role exists. OAGADx and OAGADy have created profile BOD extensions in their enterprise end user capacity and the extensions are in various states.

Test Steps:

1. An OAGi developer, OAGD1, logs into the system.
2. OAGD1 creates a new profile BOD, PB0.
3. OAGD1 makes a localized extension to PB0. This results in, says, User Extension Group ACC, UEGACC0, and corresponding ASCCP0 and ASCC0 (these two are supposed to be hidden on the CC Management page by default logic).
4. OAGD1 adds some ASCCs and BCCs to the extension, i.e., to UEGACC0.
5. OAGD1 advances the PB0 extension to the Published state.
6. OAGD1 repeats step 2 to 4 two times resulting PB1, PB2, and corresponding UEGACC1, ASCCP1, ASCC1, and UEGACC2, ASCCP2, ASCC2, respectively. Use a commonly used component like ApplicationArea’s Extension for PB1.
7. OAGD1 advances UEGACC2 to the Candidate state.
8. OAGD1 logs out.
9. OAGADx logs into the system as an OAGi developer.
10. OAGADx creates new profile BODs, PB3, PB4, PB5, and PB6.
11. OAGADx makes some localized extensions to PB3, PB4, PB5 and PB6 that results in UEGACC3, ASCCP3, ASCC3, UEGACC4, ASCCP4, ASCC4, UEGACC5, ASCCP5, ASCC5, and UEGACC6, ASCCP6, ASCC6. Some ASCCs and BCCs are added to these extensions. Use a commonly used component like Party’s Extension for PB5. Verify that no ASCCP belonging to enterprise tenants ever shows up for selection when adding an ASCC (use particularly the ASCCP that belongs to OAGADx in the enterprise end user capacity in the verification). Also, verify that user extension group ASCCPs are not available for selection. (Assertion #1, #2.1, #2.2)
12. OAGADx removes an ASCC from the PB3 extension. (Assertion #2.3)
13. OAGADx removes a BCC from the PB3 extension. (Assertion #2.4)
14. OAGDDx advances the PB5 extension to the Candidate state. (Assertion #2.5)
15. OAGADx advances the PB3 extension to the Published state. (Assertion #2.6)
16. Verify that OAGADx cannot make any change to the PB5 extension except changing the state back to Editing (note keep the PB5 extension in the Candidate state after this step). (Assertion #2.7)
17. OAGADx creates a profile BOD PB7. Makes an extension to PB7 to the same component as PB5. Here actually no user extension group created (so there is no UEGACC7) (Assertion #2.8)
18. OAGADx opens PB4. Try to make an extension to the same component as PB1. Verify that the application does not allow this. (Assertion #2.9.1)
19. OAGADx logs out.
20. OAGADy logs in as an OAGi admin.
21. OAGADy creates a profile BOD, PB8.
22. OAGADy tries to make an extension in PB8 to the same component as PB5. Verify that the application does not allow this. (Assertion #2.9.2)
23. OAGDAy logs out.
24. OAGADx logs in as an OAGi admin.
25. OAGADx opens the CC Management page.
26. Verify that OAGADx cannot see in the list any user extension group ACC belonging to OAGADx in the enterprise end user capacity. (Assertion #3.1)
27. Verify that OAGADx can see in the list UEGACC1. (Assertion #3.2.1)
28. OAGADx logs out.
29. OAGADy logs in as an OAGi admin.
30. OAGADy goes to the CC Management page.
31. Verify that OAGADy can see in the list UEGACC5. (Assertion #3.2.2)
32. Verify that OAGADy can see the details of UEGACC2 but cannot make any change. (Assertion #3.3.1)
33. Verify that OAGADy can see the details of UEGACC3 but cannot make any change. (Assertion #3.3.2)
34. Verify that OAGADy cannot see the details of the UEGACC6. (Assertion #3.3.3)
35. OAGADy logs out.
36. OAGADx logs in as an OAGi admin and goes to the CC Management page.
37. OAGADx open UEGACC4 to add a new ASCC. Verify that no ASCCP belonging to enterprise tenants ever shows up for selection when adding an ASCC (use particularly the ASCCP that belongs to OAGADx in the enterprise end user capacity in the verification). Also, verify that user extension group ASCCPs are not available for selection. (Assertion #3.4)
38. OAGADx adds a new BCC to UEGACC4. (Assertion #3.5)
39. OAGADx removes an ASCC from UEGACC4. (Assertion #3.6)
40. OAGADx removes a BCC from UEGACC4. (Assertion #3.7)
41. OAGADx moves UEGACC4 to the Candidate state. (Assertion #3.8)
42. OAGADx moves UEGACC4 back to the Editing state. (Assertion #3.9)
43. OAGADx moves UEGACC4 to the Candidate and then Published state. (Assertion #3.10)
44. Verify that there is no sharing capability in the CC Management. (Assertion #4)
45. OAGADx change the ownership of UEGACC8 to OAGD1. Verify that no user without an OAGi tenant role is available for selection; and verify that the transfer is successful. (Assertion #5.1)
46. Verify that OAGADx cannot change the ownership of UEGACC5. (Assertion #5.2)
47. Verify that OAGADx can successfully change the ownership of UEGACC8 to OAGADx. (Assertion #5.3)
48. OAGADx changes the state of UEGACC5 to the Editing state.
49. OAGADx transfers the ownership of UEGACC5 to OAGADy. (Assertion #5.4)
50. OAGADx logs out.
51. OAGADy logs in as an OAGi admin.
52. OAGADy change the state of UEGACC5 to the Published state.
53. OAGADy logs out.
54. OAGADx logs in as an OAGi admin.
55. OAGADx opens PB7.
56. Verify that OAGADx gets a notification to up take the extension when opening the PB7 for edit. (note this is because PB7 uses the same extension as PB5). (Assertion #6.1)
57. OAGADx opens PB3.
58. Verify that OAGADx gets a notification to up take the extension when opening the PB3 for edit. (Assertion #6.2)
59. OAGADx logs out.

Assertions covered in this test case:

1. OAGi admin can creates a localized profile BOD extension.
2. OAGi admin can manage a localized profile BOD extension (through the localized profile BOD extension menu).
   1. An OAGi admin can add a new ASCC to an owned profile BOD extension in the Editing state using ASCCP belonging to the OAGi tenant but not ASCCP belonging to enterprise tenants in any state. Also, user extension group ASCCPs shall not be available for selection.
   2. An OAGi admin can add a new BCC to an owned profile BOD extension in the Editing state.
   3. An OAGi admin can remove an ASCC from an owned profile BOD extension in the Editing state.
   4. An OAGi admin can remove an BCC from an owned profile BOD extension in the Editing state.
   5. An OAGi admin can change the state of an owned profile BOD extension from Editing to Candidate.
   6. An OAGi admin can change the state of an owned profile BOD extension from Candidate to Published.
   7. An OAGi admin can change the state of an owned profile BOD extension from Candidate back to Editing.
   8. An OAGi admin can open up the extension to the same component that is still in the Editing state through another profile BOD.
   9. The system shall not allow an OAGi admin to create an extension to a component being extended by another OAGi tenant’s user.
      1. The system shall not allow an OAGi admin to create an extension to a component being extended by another OAGi developer in the Editing state.
      2. The system shall not allow an OAGi admin to create an extension to a component being extended by another OAGi admin in the Candidate state.
3. OAGi admin can manage user extension group ACCs on the CC Management page.
   1. An OAGi admin cannot see in the CC list any user extension group ACC belonging to enterprise tenants (test particularly on those ACCs owned by the OAGi admin in the enterprise tenant’s capacity).
   2. An OAGi admin can see in the CC list user extension group ACCs belonging to others in the OAGi tenant. The ACCs can be in any state.
      1. An OAGi admin can see in the list user extension group ACCs in the Editing state belonging to another OAGi developer.
      2. An OAGi admin can see in the list user extension group ACCs in the Candidate state belonging to another OAGi admin.
   3. An OAGi admin can see the detail of user extension group ACCs belonging to others in the OAGi tenant only in the Candidate or Published state but cannot make any change.
      1. An OAGi admin can see the detail of a user extension group ACC belonging to another OAGi developer in the Candidate state but cannot make any change.
      2. An OAGi admin can see the detail of a user extension group ACC belonging to another OAGi admin in the Published state but cannot make any change.
      3. An OAGi admin cannot see the detail of a user extension group ACC belonging to another OAGi admin in the Editing state.
   4. An OAGi admin can add a new ASCC to an owned ACC in the Editing state using ASCCP belonging to the OAGi tenant but not ASCCP belonging to enterprise tenants in any state. User extension group ASCCPs shall not be available for selection.
   5. An OAGi admin can add a new BCC to an owned ACC in the Editing state.
   6. An OAGi admin can remove an ASCC from an owned ACC in the Editing state.
   7. An OAGi admin can remove an BCC from an owned ACC in the Editing state.
   8. An OAGi admin can change the state of an ACC it owns from Editing to Candidate.
   9. An OAGi admin can change the state of an ACC it owns from Candidate back to Editing.
   10. An OAGi admin can change the state of an ACC it owns from Candidate to Published.
4. There is no sharing capability in the CC Management.
5. An OAGi admin can transfer the ownership of the user extension group ACC belonging to himself or other OAGi tenant’s users. The admin can transfer the ownership to any user with a OAGi tenant’s role. The ACC has to be in the Editing state.
   1. The OAGi admin can transfer the ownership of a user extension group ACC in the Editing state from another OAGi admin to another OAGi developer and not a user without an OAGi tenant’s role.
   2. The OAGi admin cannot transfer the ownership of his own user extension group ACC in the Candidate state.
   3. The OAGi admin can transfer the ownership of a user extension group ACC in the Editing state from another OAGi developer to another OAGi admin.
   4. The OAGi admin can transfer the ownership of his user extension group ACC in the Editing state to another OAGi admin.
6. The system shall make a notification to the user while opening up a profile BOD, to which a related user extension group ACC has been published.
   1. The system shall notify the OAGi admin when another OAGi admin published a user extension group ACC relevant to his profile BOD.
   2. The system shall notify the OAGi admin while opening a profile BOD when his user extension group ACC relevant to the profile BOD has been published.

## OAGi admin developer authorized access to CC management functions with the global extension

## OAGi admin developer authorized access to code list management functions.

Pre-condition: At least two OAGi admins, say OAGADx and OAGADy, with OAGi admin role and enterprise admin user role exists. OAGi tenant code lists exist in the system that are created by different OAGi developers and are in varying states. Private and shared enterprise tenants’ code lists exist in the system that are in varying states, some of which are owned by OAGADx and OAGADy in their enterprise admin user capacities

**From developer**: A user, say OAGD1, with OAGi developer role and enterprise end user role exists. Another user, say OAGD2, with OAGi developer role and enterprise admin user role exists. OAGi tenant code lists exist in the system that are created by different OAGi developers and are in varying states. Private and shared enterprise tenants’ code lists exist in the system that are in varying states, some of which are owned by OAGD1 and OAGD2 in their enterprise end user capacities.

Test steps:

1. A user, OAGD1, logs into the system as an OAGi developer.
2. OAGD1 creates new code lists and made some customizations, says CL0 (extensible), CL1, CL2. The developer moves to publish CL0, but leave the rest in the Editing.
3. The developer logs out.
4. A user, OAGADx logs into the system as an OAGi admin.
5. OAGADx creates a code list a code list without base, CL3 and add some values. (Assertion #1)
6. OAGADx creates a code list, CL4, using an extensible code list shared by an enterprise tenant as a base (use a code list not owned by the OAGADx user in the enterprise admin capacity). OAGADx creates a code list, CL5, using a published code list CL0 as a base. Verify that shared enterprise tenant’s code list that is not extensible can’t be used as a base. (Assertion #2)
7. OAGADx goes back and edit CL4 add/change some values. (Assertion #3)
8. OAGADx logs out.
9. A user, OAGADy, logs in as an OAGi admin.
10. OAGADy edits CL4 and and then edits CL1. (Assertion #4)
11. OAGADy moves CL1 and CL4 to the Published state (Assertion #5)
12. OAGADy delete CL2 and CL3. (Assertion #6)
13. Verify that there is no code-list-sharing functionality provided to the OAGADy. (Assertion #7)
14. Verify that there is no code-list-freeing functionality provided to the OAGADy. (Assertion #8)
15. OAGADy goes to the view code list page.
16. Verify that there is no private code list from an enterprise tenant in the list of code lists. (Assertion #9)
17. Verify that enterprise users’ shared code lists are in the list. (Assertion #10)
18. Verify that OAGADy can click on the shared code lists and see their details but cannot make any change. (Assertion #11)
19. Verify that OAGADy can see the details of CL0, CL1, and CL4 but cannot make any change. (Assertion #12)
20. OAGADy logs out.

Assertions covered in this test case:

1. An OAGi admin can create a code list without base.
2. An OAGi admin can create a code list based on another code list. Only published, extensible OAGi tenant’s code lists and shared enterprise tenant’s code list can be used as the based code list.
3. An OAGi admin can edit a code list he owns that is in the Editing state.
4. An OAGi admin can edit a code list he didn’t create but belongs to the OAGi tenant and in the Editing state.
5. An OAGi admin can change the state of a code list from Editing to Published.
6. An OAGi admin can delete a code list that is in the Editing state even though he didn’t create it.
7. Verify that there is no sharing functionality provided in the code list management. OAGi tenant cannot share code list.
8. Verify that there is no make free functionality provided in the code list management.
9. An OAGi admin cannot see in the list of code lists enterprise users’ private code lists in any state.
10. An OAGi admin can see in the list of code lists enterprise users’ shared code lists.
11. An OAGi admin can see the details of enterprise users’ shared code lists but cannot make any change.
12. An OAGi admin can see the details of published code list belonging to any OAGi tenant’s user but cannot make any change.

# Enterprise end user access right to SRT core functions

Requirement document reference: Section 3.2.2, 3.4.3.1

## Enterprise end user’s authorized management of context categories

Pre-condition: At least two users, say ENTU1 and ENTU2, with both enterprise end user and OAGi developer roles exist. There are context categories created by various enterprise users[[6]](#footnote-6), some are created by ENTU1 and ENTU2 in the OAGi developer role capacity, and some of which have been shared.

Test Steps:

1. An enterprise admin user logs into the system.
2. The admin user creates new context categories, says CAT0 and CAT1. (Assertion #1)
3. The admin user shares CAT0. (Assertion #2)
4. The admin user logs out.
5. An enterprise end user, says ENTU1, logs into the system.
6. ENTU1 creates two new context category, says CAT2 and CAT3. (Assertion #3)
7. ENTU1 log out.
8. Another enterprise end user, says ENTU2, logs into the system.
9. ENTU2 views the list of context categories.
10. Verify that enterprise users’ context categories, which have not been shared, are not visible to ENTU2 in the list. Make sure that the verification statements include context categories that are created by ENTU1and ENTU2 in the OAGi developer role capacity. (Assertion #4)
11. Verify that enterprise users’ context categories, which have been shared, are visible to ENTU2 in the list. (Assertion #5)
12. Verify that OAGi developers’ context categories, which have been shared, are visible to ENTU2 in the list. (Assertion #6)
13. Verify that ENTU2 can see the details of those visible OAGi developers’ context categories but cannot make any change (verify with one created by other users and another one created by ENTU2 in the OAGi developer capacity). (Assertion #7)
14. Verify that CAT0, CAT1, CAT2, and CAT3 are in the list. (Assertion #8)
15. ENTU2 deletes CAT1 and CAT2. (Assertion #9)
16. Verify that ENTU2 cannot delete CAT0. (Assertion #10)
17. ENTU2 edits CAT3. (Assertion #11)
18. Verify that ENTU2 cannot share CAT3. (Assertion #12)
19. ENTU2 logs out.
20. The admin user logs in again.
21. The admin user shares CAT3.
22. The admin user logs out.
23. ENTU2 logs in as an enterprise end user.
24. Verify that ENTU2 cannot make change or delete CAT3. (Assertion #10, 13)

Assertions covered in this test case:

1. An enterprise admin user can create a context category.
2. An enterprise admin user can share a context category.
3. An enterprise end user can create a context category.
4. An enterprise end user cannot see a list of context categories that are not shared.
5. An enterprise end user can see a list of context categories that includes enterprise users’ shared context categories.
6. An enterprise end user can see a list of context categories that includes OAGi developers’ shared context categories.
7. An enterprise end user can see the details of OAGi developers’ shared context categories but cannot make any change.
8. An enterprise end user can see a list of context categories, including the ones owned by users from the same enterprise tenant.
9. An enterprise end user can delete context categories, including the ones owned by users from the same enterprise tenant.
10. An enterprise end user cannot delete shared context categories owned by users from the same enterprise tenant.
11. An enterprise end user can edit non-shared context categories, including the ones by users from the same enterprise tenant.
12. An enterprise end user cannot share context category.
13. An enterprise end user cannot edit shared context categories.

## Enterprise end user fails to create Context Category due to omitting name

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise end user role.

Test Steps:

1. A user with the Enterprise end user role, aEntUser1, logs in.
2. The Enterprise end user accesses the context category management functionality.
3. The Enterprise end user creates new context category, but omits name value.
4. Verify that context category creation failed.

## Enterprise end user fails to edit Context Category due to omitting name

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise end user role. Context Category, testCategory, already exists.

Test Steps:

1. A user with the Enterprise end user role, aEntUser1, logs in.
2. The Enterprise end user accesses the context category management functionality.
3. The Enterprise end user removes the value for context category name.
4. Verify that context category edits failed.

## Enterprise end user’s authorized management of context schemes

Pre-condition: At least two users, say ENTU1 and ENTU2, with both enterprise end user and OAGi developer roles exist. There are context categories created by various enterprise users[[7]](#footnote-7), some are created by ENTU1 and ENTU2 in the OAGi developer role capacity, and some of which have been shared.

Test Steps:

1. An enterprise admin user logs into the system.
2. The admin user creates new context schemes (with some values), says CS0 and CS1. (Assertion #1)
3. The admin user shares CS0. (Assertion #2)
4. The admin user logs out.
5. An enterprise end user, says ENTU1, logs into the system.
6. ENTU1 creates two new context schemes, says CS2 and CS3. Verify that both context categories that are owned by OAGi tenant and that are shared by enterprise tenants are selectable. Verify that enterprise tenants’ private context categories are not selectable. (Assertion #3, 4, 5, 6)
7. ENTU1 log out.
8. Another enterprise end user, says ENTU2, logs into the system.
9. ENTU2 views the list of context schemes.
10. Verify that enterprise users’ context schemes, which have not been shared, are not visible to ENTU2 in the list. Make sure that the verification statements include context schemes that are created by ENTU1 and ENTU2 in the enterprise end user capacity. (Assertion #7)
11. Verify that enterprise users’ context schemes, which have been shared, are visible to ENTU2 in the list. (Assertion #8)
12. Verify that OAGi developers’ context categories, which have been shared, are visible to ENTU2 in the list. (Assertion #9)
13. Verify that ENTU2 can see the details of those visible OAGi developers’ context schemes but cannot make any change and cannot drill down to see the context category details (verify with one created by other users and another one created by ENTU2 in the OAGi developer capacity). (Assertion #10)
14. Verify that CS0, CS1, CS2, and CS3 are in the list. (Assertion #11)
15. ENTU2 deletes CS1 and CS2. (Assertion #12)
16. Verify that ENTU2 cannot delete CS0. (Assertion #13)
17. ENTU2 edits CS3 (CS3 data update, add, update, remove CS3 values). (Assertion #14)
18. Verify that ENTU2 cannot share CS3. (Assertion #15)
19. ENTU2 logs out.
20. The admin user logs in again.
21. The admin user shares CS3.
22. The admin user logs out.
23. ENTU2 logs in as an enterprise end user.
24. Verify that ENTU2 cannot make change or delete CS3. (Assertion #13, 16)

Assertions covered in this test case:

1. An enterprise admin user can create a context scheme.
2. An enterprise admin user can share a context scheme.
3. An enterprise end user can create a context scheme.
4. An enterprise end user cannot see a list of context categories that are not shared.
5. An enterprise end user can see a list of context categories that includes enterprise users’ shared context categories.
6. An enterprise end user can see a list of context categories that includes OAGi developers’ shared context categories.
7. An enterprise end user cannot see a list of context schemes that are not shared.
8. An enterprise end user can see a list of context schemes that includes enterprise users’ shared context schemes.
9. An enterprise end user can see a list of context schemes that includes OAGi developers’ shared context schemes.
10. An enterprise end user can see the details of OAGi developers’ shared context schemes but cannot make any change or drill down to see details of context categories.
11. An enterprise end user can see a list of context schemes, including the ones owned by users from the same enterprise tenant.
12. An enterprise end user can delete context schemes, including the ones owned by users from the same enterprise tenant.
13. An enterprise end user cannot delete shared context schemes owned by users from the same enterprise tenant.
14. An enterprise end user can edit non-shared context schemes, including the ones by users from the same enterprise tenant.
15. An enterprise end user cannot share context scheme.
16. An enterprise end user cannot edit shared context schemes.

## Enterprise end user fails to create Context Scheme due to omitting required fields

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise end user role.

Test Steps:

1. A user with the Enterprise end user role, aEntUser1, logs in.
2. The Enterprise end user accesses the context scheme management functionality.
3. The Enterprise end user creates new context category, but omits some of required fields (e.g. Context Category, Name, Scheme ID, Agency ID or Version).
4. Verify that context category creation failed.

## Enterprise end user fails to edit Context Scheme due to omitting required fields

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise end user role. Context Scheme, testScheme, already exists.

Test Steps:

1. A user with the Enterprise end user role, aEntUser1, logs in.
2. The Enterprise end user accesses the context scheme management functionality.
3. The Enterprise end user edits testScheme, but omits some of required fields (e.g. Context Category, Name, Scheme ID, Agency ID or Version).
4. Verify that context scheme edit failed.

## Enterprise end user fails to add Context Scheme Value due to omitting value

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise end user role. Context Scheme, newScheme, already exists.

Test Steps:

1. A user with the Enterprise tenant end user role, aEntUser1, logs in.
2. The Enterprise end user accesses the context scheme management functionality.
3. The Enterprise end user selects context scheme, newScheme.
4. The Enterprise end user add context scheme value to the newScheme, but omits value.
5. Verify that context scheme is not updated and values addition fails.

## Enterprise end user fails to edit Context Scheme Value due to omitting value

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise end user role. Context Scheme, newScheme, already exists.

Test Steps:

1. A user with the Enterprise tenant end user role, aEntUser1, logs in.
2. The Enterprise end user accesses the context scheme management functionality.
3. The Enterprise end user selects context scheme, newScheme.
4. The Enterprise end user add context scheme value to the newScheme, but omits the value.
5. Verify that context scheme is not updated and value addition fails.

## Enterprise end user’s authorized access to code list management functions.

Pre-condition: At least two users, say ENTU1 and ENTU2, with both enterprise end user and OAGi developer roles exist. There are private and shared code lists created by various enterprise users[[8]](#footnote-8) and in varying states, some are created by ENTU1 and ENTU2 in the OAGi developer role capacity, and some of which have been shared. OAGi tenant code lists exist in the system that are created by different OAGi developers and are in varying states.

1. An enterprise admin user logs into the system.
2. The admin user creates new code list and made some customizations, says CL0 and CL1. The admin user moves to publish CL0, but still leave CL1 in the Editing state.
3. The admin user logs out.
4. An enterprise end user, ENTU1, logs into the system.
5. ENTU1 creates new code lists with some values, says CL2, CL3, and CL4. OAGD1 advances CL2 to the Published state while CL3 and CL4 are left in the Editing state. (Assertion #1, 2, 3)
6. Verify that ENTU1 cannot make any change to CL2. (Assertion #4)
7. Verify that ENTU1 can successfully discard CL4. (Assertion #5)
8. Verify that ENTU1 cannot change the discarded state of CL4. (Assertion #6)
9. Verify that ENTU1 cannot share CL0 and CL1. (Assertion #7)
10. Verify that ENTU1 cannot make CL0 and CL1 free. (Assertion #8)
11. Verify that ENTU1 cannot share CL2 and CL3. (Assertion #7)
12. Verify that ENTU1 cannot make CL2 and CL3 free. (Assertion #8)
13. OAGD2 logs out.
14. ENTU2 views the list of code lists.
15. Verify that enterprise users’ private code lists are not visible to ENTU2 in the list including those owned by ENTU2 in the enterprise end user capacity. (Assertion #9)
16. Verify that enterprise users’ shared code lists are visible to ENTU2 in the list. (Assertion #10)
17. Verify that CL0, CL1, CL2, CL2, CL3 are in the list. (Assertion #11)
18. Verify that ENTU2 can see the details of at least one of those visible enterprise users’ shared code lists but cannot make any change. (Assertion #12)
19. Verify that ENTU2 can see the details of CL0 but cannot make any change (because it is published even though it is owned by the admin developer). (Assertion #13)
20. Verify that ENTU2 cannot see nor edit the detail of CL1 (which is still in the Editing state and owned by the admin user). (Assertion #14)
21. Verify that ENTU2 can see the details of CL2 but cannot make any change (because it is published even though it is owned by the ENTU1). (Assertion #15)
22. Verify that ENTU2 cannot see the detail nor making any change to CL3 (because it is still in the Editing state and is owned by another developer, ENTU1). (Assertion #16)
23. ENTU2 logs out.
24. ENTU1 logs in.
25. ENTU1 accesses the code list management functionality end creates new code list without base.
26. ENTU1 populates the name and version. User arbitrarily populates other code list properties, adds some values, and creates a new code list.
27. Verify that new code list is created. (Assertion #17)
28. ENTU1 accesses the code list management functionality end creates new code list based on other code list.
29. Verify that all code lists available to be chosen for the base are both extensible and in published state. (Assertion #18)
30. ENTU1 keeps default values for code list properties. User arbitrarily populates other code list properties and creates a new code list.
31. Verify that code list with base is created. (Assertion #19)
32. Verify that ENTU1 cannot share any of enterprise tenant’s code lists. (Assertion #20)

Assertions covered in this test case:

1. An enterprise end user can create a code list.
2. An enterprise end user can edit a code list it owns.
3. An enterprise end user can change the state of a code list he owns from Editing to Published.
4. An enterprise end user cannot make any change to an owned code list that is in the Published state.
5. An enterprise end user can discard his own code list that is in the Editing state.
6. An enterprise end user cannot change the state of the code list in Discarded state.
7. An enterprise end user cannot tenant’s owned share code list.
8. An enterprise end user cannot make code list free.
9. An enterprise end user cannot see a list of code lists that includes enterprise users’ private code lists in any state.
10. An enterprise end user can see a list of code lists that includes enterprise users’ shared code lists.
11. An enterprise end user can see a list of code lists that includes enterprise tenant’s code lists in any state.
12. An enterprise end user can see the details of published code lists owned by another enterprise end user, but cannot make any change.
13. An enterprise end user enterprise end user can see the details of published code lists owned by an admin developer, but cannot make any change.
14. An enterprise end user cannot see the details of or make any change to code lists in the Editing state owned by another enterprise admin user.
15. An enterprise end user cannot see the details of or make any change to code lists in the Editing state owned by another enterprise end user.
16. An enterprise end user cannot see the details of or make any change to code lists in the Published state owned by another enterprise end user.
17. An enterprise end user can create a code list without base.
18. An enterprise end user cannot create a code list with base that is not extensible nor published.
19. An enterprise end user can create a code list with base.
20. An enterprise end user cannot share tenant’s owned code list.

## Enterprise end user fails to create Code List due to omitting required fields

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise end user role.

Test Steps:

1. A user with the Enterprise tenant end user role logs in.
2. The Enterprise end user accesses the code list management functionality end creates new code list without base.
3. The Enterprise end user omits populating required fields (e.g. Name, Version) and tries to create code list.
4. Verify that code list creation fails.

## Enterprise end user successfully edits Code List

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise end user role. Code List, aNewCodeList, already exists.

Test Steps:

1. A user with the Enterprise tenant end user role, aEntUser1, logs in.
2. The Enterprise end user accesses the code list management functionality.
3. The Enterprise end user renames code list, aNewCodeList, to testCodeList. Optionally, user updates other testCodeList properties.
4. Verify that code list is updated.

## Enterprise end user fails to edit Code List due to omitting required fileds

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise end user role. Code List, testCodeList, already exists.

Test Steps:

1. A user with the Enterprise tenant end user role, aEntUser1, logs in.
2. The Enterprise end user accesses the code list management functionality.
3. The Enterprise end user finds testCodeList , removed values of required fields (Name, Version) and tries to update code list.
4. Verify that code list edits fail.

## Enterprise end user successfully adds Code List Values

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise end user role. Code List, oacl\_TaxCode\_Extension, already exists.

Test Steps:

1. A user with the Enterprise tenant end user role, aEntUser1, logs in.
2. The Enterprise end user accesses the code list management functionality.
3. The Enterprise end user selects code list, oacl\_TaxCode\_Extension.
4. The Enterprise end user add several code list values to the oacl\_TaxCode\_Extension.
5. Verify that code list is updated and values are added.

## Enterprise end user fails to add Code List Value due to omitting code

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise end user role. Code List, oacl\_TaxCode\_Extension, already exists.

Test Steps:

1. A user with the Enterprise tenant end user role, aEntUser1, logs in.
2. The Enterprise end user accesses the context scheme management functionality.
3. The Enterprise end user selects code list, oacl\_TaxCode\_Extension.
4. The Enterprise end user add code list value, but omits the code.
5. Verify that code list is not updated and values addition fails.

## Enterprise end user successfully edits Code List Values

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise end user role. Code List, oacl\_TaxCode\_Extension, already exists.

Test Steps:

1. A user with the Enterprise tenant end user role, aEntUser1, logs in.
2. The Enterprise end user accesses the code list management functionality.
3. The Enterprise end user selects code list, oacl\_TaxCode\_Extension.
4. The Enterprise end user edit code list scheme values of the oacl\_TaxCode\_Extension.
5. Verify that code list values are updated.

## Enterprise end user fails to edit Code List Value due to omitting value

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise end user role. Code List, oacl\_TaxCode\_Extension, already exists.

Test Steps:

1. A user with the Enterprise tenant end user role, aEntUser1, logs in.
2. The Enterprise end user accesses the code list management functionality.
3. The Enterprise end user selects code list, oacl\_TaxCode\_Extension.
4. The Enterprise end user edit some code list scheme value, but omits the code.
5. Verify that code list value update fails.

## Enterprise end user’ authorized management of business contexts

Pre-condition: At least two users, say ENTU1 and ENTU2, with both enterprise end user and OAGi developer roles exist. There are business contexts created by various enterprise users[[9]](#footnote-9), some are created by ENTU1 and ENTU2 in the OAGi developer role capacity, and some of which have been shared.

Test Steps:

1. An enterprise admin user logs into the system.
2. The admin user creates new business contexts (with some context scheme values), says BC0 and BC1. (Assertion #1)
3. The admin user shares BC0. (Assertion #2)
4. The admin user logs out.
5. An enterprise end user, says ENTU1, logs into the system.
6. ENTU1 creates two new business contexts, says BC2 and BC3. Verify that both context categories that are owned by OAGi tenant and that are shared by enterprise tenants are selectable. Verify that enterprise tenants’ private context categories are not selectable. (Assertion #3, 4, 5, 6)
7. ENTU1 log out.
8. Another enterprise end user, says ENTU2, logs into the system.
9. ENTU2 views the list of business contexts.
10. Verify that enterprise users’ business contexts, which have not been shared, are not visible to ENTU2 in the list. Make sure that the verification statements include business context that are created by ENTU1 and ENTU2 in the enterprise end user capacity. (Assertion #7)
11. Verify that enterprise users’ business contexts, which have been shared, are visible to ENTU2 in the list. (Assertion #8)
12. Verify that OAGi developers’ business contexts, which have been shared, are visible to ENTU2 in the list. (Assertion #9)
13. Verify that ENTU2 can see the details of those visible OAGi developers’ business contexts but cannot make any change and cannot drill down to see the context scheme detail nor the context category detail (verify with one created by other users and another one created by ENTU2 in the OAGi developer capacity). (Assertion #10)
14. Verify that BC0, BC1, BC2, and BC3 are in the list. (Assertion #11)
15. ENTU2 deletes BC1 and BC2. (Assertion #12)
16. Verify that ENTU2 cannot delete BC0. (Assertion #13)
17. ENTU2 edits BC3 (BC3 data update, add, update, remove BC3 values). (Assertion #14)
18. Verify that ENTU2 cannot share BC3. (Assertion #15)
19. ENTU2 logs out.
20. The admin user logs in again.
21. The admin user shares CBC.
22. The admin user logs out.
23. ENTU2 logs in as an enterprise end user.
24. Verify that ENTU2 cannot make change or delete BC3. (Assertion #13, 16)

Assertions covered in this test case:

1. An enterprise admin user can create a business context.
2. An enterprise admin user can share a business context.
3. An enterprise end user can create a business context.
4. An enterprise end user cannot see a list of business contexts that are not shared.
5. An enterprise end user can see a list of business contexts that includes enterprise users’ shared context categories.
6. An enterprise end user can see a list of business contexts that includes OAGi developers’ shared context categories.
7. An enterprise end user cannot see a list of business contexts that are not shared.
8. An enterprise end user can see a list of business contexts that includes enterprise users’ shared business contexts.
9. An enterprise end user can see a list of business contexts that includes OAGi developers’ shared business contexts.
10. An enterprise end user can see the details of OAGi developers’ shared business contexts but cannot make any change or drill down to see details of context schemes or context categories.
11. An enterprise end user can see a list of business contexts, including the ones owned by users from the same enterprise tenant.
12. An enterprise end user can delete business contexts, including the ones owned by users from the same enterprise tenant.
13. An enterprise end user cannot delete shared business contexts owned by users from the same enterprise tenant.
14. An enterprise end user can edit non-shared business contexts, including the ones by users from the same enterprise tenant.
15. An enterprise end user cannot share business context.
16. An enterprise end user cannot edit shared business context.

## Enterprise end user fails to create Business Context due to omitting name

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise end user role.

Test Steps:

1. A user with the Enterprise tenant end user role logs in.
2. The Enterprise end user accesses the business context management functionality.
3. The Enterprise end user creates new business context, but omits the name.
4. Verify that business context creation fails.

## Enterprise end user fails to edit Business Context due to omitting name

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise end user role. Business Context, aBizCtx, already exists.

Test Steps:

1. A user with the Enterprise end user role, aEntUser1, logs in.
2. The Enterprise end user accesses the context category management functionality.
3. The Enterprise end user removes the value for business context name.
4. Verify that business context edits failed.

## Enterprise end user’s authorized management of profile BODs

Pre-condition: At least two users, say ENTU1 and ENTU2, with both enterprise end user and OAGi developer roles exist. Free profile BODs published by the OAGi tenant exist in the system. Private and shared OAGi tenant profile BODs exist in the system that are created by different OAGi developers, as well as by ENTU1 and ENTU2 in the OAGi developer role capacity, and are in varying states. Private and shared enterprise tenants profile BODs exist in the system that are in varying states, some of which are owned by ENTU1 and ENTU2 in their enterprise end user capacities.

Test Steps:

1. An enterprise admin user, ENTADx, logs into the system.
2. The admin user creates new profile BODs and made some customizations, says PB0, PB1, PB2. The admin developer moves to publish PB0, but still leave PB1 in the Editing state and PB2 in the Candidate state.
3. The admin user shares PB0.
4. The admin user logs out.
5. An enterprise end user, says ENTU1, logs into the system.
6. ENTU1 creates new profile BODs with some customizations (here, make sure that the local and global extension functions are invoked in the customizations), says PB3, PB4, PB5, and PB6. ENTU1 advances PB3 to the Published state while PB4 and PB6 are left in the Editing state and PB5 is in the Candidate state. (Assertion #1, 2, 3, 4)
7. Verify that ENTU1 cannot make any change to PB3[[10]](#footnote-10). (Assertion #6)
8. Verify that ENTU1 cannot make any change to PB5 except changing the state back to Editing or deleting it (note keep PB5 at the Editing state, just check that button to change the state back is enabled). (Assertion #5)
9. Verify that ENTU1 can successfully delete PB6. (Assertion #7)
10. Verify that ENTU1 cannot share PB0, PB1, and PB2. (Assertion #8)
11. Verify that ENTU1 cannot make PB0, PB1, and PB2 free. (Assertion #9)
12. Verify that ENTU1 cannot share PB3, PB4, and PB5. (Assertion #10)
13. Verify that ENTU1 cannot make PB3, PB4, and PB5 free. (Assertion #11)
14. ENTU1 logs out.
15. Another enterprise end user, says ENTU2, logs into the system.
16. ENTU2 views the list of profile BODs.
17. Verify that enterprise users’ private profile BODs are not visible to ENTU2. (Assertion #12)
18. Verify that enterprise users’ shared profile BODs are visible to ENTU2 in the list. (Assertion #13)
19. Verify that PB0, PB1, PB2, PB3, PB4, and PB5 are in the list. (Assertion #14)
20. Verify that ENTU2 can see the details of at least one of those visible enterprise users’ shared profile BODs but cannot make any change. Verify that ENTU2 cannot drill down to make change to the associated business context and context categories or see and edit the whole context scheme (the user should be able to see the context scheme values used in the business context). (Assertion #15)
21. Verify that ENTU2 can see the details of PB0 but cannot make any change (because it is published even though it is owned by the admin user). (Assertion #16)
22. Verify that ENTU2 cannot see, edit the detail, nor change state and ownership of PB1 (which is still in the Editing state and owned by the admin user). (Assertion #17)
23. Verify that ENTU2 can see (review) the detail of PB2 (which is in the Candidate state) but cannot make any change. (Assertion #18)
24. Verify that ENTU2 can see the details of PB3 but cannot make any change (because it is published even though it is owned by the ENTU1). (Assertion #19)
25. Verify that ENTU2 cannot see the detail nor making any change to PB4 (because it is still in the Editing state and is owned by another developer, ENTU1). (Assertion #20)
26. Verify that ENTU2 can see (review) the detail of PB5 (which is in the Candidate state) but cannot make any change. (Assertion #21)
27. ENTU2 logs out.
28. ENTU1 logs in.
29. Verify that ENTU1 can successfully change the ownership of PB4 to ENTU2. (Assertion #22)
30. Verify that ENTU1 cannot change the ownership of PB5. (Assertion #23)
31. Verify that ENTU1 can successfully change the state of PB5 back to the Editing state. (Assertion #5)
32. Verify that ENTU1 can successfully change the ownership of PB5 to ENTADx. (Assertion #24)
33. ENTU1 logs out.
34. ENTU2 logs in as an enterprise end user.
35. Verify that ENTU2 can edit (be sure to invoke the local & global extension functions) and change the state of PB4 to the Published state. (Assertion #2, 4)
36. ENTU2 logs out.
37. The admin user, ENTADx, logs in.
38. ENTADx transfer the ownership of PB5 to ENTU2.
39. ENTADx logs out.
40. ENTU2 logs in.
41. ENTU2 goes to the Copy Profile BOD page.
42. Verify that ENTU2 cannot see and choose any enterprise users’ private profile BODs for copying including those owned by ENTU2 in the enterprise end user tenant capacity. (Assertion #25)
43. Verify that ENTU2 can see and choose enterprise tenant’s shared profile BODs for copying. (Assertion #26)
44. Verify that ENTU2 can see and choose PB0, PB2, PB3, and PB4 for copying. (Assertion #27)
45. Verify that ENTU2 cannot see and choose PB1 for copying (it is ok to fail this b/c this is a known bug in the version given to JT). (Assertion #28)
46. Verify that ENTU2 cannot see and choose PB5 for copying (it is ok to fail this b/c this is a known bug in the version given to JT). (Assertion #29)
47. ENTU2 goes to the expression generation page.
48. Verify that ENTU2 cannot see and choose any enterprise users’ private profile BODs for generating an expression including those owned by ENTU2 in the enterprise end user’s capacity. (Assertion #30)
49. Verify that ENTU2 can see and choose enterprise tenant’s shared profile BODs for generating an expression. (Assertion #31)
50. Verify that ENTU2 can see and choose PB0, PB2, PB3, and PB4 for generating an expression. (Assertion #32)
51. Verify that ENTU2 cannot see and choose PB1 for generating an expression (it is ok to fail this b/c this is a known bug in the version given to JT). (Assertion #33)
52. Verify that ENTU2 cannot see and choose PB5 for generating an expression (it is ok to fail this b/c this is a known bug in the version given to JT). (Assertion #34)
53. ENTU2 logs out.

Assertions covered in this test case:

1. An enterprise end user can create a profile BOD.
2. An enterprise end user can edit a profile BOD it owns.
3. An enterprise end user can change the state of a profile BOD he owns from Editing to Candidate.
4. An enterprise end user can change the state of a profile BOD he owns from Candidate to Published.
5. An enterprise end user cannot make any change to his own profile BOD in Candidate state except changing the state from Candidate back to Editing or deleting it.
6. An enterprise end cannot make any change to an owned profile BOD that is in the Published state.
7. An enterprise end user can delete his own profile BOD that is in the Editing state.
8. An enterprise end user cannot share an unowned profile BOD in any state.
9. An enterprise end user cannot make an unowned profile BOD in any state free.
10. An enterprise end user cannot share an owned profile BOD in any state.
11. An enterprise end user cannot make an owned profile BOD in any state free.
12. An enterprise end user cannot see a list of profile BODs that includes enterprise users’ private profile BODs in any state.
13. An enterprise end user can see a list of profile BODs that includes enterprise users’ shared profile BODs.
14. An enterprise end user can see a list of profile BODs that includes other enterprise tenant’s profile BODs in any state.
15. An enterprise end user can see the details of enterprise users’ shared profile BODs including its business context but cannot make any change to the profile BODs and any associated information, and cannot drill down to see the whole context scheme values.
16. An enterprise end user can see the details of published profile BODs owned by an admin user, but cannot make any change.
17. An enterprise end user cannot see the details of or make any change to profile BODs in the Editing state owned by another enterprise admin user.
18. An enterprise end user can see the details of the candidate profile BODs owned by an admin user, but cannot make any change.
19. An enterprise end user can see the details of published profile BODs owned by another enterprise end user, but cannot make any change.
20. An enterprise end user cannot see the details of or make any change to profile BODs in the Editing state owned by another enterprise end user.
21. An enterprise end user can see the details of the candidate profile BODs owned by another enterprise end user, but cannot make any change.
22. An enterprise end user can transfer the ownership of his own profile BODs in Editing state to another enterprise end user.
23. An enterprise end user cannot change the ownership of his own profile BODs in Candidate state.
24. An enterprise end user can transfer the ownership of his own profile BODs in Editing state to an enterprise admin user.
25. An enterprise end user cannot copy an enterprise tenant’s private profile BOD.
26. An enterprise end user can copy an enterprise tenant’s shared profile BOD.
27. An enterprise end user can generate expressions of profile BODs that belong to other enterprise end and admin users and that are in candidate or published states.
28. An enterprise end user cannot copy profile BODs that belong to an enterprise admin user and that are in editing states.
29. An enterprise end user cannot copy profile BODs that belong to another Enterprise end user and that are in editing states.
30. An enterprise end user cannot generate expressions of an enterprise users’ private profile BODs.
31. An enterprise end user can generate expressions of an enterprise users’ shared profile BODs.
32. An enterprise end user can generate expressions of profile BODs that belong to other enterprise end and admin users and that are in candidate or published states.
33. An enterprise end user cannot generate expressions of profile BODs that belong to another Enterprise end user and that are in editing states.
34. An enterprise end user cannot generate expressions of profile BODs that belong to an Enterprise admin user and that are in editing states.

## Enterprise end user successfully edits profile BOD

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise end user role. Acknowledge BOM profile BOD for newBizCtx business context already exists.

Test Steps:

1. A user with the Enterprise tenant end user role, aEntUser1, logs in.
2. The Enterprise end user accesses the profile BODs functionality.
3. The Enterprise end user finds Acknowledge BOM profile BOD for the newBizCtx business context.
4. The Enterprise end user arbitrarily updates profile BOD by checking components that should be added/removed from the profile and updates the profile BOD.
5. Verify that profile BOD is updated.

## Enterprise end user’s authorized access to CC management functions with localized extension

Pre-condition: At least two users, say ENTU1 and ENTU2, with both enterprise end user and OAGi developer roles exist. There are business contexts created by various enterprise users[[11]](#footnote-11), some are created by ENTU1 and ENTU2 in the OAGi developer role capacity, and some of which have been shared.

Test Steps:

1. An enterprise admin user, ENTADx,, logs into the system.
2. ENTADx, creates a new profile BOD, PB0.
3. ENTADx, makes an localized extension to PB0. This results in, says, User Extension Group ACC, UEGACC0, and corresponding ASCCP0 and ASCC0 (these two are supposed to be hidden on the CC Management page by default logic).
4. ENTADx adds some ASCCs and BCCs to the extension, i.e., to UEGACC0.
5. ENTADx advances UEGACC1 to the Published state.
6. ENTADx repeats step 2 to 4 two times resulting PB1, PB2, and corresponding UEGACC1, ASCCP1, ASCC1, and UEGACC2, ASCCP2, ASCC2, respectively.
7. ENTADx advances UEGACC2 to the Candidate state.
8. ENTADx logs out.
9. ENTU1 logs into the system as an enterprise end user.
10. ENTU1 creates new profile BODs, PB3, PB4, PB5, and PB6.
11. ENTU1 makes some extension to PB3, PB4, PB5 and PB6 that results in UEGACC3, ASCCP3, ASCC3, UEGACC4, ASCCP4, ASCC4, UEGACC5, ASCCP5, ASCC5, and UEGACC6, ASCCP6, ASCC6.
12. ENTU1 advances PB3 extension, i.e., UEGACC3 to the Published state and PB5 extension to the Candidate state.
13. Verify that ENTU1 can discard PB6 extension resulting removal of UEGACC6, ASCCP6, ASCC6.
14. Verify that ENTU1 can no longer make any change to UEGACC3. Since UEGACC3 won’t be accessible thru the BIE extension anymore, this means searching for the UEGACC3 on the CC Management page and verify that it can no longer be edited. Also, verify that ASCCP3 and ASCC3 cannot be edited.
15. Verify that ENTU1 cannot make any change to UEGACC5 except changing the state back to Editing (note keep UEGACC5 in the Editing state after this step). Verify this through both the BIE extension page and the CC Management page.
16. Verify that ENTU1 can review UEGACC2 through the CC Management page.
17. Verify that ENTU1 cannot make an extension related to UEGACC1 and UEGACC2 in another profile BOD he owns.
18. ENTU1 logs out.
19. Another enterprise end user, says, ENTU2, logs into the system.
20. ENTU2 goes to the CC Management page.
21. Verify that enterprise users’ CCs are not visible to ENTU2, including those that ENTU1 and ENTU2 created in their enterprise user capacity.
22. Verify that UEGACC0 to UEGACC5 are in the list.
23. Verify that ENTU2 cannot make any change to UEGACC0.
24. Verify that ENTU2 cannot see the detail nor make any change to UEGACC1.
25. Verify that ENTU2 can see the detail of UEGACC2 but cannot make any change.
26. Verify that ENTU2 cannot make any change to UEGACC3.
27. Verify that ENTU2 cannot see the detail nor make any change to UEGACC4.
28. Verify that ENTU2 can see the detail of UEGACC5 but cannot make any change.
29. ENTU2 logs out.
30. ENTU1 logs in as an enterprise end user.
31. Verify that ENTU1 can successfully change the ownership of UEGACC4 to ENTU2.
32. Verify that ENTU1 cannot change the ownership of UEGACC5.
33. Verify that ENTU1 can successfully change the state of UEGACC5 back to the Editing state.
34. Verify that ENTU1 can successfully change the ownership of UEGACC5 to OAGADx.
35. ENTU1 logs out.
36. ENTU2 logs in as an enterprise end user.
37. Verify that ENTU2 can edit and change the state of UEGACC4 to the Published state.
38. ENTU2 logs out.
39. ENTU1 logs in as an enterprise end user.
40. ENTU1 opens PB4.
41. Verify that ENTU1 gets a notification to up take the extension when opening the PB4 for edit.
42. ENTU1 opens PB3.
43. Verify that ENTU1 gets a notification to up take the extension when opening the PB3 for edit.
44. ENTU1 logs out.
45. ENTADx logs in as an enterprise admin user.
46. ENTADx transfers the ownership of UEGACC5 to ENTU2.
47. ENTADx logs out.
48. ENTU2 logs in as an enterprise end user.
49. ENTU2 goes to the CC Management page.
50. Verify that ENTU2 can see and edit UEGACC5.

Assertions covered in this test case:

1. An Enterprise end user can add a new ASCC to an owned ACC in the Editing state using ASCCP belonging to the enterprise tenant but not ASCCP belonging to OAGI or other enterprise tenants in any state.
2. An Enterprise end user can remove a new ASCC from an owned ACC in the Editing state.
3. An Enterprise end user can add a new BCC to an owned ACC in the Editing state.
4. An Enterprise end user can change the state of an ACC it owns from Editing to Candidate.
5. An Enterprise end user can change the state of an ACC it owns from Candidate to Published
6. An Enterprise end user can change the state of an ACC it owns from Candidate back to Editing.
7. An Enterprise end user can remove an BCC from an owned ACC in the Editing state.
8. An Enterprise end user can see the list of all OAGi tenant’s CCs in any state.
9. An Enterprise end user cannot see the list (and hence details and can’t edit) of enterprise tenants’ CCs in any state.
10. An Enterprise end user can see the details of OAGi admin developer’s CCs in the Candidate state but cannot make any change.
11. An Enterprise end user can see the details of OAGi admin developer’s CCs in the Published state but cannot make any change.
12. An Enterprise end user can see the details of another Enterprise end user’s CCs in the Candidate state but cannot make any change.
13. An Enterprise end user can see the details of OAGi admin developer’s CCs in the Published state but cannot make any change.

## Enterprise end user’s authorized access to CC management functions with global extension

Repeat Test Case 11.22 but with the global extension.

## Enterprise end user cannot delete context categories used by a context scheme

Pre-condition: At least a user, say aEntUser1, with Enterprise end user role exists. There are private context schemes created by various Enterprise tenant roles[[12]](#footnote-12). One context scheme uses a private context category, says CAT1, owned by aEntUser1 and another context scheme uses a shared context category, says CAT2, owned by aEntUser1.

Test Step:

1. aEntUser1 logs into the system as an Enterprise end user.
2. Verify that aEntUser1 cannot delete CAT1.
3. Verify that aEntUser1 cannot delete CAT2.
4. aEntUser1 log out.

## Enterprise end user cannot delete context schemes used by a business context

Pre-condition: At least a user, say aEntUser1, with Enterprise end user role exists. There are business contexts created by various Enterprise tenant roles. One business context, says BC1, uses a private context scheme, CS1, owned by aEntUser1and another business context uses a shared context scheme, says CS2, owned by aEntUser1.

Test Step:

1. aEntUser1, logs into the system.
2. Verify that aEntUser1cannot delete CS1.
3. Verify that aEntUser1cannot delete CS2.
4. aEntUser1 log out.

## Enterprise end user cannot delete business context used by a profile BOD

Pre-condition: At least a user, say aEntUser1, with the Enterprise end user role exists. There are profile BODs created by various Enterprise tenant roles. One profile BOD uses a private business context, says BC1, owned by aEntUser1and another profile BOD uses a shared business context, says BC2, owned by aEntUser1.

Test Step:

1. An Enterprise end user, says aEntUser1, logs into the system.
2. Verify that aEntUser1cannot delete BC1.
3. Verify that aEntUser1cannot delete BC2.
4. aEntUser1log out.

# Enterprise admin user access right to SRT core functions

Requirement document reference: Section 3.2.2, 3.4.3.1

## Enterprise admin user’s authorized management of context categories

Pre-condition: At least two users, say ENTADx and ENTADy, with both enterprise admin users and OAGi developer roles exist. There are context categories created by various enterprise and OAGi users, some are created by ENTADx and ENTADy in the OAGi developer capacity, and some of which have been shared.

Test Step:

1. An enterprise admin user, ENTADx, logs into the system.
2. ENTADx creates new context categories, says CAT0, CAT1, and CAT4. (Assertion #1)
3. ENTADx edits CAT1. (Assertion #2)
4. The admin user logs out.
5. An enterprise end user, says ENTU1, logs into the system.
6. ENTU1 creates two new context categories, says CAT2 and CAT3.
7. ENTU1 log out.
8. Another enterprise admin user, says ENTADy, logs into the system.
9. ENTADy views the list of context categories.
10. Verify that enterprise users’ context categories, which have not been shared, are not visible to ENTADy in the list. Make sure that the verification statements include context categories that are created by ENTADx and ENTADy in the enterprise end user capacity. (Assertion #3)
11. Verify that enterprise users’ context categories, which have been shared, are visible to ENTADy in the list. (Assertion #4)
12. Verify that ENTADy can see the details of those visible shared enterprise users’ and OAGi users context categories but cannot make any change (verify with one created by other users and another one created by ENTADy in the OAGi developer capacity). (Assertion #5)
13. Verify that CAT0, CAT1, CAT2, and CAT3 are in the list. (Assertion #6)
14. Verify that ENTADy can see the details of CAT0, CAT1, CAT2, and CAT3. (Assertion #7)
15. ENTADy deletes CAT1 and CAT2. (Assertion #8)
16. ENTADy edits CAT3. (Assertion #9)
17. ENTADy edits CAT4. (Assertion #10)
18. ENTADy shares CAT0, CAT3, CAT4. (Assertion #11)
19. ENTADy creates CAT5.
20. ENTADy logs out.
21. ENTADx logs in again as an enterprise admin user.
22. ENTADx shares CAT5. (Assertion #11)
23. ENTADx logs out.
24. ENTADy logs in as an enterprise admin user.
25. Verify that ENTADy cannot edit or delete CAT3. (Assertion #12)
26. Verify that ENTADy cannot make change or delete CAT4 and CAT5. (Assertion #13)

Test Assertions covered by this test case:

1. The enterprise admin user can create a context category.
2. The enterprise admin user can edit a context category he created.
3. The enterprise admin user cannot see in the context category list, enterprise users’ private context categories.
4. The Enterprise admin user can see in the context category list, enterprise users’ private context categories.
5. The Enterprise admin user can see the details of enterprise user’s and OAGi user’s shared context categories.
6. The Enterprise admin user can see in the list context categories in any status created by others in the enterprise tenant.
7. The Enterprise admin user can see details of context categories in any status created by others in the enterprise tenant.
8. The Enterprise admin user can delete context categories created by others in the enterprise tenant.
9. The Enterprise admin user can edit a context category even if it is created by another enterprise end user.
10. The Enterprise admin user can edit a context category even if it is created by another enterprise admin.
11. The Enterprise admin user can share a context category created by himself or others in the enterprise tenant.
12. The Enterprise admin user can no longer change or delete a shared context category created by other Enterprise end users.
13. The Enterprise admin user can no longer change or delete a shared context category created by other enterprise admins.

## Enterprise admin user fails to create Context Category due to omitting name

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise admin user role.

Test Steps:

1. A user with the Enterprise admin user role, aEntUser1, logs in.
2. The Enterprise admin user accesses the context category management functionality.
3. The Enterprise admin user creates new context category, but omits name value.
4. Verify that context category creation failed.

## Enterprise admin user fails to edit Context Category due to omitting name

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise admin user role. Context Category, testCategory, already exists.

Test Steps:

1. A user with the Enterprise admin user role, aEntUser1, logs in.
2. The Enterprise admin user accesses the context category management functionality.
3. The Enterprise admin user removes the value for context category name.
4. Verify that context category edits failed.

## Enterprise admin user’s authorized management of context schemes

Pre-condition: At least two users, say ENTADx and ENTADy, with both enterprise admin users and OAGi developer roles exist. There are context categories created by various enterprise and OAGi users, some are created by ENTADx and ENTADy in the OAGi developer capacity, and some of which have been shared.

Test Step:

1. An enterprise admin user, ENTADx, logs into the system.
2. The admin user creates new context schemes (with some values), says CS0, CS1, and CS4. Verify that both context categories that are owned by OAGi tenant and that are shared by enterprise tenants are selectable for associating with the context schemes during the creation. Verify that enterprise tenants’ private context categories are not selectable. (Assertion #1, #2)
3. ENTADx edits CS1. (Assertion #3)
4. The admin user logs out.
5. A user, says ENTU1, logs into the system as an enterprise end user.
6. ENTU1 creates two new context schemes, says CS2 and CS3.
7. ENTU1 logs out.
8. Another user, ENTADy, logs into the system as an enterprise admin user.
9. ENTADy views the list of context schemes.
10. Verify that enterprise users’ context schemes, which have not been shared, are not visible to ENTADy in the list. Make sure that the verification statements include context schemes that are created by ENTADx and ENTADy in the enterprise end user capacity. (Assertion #4)
11. Verify that enterprise users’ context schemes, which have been shared, are visible to ENTADy in the list. (Assertion #5)
12. Verify that ENTADy can see the details of those visible shared enterprise users’ context schemes but cannot make any change (verify with one created by other users and another one created by ENTADy in the OAGi developer capacity). (Assertion #6)
13. Verify that CS0, CS1, CS2, and CS3 are in the list. (Assertion #7)
14. Verify that ENTADy can see the details of CS0, CS1, CS2, and CS3. (Assertion #8)
15. ENTADy deletes CS1 and CS2. (Assertion #9)
16. ENTADy edits CS3. (Assertion #10)
17. ENTADy edits CS4. (Assertion #11)
18. ENTADy shares CS0, CS3, CS4. (Assertion #12)
19. ENTADy creates CS5.
20. ENTADy shares CS5. (Assertion #12)
21. ENTADy logs out.
22. ENTADx logs in as an enterprise admin user.
23. Verify that ENTADx cannot edit or delete CS3. (Assertion #13)
24. Verify that ENTADx cannot make change or delete CS4 and CS5. (Assertion #14)
25. Verify that ENTADx cannot delete CS0. (Assertion #14)

Test assertions covered by this test case:

1. The enterprise admin user can create a context scheme.
2. Context categories belonging to the OAGi tenant and those shared by enterprise tenants shall be selectable by enterprise admin users for association with the context schemes and those private ones are not selectable.
3. The enterprise admin user can edit a context scheme he created.
4. The enterprise admin user cannot see in the context scheme list, enterprise users’ private context schemes.
5. The enterprise admin user can see in the context scheme list, enterprise users’ shared context schemes.
6. The enterprise admin user can see the details of enterprise user’s and OAGi user’s shared context schemes.
7. The enterprise admin user can see in the list context schemes in any status created by others in the enterprise tenant.
8. The enterprise admin user can see details of context schemes in any status created by others in the enterprise tenant.
9. The enterprise admin user can delete context schemes created by others in the enterprise tenant.
10. The enterprise admin user can edit a context scheme even if it is created by other enterprise end users.
11. The enterprise admin user can edit a context scheme even if it is created by other enterprise admin users.
12. The enterprise admin user can share a context scheme created by himself or others in the enterprise tenant.
13. The enterprise admin user can no longer change or delete a shared context schemes created by enterprise end users.
14. The enterprise admin user can no longer change or delete a shared context schemes created by another enterprise admin user.

## Enterprise admin user fails to create Context Scheme due to omitting required fields

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise admin user role.

Test Steps:

1. A user with the Enterprise admin user role, aEntUser1, logs in.
2. The Enterprise admin user accesses the context scheme management functionality.
3. The Enterprise admin user creates new context scheme, but omits some of required fields (e.g. Context Category, Name, Scheme ID, Agency ID or Version).
4. Verify that context category creation failed.

## Enterprise admin user fails to edit Context Scheme due to omitting required fields

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise admin user role. Context Scheme, testScheme, already exists.

Test Steps:

1. A user with the Enterprise admin user role, aEntUser1, logs in.
2. The Enterprise admin user accesses the context scheme management functionality.
3. The Enterprise admin user edits testScheme, but omits some of required fields (e.g. Context Category, Name, Scheme ID, Agency ID or Version).
4. Verify that context scheme edit failed.

## Enterprise admin user successfully adds Context Scheme Values

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise admin user role. Context Scheme, newScheme, already exists.

Test Steps:

1. A user with the Enterprise tenant admin user role, aEntUser1, logs in.
2. The Enterprise admin user accesses the context scheme management functionality.
3. The Enterprise admin user selects context scheme, newScheme.
4. The Enterprise admin user add several context scheme values to the newScheme.
5. Verify that context scheme is updated and values are added.

## Enterprise admin user fails to add Context Scheme Value due to omitting value

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise admin user role. Context Scheme, newScheme, already exists.

Test Steps:

1. A user with the Enterprise tenant admin user role, aEntUser1, logs in.
2. The Enterprise admin user accesses the context scheme management functionality.
3. The Enterprise admin user selects context scheme, newScheme.
4. The Enterprise admin user add context scheme value to the newScheme, but omits the value.
5. Verify that context scheme is not updated and values addition fails.

## Enterprise admin user successfully edits Context Scheme Values

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise admin user role. Context Scheme, newScheme, already exists.

Test Steps:

1. A user with the Enterprise tenant admin user role, aEntUser1, logs in.
2. The Enterprise admin user accesses the context scheme management functionality.
3. The Enterprise admin user selects context scheme, newScheme.
4. The Enterprise admin user edit context scheme values of the newScheme.
5. Verify that context scheme values are updated.

## Enterprise admin user fails to edit Context Scheme Value due to omitting value

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise admin user role. Context Scheme, newScheme, already exists.

Test Steps:

1. A user with the Enterprise tenant admin user role, aEntUser1, logs in.
2. The Enterprise admin user accesses the context scheme management functionality.
3. The Enterprise admin user selects context scheme, newScheme.
4. The Enterprise admin user add context scheme value to the newScheme, but omits the value.
5. Verify that context scheme is not updated and values addition fails.

## Enterprise admin user authorized access to code list management functions.

Pre-condition: At least two users, say ENTADx and ENTADy, with OAGi admin role and enterprise admin user role exists. OAGi tenant code lists exist in the system that are created by different OAGi developers and are in varying states. Private and shared enterprise tenants’ code lists exist in the system that are in varying states, some of which are owned by ENTADx and ENTADy in their enterprise admin user capacities

Test steps:

1. A user, ENTU1, logs into the system as an enterprise end user.
2. ENTU1 creates new code lists and made some customizations, says CL0 (extensible), CL1, CL2. The developer moves to publish CL0, but leave the rest in the Editing.
3. The end user logs out.
4. A user, ENTADx logs into the system as an enterprise admin user.
5. ENTADx creates a code list a code list without base, CL3 and add some values. (Assertion #1)
6. ENTADx creates a code list, CL4, using an extensible code list shared by an enterprise tenant as a base (use a code list not owned by the ENTADx user in the enterprise admin capacity). ENTADx creates a code list, CL5, using a published code list CL0 as a base. Verify that shared enterprise tenant’s code list that is not extensible can’t be used as a base. (Assertion #2)
7. ENTADx goes back and edit CL4 add/change some values. (Assertion #3)
8. ENTADx logs out.
9. A user, ENTADy, logs in as an enterprise admin user.
10. ENTADy edits CL4 and and then edits CL1. (Assertion #4)
11. ENTADy moves CL1 and CL4 to the Published state (Assertion #5)
12. ENTADy delete CL2 and CL3. (Assertion #6)
13. Verify that ENTADy can share CL0, while ENTADy cannot share CL1. (Assertion #7)
14. Verify that there is no code-list-freeing functionality provided to the ENTADy. (Assertion #8)
15. ENTADy goes to the view code list page.
16. Verify that there is no private code list from an enterprise tenant in the list of code lists. (Assertion #9)
17. Verify that enterprise users’ shared code lists are in the list. (Assertion #10)
18. Verify that ENTADy can click on the shared code lists and see their details but cannot make any change. (Assertion #11)
19. Verify that ENTADy can see the details of CL0, CL1, and CL4 but cannot make any change. (Assertion #12)
20. ENTADy logs out.

Assertions covered in this test case:

1. An enterprise admin user can create a code list without base.
2. An enterprise admin user can create a code list based on another code list. Only published, extensible enterprise tenant’s code lists and shared enterprise tenant’s code list can be used as the based code list.
3. An enterprise admin user can edit a code list he owns that is in the Editing state.
4. An enterprise admin user can edit a code list he didn’t create but belongs to the OAGi tenant and in the Editing state.
5. An enterprise admin user can change the state of a code list from Editing to Published.
6. An enterprise admin user can delete a code list that is in the Editing state even though he didn’t create it.
7. An enterprise admin user can share a code list in published state, but cannot share not published code list.
8. Verify that there is no make free functionality provided in the code list management.
9. An enterprise admin user cannot see in the list of code lists enterprise users’ private code lists in any state.
10. An enterprise admin user can see in the list of code lists enterprise users’ shared code lists.
11. An enterprise admin user can see the details of enterprise users’ shared code lists but cannot make any change.
12. An enterprise admin user can see the details of published code list belonging to any enterprise tenant’s user but cannot make any change.

## Enterprise admin user fails to create Code List due to omitting required fields

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise admin user role.

Test Steps:

1. A user with the Enterprise tenant admin user role logs in.
2. The Enterprise admin user accesses the code list management functionality end creates new code list without base.
3. The Enterprise admin user omits populating required fields (e.g. Name, Version) and tries to create code list.
4. Verify that code list creation fails.

## Enterprise admin user fails to edit Code List due to omitting required fileds

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise admin user role. Code List, testCodeList, already exists.

Test Steps:

1. A user with the Enterprise tenant end user role, aEntUser1, logs in.
2. The Enterprise end user accesses the code list management functionality.
3. The Enterprise end user finds testCodeList , removed values of required fields (Name, Version) and tries to update code list.
4. Verify that code list edits fail.

## Enterprise admin user fails to add Code List Value due to omitting code

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise admin user role. Code List, oacl\_TaxCode\_Extension, already exists.

Test Steps:

1. A user with the Enterprise tenant admin user role, aEntUser1, logs in.
2. The Enterprise admin user accesses the context scheme management functionality.
3. The Enterprise admin user selects code list, oacl\_TaxCode\_Extension.
4. The Enterprise admin user add code list value, but omits the code.
5. Verify that code list is not updated and values addition fails.

## Enterprise admin user fails to edit Code List Value due to omitting value

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise admin user role. Code List, oacl\_TaxCode\_Extension, already exists.

Test Steps:

1. A user with the Enterprise tenant end user role, aEntUser1, logs in.
2. The Enterprise end user accesses the code list management functionality.
3. The Enterprise end user selects code list, oacl\_TaxCode\_Extension.
4. The Enterprise end user edit some code list scheme value, but omits the code.
5. Verify that code list value update fails.

## Enterprise admin user authorized management of business contexts

Pre-condition: At least two users, say ENTADx and ENTADy, with both enterprise admin users and OAGi developer roles exist. There are context categories created by various enterprise and OAGi users, some are created by ENTADx and ENTADy in the OAGi developer capacity, and some of which have been shared.

Test Step:

1. An enterprise admin user, ENTADx, logs into the system.
2. The admin user creates new business contexts (with some context scheme values), says BC0, BC1, and BC4. Verify that context categories and context schemes that are owned by enterprise tenant and that are shared by enterprise tenants are selectable for context values while creating the business contexts. Verify that enterprise tenants’ private context categories and context schemes are not selectable. (Assertion #1, #2)
3. ENTADx edits BC1. (Assertion #3)
4. The admin user logs out.
5. An enterprise end user, says ENTU1, logs into the system.
6. ENTU1 creates two new business contexts, says BC2 and BC3.
7. ENTU1 logs out.
8. Another user, ENTADy, logs into the system as an OAGi admin.
9. ENTADy views the list of business contexts.
10. Verify that enterprise users’ business contexts, which have not been shared, are not visible to ENTADy in the list. Make sure that the verification statements include business context that are created by ENTADx and ENTADy in the OAGi developer capacity. (Assertion #4)
11. Verify that enterprise users’ business contexts, which have been shared, are visible to ENTADy in the list. (Assertion #5)
12. Verify that ENTADy can see the details of at least one of those visible shared enterprise users’ business contexts but cannot make any change (verify with one created by other users and another one created by ENTADy in the enterprise end user capacity). (Assertion #6)
13. Verify that BC0, BC1, BC2, and BC3 are in the list. (Assertion #7)
14. Verify that ENTADy can see the details of BC0, BC1, BC2, and BC3. (Assertion #8)
15. ENTADy deletes BC1 and BC2. (Assertion #9)
16. ENTADy edits BC3. (Assertion #10)
17. ENTADy edits BC4. (Assertion #11)
18. ENTADy shares BC0, BC3, BC4. (Assertion #12)
19. ENTADy creates BC5.
20. ENTADy shares BC5. (Assertion #12)
21. ENTADy logs out.
22. ENTADx logs in as an enterprise admin user.
23. Verify that ENTADx cannot make change and cannot delete BC3. (Assertion #13)
24. Verify that ENTADx cannot make change or delete BC4 and BC5. (Assertion #14)
25. Verify that ENTADx cannot change and cannot delete BC0. (Assertion #14)

Test assertions covered by this test case:

1. The enterprise admin user can create a business context.
2. Context categories and context schemes belonging to the enterprise tenant and those shared by enterprise tenants shall be selectable by admin users for specifying context values, and those private ones are not selectable.
3. The enterprise admin user can edit a business context he created.
4. The enterprise admin user cannot see in the business context list, enterprise users’ private business context.
5. The enterprise admin user can see in the business context list, enterprise users’ shared business context.
6. The enterprise admin user can see the details of enterprise user’s shared business context.
7. The enterprise admin user can see in the list business context in any status created by others in the enterprise tenant.
8. The enterprise admin user can see details of business context in any status created by others in the enterprise tenant.
9. The enterprise admin user can delete business context created by others in the enterprise tenant.
10. The enterprise admin user can edit a business context even if it is created by other enterprise end users.
11. The enterprise admin user can edit a business context even if it is created by other enterprise admins.
12. The enterprise admin can share a business context created by himself or others in the enterprise tenant.
13. The enterprise admin user can no longer change or delete a shared business context created by enterprise end users.
14. The enterprise admin user can no longer change or delete a shared business context created by other enterprise admin.

## Enterprise admin user fails to create Business Context due to omitting name

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise admin user.

Test Steps:

1. A user with the Enterprise tenant admin user role logs in.
2. The Enterprise admin user accesses the business context management functionality.
3. The Enterprise admin user creates new business context, but omits the name.
4. Verify that business context creation fails.

## Enterprise end user fails to edit Business Context due to omitting name

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise admin user role. Business Context, aBizCtx, already exists.

Test Steps:

1. A user with the Enterprise end user role, aEntUser1, logs in.
2. The Enterprise end user accesses the context category management functionality.
3. The Enterprise end user removes the value for business context name.
4. Verify that business context edits failed.

## Enterprise admin user authorized management of profile BODs

Pre-condition: At least two users, say ENTADX and ENTADy, with OAGi admin role and enterprise admin user role exists. Free profile BODs published by the OAGi tenant exist in the system. Private and shared OAGi tenant profile BODs exist in the system that are created by different OAGi developers and are in varying states, some of which are owned by ENTADX and ENTADy in their OAGi admin capacities. Private and shared enterprise tenants profile BODs exist in the system that are in varying states. There are OAGi tenant’s published code lists. There are shared enterprise tenants’ code lists.

1. An enterprise admin user, ENTADX, logs into the system.
2. The admin user creates new profile BODs and made some customizations, says PB0, PB1, and PB2. Verify that both business contexts that are owned by enterprise tenant and that are shared by OAGi tenant are selectable for associating with the profile BODs during the creation. Verify that enterprise tenants’ private business contexts are not selectable. (Assertion #1, #2)
3. ENTADX edits PB1. (Assertion #3)
4. ENTADX advances PB1 to the Candidate state. (Assertion #4)
5. Verify that ENTADX cannot edit PB1. (Assertion #5)
6. ENTADX moves PB1 back to the Editing state. (Assertion #6)
7. ENTADX deletes PB1. (Assertion #7)
8. ENTADX advances PB0 to the Candidate and then Published state. (Assertion #8)
9. Verify that ENTADX cannot edit or delete PB0. (Assertion #9)
10. ENTADX deletes PB2. (Assertion #10)
11. ENTADX creates PB4 and leave it in the Editing state, PB5 and advances it to the Candidate state, and PB6, PB7, and PB8 then advances them to the Published state.
12. ENTADX shares PB0. (Assertion #11)
13. Verify that ENTADX cannot share PB4 and PB5. (Assertion #11)
14. ENTADX logs out.
15. ENTADy logs in.
16. ENTADy shares PB7. (Assertion #13)
17. ENTADy cannot share PB4 and PB5. (Assertion #13)
18. ENTADy goes to the profile BOD list page.
19. Verify that enterprise users’ private profile BODs are not visible to ENTADy in the list including those owned by ENTADy in the enterprise user capacity. (Assertion #15)
20. Verify that enterprise users’ shared profile BODs are visible to ENTADy in the list. (Assertion #16)
21. Verify that PB0, PB3, PB4, PB5, PB6, PB7 are in the list. (Assertion #17)
22. Verify that ENTADy can see the details of at least one of those visible enterprise users’ shared profile BODs but cannot make any change. Verify that ENTADy cannot drill down to make change to the associated business context and context categories or see and edit the whole context scheme (the user should be able to see the context scheme values used in the business context). (Assertion #18)
23. Verify that ENTADy can see the details of PB8 but cannot make any change (because it is published even though it is owned by the admin user). (Assertion #19)
24. Verify that ENTADy cannot see, edit the detail, nor change state and ownership of PB4 (which is still in the Editing state and owned by the admin user). (Assertion #20)
25. Verify that ENTADy can see the detail of PB5 (which is in the Candidate state) but cannot make any change. (Assertion #21)
26. ENTADy logs out.
27. A user, ENTU1, logs in as an enterprise end user.
28. ENTU1creates PB9, PB10, and PB11 and move PB10 to the Candidate state and PB11 to the Published state.
29. ENTU1 logs out.
30. ENTADy logs in as an enterprise admin user.
31. Verify that ENTADy can see the details of PB11 but cannot make any change (because it is published even though it is owned by the ENTADX). (Assertion #22)
32. Verify that ENTADy cannot see the detail nor making any change to PB9 (because it is still in the Editing state and is owned by another admin developer, ENTADX). (Assertion #23)
33. Verify that ENTADy can see (review) the detail of PB10 (which is in the Candidate state) but cannot make any change.
34. ENTADy logs out.
35. ENTADX logs in as an enterprise admin user.
36. ENTADX opens PB4 for editing.
37. ENTADX assigns a primitive restriction using built-in types. Verify that this works. (Assertion #24)
38. ENTADX assigns a code list to a field for primitive restriction. Verify that published code lists belonging to the OAGi tenant and shared code lists belonging to enterprise tenants are available for selection. (Assertion #25)
39. Verify that ENTADX can successfully change the ownership of PB4 to ENTU1. (Assertion #27)
40. Verify that ENTADX can successfully change the ownership of PB5 to ENTADy. (Assertion #28)
41. ENTADX logs out.
42. ENTU1 logs in as an enterprise end user.
43. ENTU1 transfer the ownership of PB4 to ENTADX.
44. ENTU1 logs out.
45. ENTADX logs in as an enterprise admin user.
46. Verify that ENTADX can successfully transfer the ownership of PB4 to ENTADy. (Assertion #29)
47. Verify that ENTADX can successfully transfer the ownership of PB9 to ENTADy. (Assertion #30.1)
48. Verify that ENTADX can successfully transfer the ownership of PB5 to ENTU1. (Assertion #30.2)
49. ENTADX goes to the copy profile BOD page which brings up the list or profile BODs allowed for copying.
50. ENTADX logs out.
51. ENTU1 logs in as an enterprise end user.
52. ENTU1 transfers the ownership of PB5 to ENTADX.
53. ENTU1 logs out.
54. ENTADy logs in as an enterprise admin user.
55. Verify that enterprise users’ private profile BODs are not visible to ENTADy in the list including those owned by ENTADy in the enterprise end user capacity. (Assertion #31)
56. Verify that enterprise users’ shared profile BODs are visible to ENTADy in the list. (Assertion #32)
57. Verify that ENTADy can see and choose in the list PB5, PB8, PB10, and PB11 for copying. (Assertion #33)
58. Verify that ENTADy can see and choose in the list PB0, PB6, and PB7 for copying. (Assertion #34)
59. ENTADy transfers the ownership of PB9 to ENTU1and PB4 to ENTADX.
60. Verify that ENTADy cannot see and choose PB9 for copying (it is ok to fail this b/c this is a known bug in the version given to JT). (Assertion #35)
61. Verify that ENTADy cannot see and choose PB4 for copying (it is ok to fail this b/c this is a known bug in the version given to JT). (Assertion #36)
62. ENTADy goes to the expression generation page.
63. Verify that ENTADy cannot see and choose any enterprise users’ private profile BODs for generating an expression including those owned by ENTADy in the enterprise admin user’s capacity. (Assertion #37)
64. Verify that ENTADy can see and choose enterprise tenant’s shared profile BODs for generating an expression. (Assertion #38)
65. Verify that ENTADy can see and choose PB5, PB8, PB10, and PB11 for generating an expression. (Assertion #39)
66. Verify that ENTADy can see and choose in the list PB0, PB6, and PB7 for generating an expression. (Assertion #40)
67. Verify that ENTADy cannot see and choose PB9 for generating an expression (it is ok to fail this b/c this is a known bug in the version given to JT). (Assertion #41)
68. Verify that ENTADy cannot see and choose PB4 for generating an expression (it is ok to fail this b/c this is a known bug in the version given to JT). (Assertion #42)
69. ENTADy logs out.

Assertions covered in this test case:

1. An enterprise admin user can create a profile BOD.
2. Business contexts belonging to the enterprise tenant and those shared by enterprise tenants and OAGi tenant shall be selectable by enterprise admins for associating with the profile BOD during its creation.
3. An enterprise admin user can edit a profile BOD it owns.
4. An enterprise admin user can change the state of a profile BOD he owns from Editing to Candidate.
5. An enterprise admin user cannot edit his own profile BOD in Candidate state.
6. An enterprise admin user can change the state of his own profile BOD from Candidate back to Editing.
7. An enterprise admin user can delete his own profile BOD in the Candidate state.
8. An enterprise admin user can change the state of a profile BOD he owns from Candidate to Published.
9. An enterprise admin user cannot make any change to an owned profile BOD that is in the Published state.
10. An enterprise admin user can delete his own profile BOD that is in the Editing state.
11. An enterprise admin user can share an owned profile BOD ONLY in the Published state.
12. remove later
13. An enterprise admin user can share an unowned profile BOD ONLY in the Published state.
14. remove later
15. An enterprise admin user cannot see, in the list of profile BODs, enterprise users’ private profile BODs in any state including those owned by him in enterprise tenant roles.
16. An enterprise admin user can see, in the list of profile BODs, enterprise users’ shared profile BODs.
17. An enterprise admin user can see, in the list of profile BODs, enterprise tenant’s profile BODs in any state.
18. An enterprise admin user can see the details of enterprise users’ shared profile BODs including its business context but cannot make any change to the profile BODs and any associated information, and cannot drill down to see the whole context scheme values.
19. An enterprise admin user can see the details of published profile BODs owned by another admin user, but cannot make any change.
20. An enterprise admin user cannot see the details of or make any change to profile BODs in the Editing state owned by another enterprise admin user.
21. An enterprise admin user can see the details of the candidate profile BODs owned by an admin user, but cannot make any change.
22. An enterprise admin user can see the details of published profile BODs owned by an another enterprise end user, but cannot make any change.
23. An enterprise admin user cannot see the details of or make any change to profile BODs in the Editing state owned by another enterprise end user.
24. Built-in type primitive restriction works for the enterprise admin.
25. Enterprise tenant’s published code lists and enterprise tenants’ shared code lists are available for primitive restriction using a code list.
26. An Enterprise admin user can see the details of the candidate profile BODs owned by an another enterprise end user, but cannot make any change.
27. An enterprise admin user can transfer the ownership of his own profile BODs in Editing state to another enterprise end user.
28. An eEnterprise admin user can transfer the ownership of his own profile BODs in Candidate state to another enterprise admin user.
29. An enterprise admin user can transfer the ownership of his own profile BODs in Editing state to an enterprise admin user.
30. An enterprise admin user can transfer the ownership of a profile BOD in the Editing or Candidate state belonging to another user in the enterprise tenant to another user in the enterprise tenant.
    1. An enterprise admin user can transfer the ownership of a profile BOD in the Editing state belonging to an enterprise end user to an enterprise admin user.
    2. An enterprise admin user can transfer the ownership of a profile BOD in the Candidate state belonging to another enterprise admin user to an enterprise end user.
31. An enterprise admin user cannot copy an enterprise tenant’s private profile BOD, particularly even the one he owns in an enterprise tenant role.
32. An enterprise admin user can copy an enterprise tenant’s shared profile BOD.
33. An enterprise admin user can copy profile BODs that belong to other enterprise end users and admin developers and that are in candidate or published states.
34. An enterprise admin user can copy free profile BODs.
35. An enterprise admin user cannot copy profile BODs that belong to another Enterprise end user and that are in editing states.
36. An Enterprise admin user cannot copy profile BODs that belong to an Enterprise admin user and that are in editing states.
37. An Enterprise admin user cannot generate expressions of an enterprise users’ private profile BODs.
38. An enterprise admin user can generate expressions of an enterprise users’ shared profile BODs.
39. An enterprise admin user can generate expressions of profile BODs that belong to other Enterprise end users and admin developers and that are in candidate or published states.
40. An enterprise admin user can generate expressions from free profile BODs.
41. An enterprise admin user cannot generate expressions of profile BODs that belong to another enterprise end user and that are in editing states.
42. An enterprise admin user cannot generate expressions of profile BODs that belong to an Enterprise admin user and that are in editing states.

## Enterprise admin user successfully edits profile BOD

Requirement document reference: Section 3.2.2, 2.1.2

Pre-condition: A user with the username, aEntUser1, already exists with Enterprise admin user role. Acknowledge BOM profile BOD for newBizCtx business context already exists.

Test Steps:

1. A user with the Enterprise tenant admin user role, aEntUser1, logs in.
2. The Enterprise admin user accesses the profile BODs functionality.
3. The Enterprise admin user finds Acknowledge BOM profile BOD for the newBizCtx business context.
4. The Enterprise admin user arbitrarily updates profile BOD by checking components that should be added/removed from the profile and updates the profile BOD.
5. Verify that profile BOD is updated.

## Enterprise admin user authorized access to CC management functions with localized extension

Pre-condition: At least two users, say ENTADx and ENTADy, with both enterprise admin user and OAGi developer roles exist. There are profile BOD extensions created by various enterprise users[[13]](#footnote-13), some are created by ENTU1 and ENTU2 in the OAGi developer role capacity, and some of which have been shared.

Test Steps:

1. An Enterprise end user, ENTU1, logs into the system.
2. ENTU1 creates a new profile BOD, PB0.
3. ENTU1 makes a localized extension to PB0. This results in, says, User Extension Group ACC, UEGACC0, and corresponding ASCCP0 and ASCC0 (these two are supposed to be hidden on the CC Management page by default logic).
4. ENTU1 adds some ASCCs and BCCs to the extension, i.e., to UEGACC0.
5. ENTU1 advances the PB0 extension to the Published state.
6. ENTU1 repeats step 2 to 4 two times resulting PB1, PB2, and corresponding UEGACC1, ASCCP1, ASCC1, and UEGACC2, ASCCP2, ASCC2, respectively. Use a commonly used component like ApplicationArea’s Extension for PB1.
7. ENTU1 advances UEGACC2 to the Candidate state.
8. ENTU1 logs out.
9. ENTADx logs into the system as an enterprise admin user.
10. ENTADx creates new profile BODs, PB3, PB4, PB5, and PB6.
11. ENTADx makes some localized extensions to PB3, PB4, PB5 and PB6 that results in UEGACC3, ASCCP3, ASCC3, UEGACC4, ASCCP4, ASCC4, UEGACC5, ASCCP5, ASCC5, and UEGACC6, ASCCP6, ASCC6. Some ASCCs and BCCs are added to these extensions. Use a commonly used component like Party’s Extension for PB5. Verify that no ASCCP belonging to enterprise tenants ever shows up for selection when adding an ASCC (use particularly the ASCCP that belongs to ENTADx in the enterprise end user capacity in the verification). Also, verify that user extension group ASCCPs are not available for selection. (Assertion #1, #2.1, #2.2)
12. ENTADx removes an ASCC from the PB3 extension. (Assertion #2.3)
13. ENTADx removes a BCC from the PB3 extension. (Assertion #2.4)
14. ENTADx advances the PB5 extension to the Candidate state. (Assertion #2.5)
15. ENTADx advances the PB3 extension to the Published state. (Assertion #2.6)
16. Verify that ENTADx cannot make any change to the PB5 extension except changing the state back to Editing (note keep the PB5 extension in the Candidate state after this step). (Assertion #2.7)
17. ENTADx creates a profile BOD PB7. Makes an extension to PB7 to the same component as PB5. Here actually no user extension group created (so there is no UEGACC7) (Assertion #2.8)
18. ENTADx opens PB4. Try to make an extension to the same component as PB1. Verify that the application does not allow this. (Assertion #2.9.1)
19. ENTADx logs out.
20. ENTADy logs in as an Enterprise admin user.
21. ENTADy creates a profile BOD, PB8.
22. ENTADy tries to make an extension in PB8 to the same component as PB5. Verify that the application does not allow this. (Assertion #2.9.2)
23. ENTADy logs out.
24. ENTADx logs in as an Enterprise admin user.
25. ENTADx opens the CC Management page.
26. Verify that ENTADx cannot see in the list any user extension group ACC belonging to ENTADx in the enterprise end user capacity. (Assertion #3.1)
27. Verify that ENTADx can see in the list UEGACC1. (Assertion #3.2.1)
28. ENTADx logs out.
29. ENTADy logs in as an Enterprise admin user.
30. ENTADy goes to the CC Management page.
31. Verify that ENTADy can see in the list UEGACC5. (Assertion #3.2.2)
32. Verify that ENTADy can see the details of UEGACC2 but cannot make any change. (Assertion #3.3.1)
33. Verify that ENTADy can see the details of UEGACC3 but cannot make any change. (Assertion #3.3.2)
34. Verify that ENTADy cannot see the details of the UEGACC6. (Assertion #3.3.3)
35. ENTADy logs out.
36. ENTADx logs in as an Enterprise admin user and goes to the CC Management page.
37. ENTADx open UEGACC4 to add a new ASCC. Verify that no ASCCP belonging to other enterprise tenants or OAGi tenant ever shows up for selection when adding an ASCC (use particularly the ASCCP that belongs to ENTADx in the OAGI user capacity in the verification). Also, verify that user extension group ASCCPs are not available for selection. (Assertion #3.4)
38. ENTADx adds a new BCC to UEGACC4. (Assertion #3.5)
39. ENTADx removes an ASCC from UEGACC4. (Assertion #3.6)
40. ENTADx removes a BCC from UEGACC4. (Assertion #3.7)
41. ENTADx moves UEGACC4 to the Candidate state. (Assertion #3.8)
42. ENTADx moves UEGACC4 back to the Editing state. (Assertion #3.9)
43. ENTADx moves UEGACC4 to the Candidate and then Published state. (Assertion #3.10)
44. Verify that there is no sharing capability in the CC Management. (Assertion #4)
45. ENTADx change the ownership of UEGACC8 to ENTU1. Verify that no user without an enterprise tenant role is available for selection; and verify that the transfer is successful. (Assertion #5.1)
46. Verify that ENTADx cannot change the ownership of UEGACC5. (Assertion #5.2)
47. Verify that ENTADx can successfully change the ownership of UEGACC8 to ENTADx. (Assertion #5.3)
48. ENTADx changes the state of UEGACC5 to the Editing state.
49. ENTADx transfers the ownership of UEGACC5 to ENTADy. (Assertion #5.4)
50. ENTADx logs out.
51. ENTADy logs in as an Enterprise admin user.
52. ENTADy change the state of UEGACC5 to the Published state.
53. ENTADy logs out.
54. ENTADx logs in as an Enterprise admin user.
55. ENTADx opens PB7.
56. Verify that ENTADx gets a notification to up take the extension when opening the PB7 for edit. (note this is because PB7 uses the same extension as PB5). (Assertion #6.1)
57. ENTADx opens PB3.
58. Verify that ENTADx gets a notification to up take the extension when opening the PB3 for edit. (Assertion #6.2)
59. ENTADx logs out.

Assertions covered in this test case:

1. Enterprise admin user can creates a localized profile BOD extension.
2. Enterprise admin user can manage a localized profile BOD extension (through the localized profile BOD extension menu).
   1. An Enterprise admin user can add a new ASCC to an owned profile BOD extension in the Editing state using ASCCP belonging to the enterprise tenant but not ASCCP belonging to other enterprise tenants or OAGi tenant in any state. Also, user extension group ASCCPs shall not be available for selection.
   2. An Enterprise admin user can add a new BCC to an owned profile BOD extension in the Editing state.
   3. An Enterprise admin user can remove an ASCC from an owned profile BOD extension in the Editing state.
   4. An Enterprise admin user can remove an BCC from an owned profile BOD extension in the Editing state.
   5. An Enterprise admin user can change the state of an owned profile BOD extension from Editing to Candidate.
   6. An Enterprise admin user can change the state of an owned profile BOD extension from Candidate to Published.
   7. An Enterprise admin user can change the state of an owned profile BOD extension from Candidate back to Editing.
   8. An Enterprise admin user can open up the extension to the same component that is still in the Editing state through another profile BOD.
   9. The system shall not allow an Enterprise admin user to create an extension to a component being extended by another enterprise tenant’s user.
      1. The system shall not allow an Enterprise admin user to create an extension to a component being extended by another Enterprise end user in the Editing state.
      2. The system shall not allow an Enterprise admin user to create an extension to a component being extended by another Enterprise admin user in the Candidate state.
3. Enterprise admin user can manage user extension group ACCs on the CC Management page.
   1. An Enterprise admin user cannot see in the CC list any user extension group ACC belonging to other enterprise tenants or OAGi tenant (test particularly on those ACCs owned by the Enterprise admin user in the OAGi tenant’s capacity).
   2. An Enterprise admin user can see in the CC list user extension group ACCs belonging to others in the enterprise tenant. The ACCs can be in any state.
      1. An Enterprise admin user can see in the list user extension group ACCs in the Editing state belonging to another Enterprise end user.
      2. An Enterprise admin user can see in the list user extension group ACCs in the Candidate state belonging to another Enterprise admin user.
   3. An Enterprise admin user can see the detail of user extension group ACCs belonging to others in the enterprise tenant only in the Candidate or Published state but cannot make any change.
      1. An Enterprise admin user can see the detail of a user extension group ACC belonging to another Enterprise end user in the Candidate state but cannot make any change.
      2. An Enterprise admin user can see the detail of a user extension group ACC belonging to another Enterprise admin user in the Published state but cannot make any change.
      3. An Enterprise admin user cannot see the detail of a user extension group ACC belonging to another Enterprise admin user in the Editing state.
   4. An Enterprise admin user can add a new ASCC to an owned ACC in the Editing state using ASCCP belonging to the enterprise tenant but not ASCCP belonging to other enterprise ot OAGi tenants in any state. User extension group ASCCPs shall not be available for selection.
   5. An Enterprise admin user can add a new BCC to an owned ACC in the Editing state.
   6. An Enterprise admin user can remove an ASCC from an owned ACC in the Editing state.
   7. An Enterprise admin user can remove an BCC from an owned ACC in the Editing state.
   8. An Enterprise admin user can change the state of an ACC it owns from Editing to Candidate.
   9. An Enterprise admin user can change the state of an ACC it owns from Candidate back to Editing.
   10. An Enterprise admin user can change the state of an ACC it owns from Candidate to Published.
4. There is no sharing capability in the CC Management.
5. An Enterprise admin user can transfer the ownership of the user extension group ACC belonging to himself or other enterprise tenant’s users. The admin can transfer the ownership to any user with a enterprise tenant’s role. The ACC has to be in the Editing state.
   1. The Enterprise admin user can transfer the ownership of a user extension group ACC in the Editing state from another Enterprise admin user to another Enterprise end user and not a user without an enterprise tenant’s role.
   2. The Enterprise admin user cannot transfer the ownership of his own user extension group ACC in the Candidate state.
   3. The Enterprise admin user can transfer the ownership of a user extension group ACC in the Editing state from another Enterprise end user to another Enterprise admin user.
   4. The Enterprise admin user can transfer the ownership of his user extension group ACC in the Editing state to another Enterprise admin user.
6. The system shall make a notification to the user while opening up a profile BOD, to which a related user extension group ACC has been published.
   1. The system shall notify the Enterprise admin user when another Enterprise admin user published a user extension group ACC relevant to his profile BOD.
   2. The system shall notify the Enterprise admin user while opening a profile BOD when his user extension group ACC relevant to the profile BOD has been published.

## Enterprise admin user developer authorized access to CC management functions with the global extension

Repeat Test Case 12.22 but with the global extension.

## Enterprise admin user cannot delete context categories used by a context scheme

Pre-condition: At least a user, say aEntUser1, with Enterprise admin user role exists. There are private context schemes created by various Enterprise tenant roles[[14]](#footnote-14). One context scheme uses a private context category, says CAT1, owned by aEntUser1 and another context scheme uses a shared context category, says CAT2, owned by aEntUser1.

Test Step:

1. aEntUser1 logs into the system as an Enterprise admin user.
2. Verify that aEntUser1 cannot delete CAT1.
3. Verify that aEntUser1 cannot delete CAT2.
4. aEntUser1 log out.

## Enterprise admin user cannot delete context schemes used by a business context

Pre-condition: At least a user, say aEntUser1, with Enterprise admin user role exists. There are business contexts created by various Enterprise tenant roles. One business context, says BC1, uses a private context scheme, CS1, owned by aEntUser1and another business context uses a shared context scheme, says CS2, owned by aEntUser1.

Test Step:

1. aEntUser1, logs into the system.
2. Verify that aEntUser1cannot delete CS1.
3. Verify that aEntUser1cannot delete CS2.
4. aEntUser1 log out.

## Enterprise admin user cannot delete business context used by a profile BOD

Pre-condition: At least a user, say aEntUser1, with the Enterprise admin user role exists. There are profile BODs created by various Enterprise tenant roles. One profile BOD uses a private business context, says BC1, owned by aEntUser1and another profile BOD uses a shared business context, says BC2, owned by aEntUser1.

Test Step:

1. An Enterprise admin user, says aEntUser1, logs into the system.
2. Verify that aEntUser1cannot delete BC1.
3. Verify that aEntUser1cannot delete BC2.
4. aEntUser1log out.

# Root tenant access right

Requirement document reference: Section 4.5, 3.4.3.5, 7

## Root tenant access right to SRT functions

Pre-condition: A user root role, say ROOTx, exist in root tenant. There are other users with various roles in other tenants that have created content and owns it, while some of the content is shared.

Test Step:

1. An enterprise admin user, ENTADx, logs into the system.
2. Verify that ENTADx cannot assign any enterprise role to ROOTx user. (Assertion #1)
3. ENTADx logs out.
4. An OAGi admin developer, OAGADx, logs into the system.
5. Verify that ENTADx cannot assign any enterprise role to ROOTx user. (Assertion #1)
6. OAGADx logs out.
7. A root role user, ROOTx, logs into the system.
8. ROOTx views the list of various content (context categories, context schemes, code lists, profile BODs, etc).
9. Verify that content which have not been shared is not visible to ROOTx in the list. (Assertion #2)
10. ROOTx goes to the expression generation page.
11. Verify that ROOTx can see and choose shared profile BODs for generating an expression. (Assertion #3)
12. Verify that ROOTx can see and choose free profile BODs for generating an expression. (Assertion #4)
13. ROOTx goes to the CC management page.
14. ROOTx views the list of CC components.
15. Verify that ROOTx can only see CC components released/owned by OAGi tenant and cannot make any changes. (Assertion #5)
16. ROOTx goes to the ownership management page.
17. Verify that ROOTx cannot change the ownership of data to the user not from the same tenant as the owner of the data. (Assertion #6)

Test Assertions covered by this test case:

1. The user with root role cannot have any other role.
2. The user with root role can view shared content.
3. The user with root role can generate expressions from shared profile BOD.
4. The user with root role can generate expressions from free profile BOD.
5. The user with root role can view OAGi released CC content.
6. The user with root role can change the ownership of data to the users from the same tenant as original owner of data.

## Root tenant user management

Pre-condition: A user root role, say ROOTx, exist in root tenant. There are other users with various roles in other tenants that have created content and owns it, while some of the content is shared.

Test Step:

1. A user, ROOTx, logs in as a root user.
2. ROOTx creates a new enterprise tenant, ENTa, filling in all the information except the title field with a valid email address. (Assertion #1)
3. Verify that the verification email is sent out to the indicated email address. (Assertion #2)
4. ROOTx creates a new enterprise tenant, ENTb, filling in all the information except the email address. Verify that the system does not allow that. (Assertion #3)
5. ROOTx goes back and fill in the email address field but with an invalid format. Verify that the system does not allow that. (Assertion #4)
6. ROOTx updates some information about ENTa. (Assertion #5).
7. ROOTx creates a new user, ENTaADMIN. (Assertion #6)
8. Verify that ROOTx can assign an ENTa admin role to ENTaADMIN. (Assertion #7)
9. ROOTx creates a new user, ENTaUSER. Verify that ROOTx can assign an ENTa end user role to ENTaUSER. (Assertion #8)
10. ROOTx creates a new user, OAGIaDEV, in OAGi tenant. Verify that ROOTx can assign an OAGi developer role to OAGIaDEV. (Assertion #9)
11. ROOTx creates a new user, OAGIaADM, in OAGi tenant. Verify that ROOTx can assign an OAGi admin role to OAGIaADM. (Assertion #10)
12. Verify that ROOTx can revoke roles assigned in previous five steps (if necessary, add other roles to the user first). (Assertion #11)
13. Root user edits OAGIaADM info. Verify that email notification is sent to the affected user. (Assertion #12, 14)
14. Root user edits its own info. Verify that email notification is sent to the root user. (Assertion #13, 14)
15. ROOTx logs out.

Test Assertions covered by this test case:

1. Root user can create a new enterprise tenant with the optional title element left off.
2. The email address verification email shall be sent out. (Optional)
3. The email address of the enterprise tenant cannot be left out. (Optional)
4. The system does not allow an invalid email address format.
5. Root user can update enterprise tenant information.
6. Root user can create user.
7. Root user can assign the admin role of the new enterprise tenant to a user.
8. Root user can assign the end user role of the new enterprise tenant to a user.
9. Root user can assign the OAGi admin role to a user.
10. Root user can assign the OAGi developer role to a user.
11. Root user can revoke enterprise admin and end user roles, and OAGi admin and developer roles from the users.
12. Root user can edit info of other users.
13. Root user can edit its own info.
14. The user info change notification email shall be sent out.

# Data retention

Requirement document reference: Section 5.1.1

## Enterprise tenant data retention

Pre-condition: An enterprise tenant, ENTa, exists. ENTa has at least one end user. There must be some profile BODs, profile BOD extensions, and code lists created by ENTa’s users some of which are shared.

Test steps:

1. A user, OAGADx, logs in as an OAGi admin.
2. OAGADx deactivates an enterprise tenant, ENTa.
3. OAGADx reactivates ENTa.
4. OAGADx logs out.
5. A user, ENTaUser1, logs in as an ENTa end user.
6. ENTaUser1 goes to the profile BOD list page.
7. Verify that ENTaUser1 can see and edit his profile BODs created before ENTa was deactivated. (Assertion #1.1)
8. ENTaUser1 goes to the CC Management page.
9. Verify that ENTaUser1 can see the user extension group CCs created by ENTa before it was deactivated. (Assertion #1.2)
10. ENTaUser1 goes to the View Code List page.
11. Verify that ENTaUser can see code lists created by ENTa before it was deactivated. (Assertion #1.3)

Test assertions:

1. When an enterprise tenant is reactivated, data created previously are accessible and manageable again
   1. Profile BODs are retained.
   2. User extension CCs are retained.
   3. Code lists are retained.
2. When an enterprise tenant has been deactivated for the time period specified in the data retention policy or agreement, the tenant’s private data is removed when the purge process is run. However, tenant’s shared data must not be removed. (Need to ask JT if this feature has been implemented)

## If the user is reinstated with the previously revoked role, he can access and manage he data he still owns again

Pre-condition: An enterprise tenant, ENTa, exists. It has at least one end user and two admin users. These users have profile BODs data and extensions in various states. At least one OAGi developer and two OAGi admin users exist. These OAGi developers have profile BODs data and extensions in various states.

Test steps:

1. A user, ENTaAdmin1, logs in as an admin of ENTa tenant.
2. ENTaAdmin1 revokes the ENTa end user role from the user, ENTaUser1.
3. ENTaAdmin 1 revokes the ENTa admin user role from the user, ENTaAdmin2.
4. ENTaAdmin1 logs out.
5. ENTaUser1 logs in as an ENTa end user role. This should fail.
6. ENTaAdmin2 logs in as an ENTa admin role. This should fail.
7. ENTaAdmin1 logs in as an admin of ENTa tenant.
8. ENTaAdmin1 reassign ENTa end user role to ENTaUser1 again.
9. ENTaAdmin1 reassign ENTa admin user role to ENTaAdmin2 again.
10. ENTaAdmin1 logs out.
11. ENTaUser1 logs in as an ENTa end user.
12. Verify that ENTaUser1 can access and manage profile BODs and CCs that he owns and are in the appropriate state. (Assertion #1)
13. ENTaUser1 logs out.
14. ENTaAdmin2 logs in as an ENTa admin.
15. Verify that ENTaAdmin2 can access and manage profile BODs and CCs that he owns and are in the appropriate state. (Assertion #1)
16. ENTaAdmin2 logs out.
17. A user, OAGADx, logs in as an OAGi admin.
18. OAGADx revokes the OAGi developer role from the user, OAGD1.
19. OAGADx revokes the OAGi admin role from the user, OAGADy.
20. OAGADx logs out.
21. OAGD1 logs in as the OAGi developer role. This should fail.
22. OAGADy logs in as the OAGi admin role. This should fail.
23. OAGADx logs in as an OAGi admin.
24. OAGADx reassign the OAGi developer role to OAGD1again.
25. OAGADx reassign the OAGi admin role to OAGADy again.
26. OAGADx logs out.
27. OAGD1 logs in as an OAGi developer.
28. Verify that OAGD1 can access and manage profile BOD and CC data that he owns and are in the appropriate state. (Assertion #2)
29. OAGD1 logs out.
30. OAGDy logs in as an OAGi admin.
31. Verify that OAGADy can access and manage profile BOD and CC data that he owns and are in the appropriate state. (Assertion #2)
32. OAGADy logs out.

Test assertions:

1. When a user is reinstated a role within an enterprise tenant, he can still access and manage data he previously owned and still owns in that tenant’s capacity.
2. When a user is reinstated a role within an OAGi tenant, he can still access and manage data he previously owned and still owns in OAGi tenant capacity.

# Password Reset

Requirement document reference: Section 5.7

## Successful password reset by OAGi admin developer

Requirement document reference: Section 5.7

Pre-condition: A user with the username, aNewUser1, already exists with the OAGi admin developer role.

Test Steps:

1. The user, aNewUser1, logs in.
2. The user accesses the password reset functionality.
3. The user enters a new valid password and reset it.
4. Verify that the user’s password is reset (logout and try to log in using the new password).

## Successful password reset by OAGi developer

Requirement document reference: Section 5.7

Pre-condition: A user with the username, aNewUser1, already exists with the OAGi developer role.

Test Steps:

1. The user, aNewUser1, logs in.
2. The user accesses the password reset functionality.
3. The user enters a new valid password and reset it.
4. Verify that the user’s password is reset (logout and try to log in using the new password).

## Successful password reset by Enterprise tenant admin user

Requirement document reference: Section 5.7

Pre-condition: A user with the username, aNewUser1, already exists with the Enterprise tenant admin role.

Test Steps:

1. The user, aNewUser1, logs in.
2. The user accesses the password reset functionality.
3. The user enters a new valid password and reset it.
4. Verify that the user’s password is reset (logout and try to log in using the new password).

## Successful password reset by Enterprise tenant end user

Requirement document reference: Section 5.7

Pre-condition: A user with the username, aNewUser1, already exists with the Enterprise tenant end user role.

Test Steps:

1. The user, aNewUser1, logs in.
2. The user accesses the password reset functionality.
3. The user enters a new valid password and reset it.
4. Verify that the user’s password is reset (logout and try to log in using the new password).

## Successful password reset by Free user

Requirement document reference: Section 5.7

Pre-condition: A user with the username, serm, already exists with the Free user role.

Test Steps:

1. The user, serm, logs in.
2. The user accesses the password reset functionality.
3. The user enters a new valid password and reset it.
4. Verify that the user’s password is reset (logout and try to log in using the new password).

## Successful password reset by Root user

Requirement document reference: Section 5.7

Pre-condition: A user with Root role already exists.

Test Steps:

1. The user logs in.
2. The user accesses the password reset functionality.
3. The user enters a new valid password and reset it.
4. Verify that the user’s password is reset (logout and try to log in using the new password).

## Fail password reset by OAGi admin developer due the omission

Requirement document reference: Section 5.7

Pre-condition: A user with the username, aNewUser1, already exists with the OAGi admin developer role.

Test Steps:

1. The user, aNewUser1, logs in.
2. The user accesses the password reset functionality.
3. The user omits the new password and tries to reset it.
4. Verify that the system fails the user password reset.

## Fail password reset by OAGi developer due the omission

Requirement document reference: Section 5.7

Pre-condition: A user with the username, aNewUser1, already exists with the OAGi developer role.

Test Steps:

1. The user, aNewUser1, logs in.
2. The user accesses the password reset functionality.
3. The user omits the new password and tries to reset it.
4. Verify that the system fails the user password reset.

## Fail password reset by Enterprise tenant admin user due the omission

Requirement document reference: Section 5.7

Pre-condition: A user with the username, aNewUser1, already exists with the Enterprise tenant admin role.

Test Steps:

1. The user, aNewUser1, logs in.
2. The user accesses the password reset functionality.
3. The user omits the new password and tries to reset it.
4. Verify that the system fails the user password reset.

## Fail password reset by Enterprise tenant end user due the omission

Requirement document reference: Section 5.7

Pre-condition: A user with the username, aNewUser1, already exists with the Enterprise tenant end user role.

Test Steps:

1. The user, aNewUser1, logs in.
2. The user accesses the password reset functionality.
3. The user omits the new password and tries to reset it.
4. Verify that the system fails the user password reset.

## Fail password reset by Free user due the omission

Requirement document reference: Section 5.7

Pre-condition: A user with the username, serm, already exists with the Free user role.

Test Steps:

1. The user, serm, logs in.
2. The user accesses the password reset functionality.
3. The user omits the new password and tries to reset it.
4. Verify that the system fails the user password reset.

## Fail password reset by Root user due the omission

Requirement document reference: Section 5.7

Pre-condition: A user with Root role already exists.

Test Steps:

1. The user logs in.
2. The user accesses the password reset functionality.
3. The user omits the new password and tries to reset it.
4. Verify that the system fails the user password reset.

## Fail password reset by OAGi admin developer due the non-compliant password

Requirement document reference: Section 5.7 (**Optional**)

Pre-condition: A user with the username, aNewUser1, already exists with the OAGi admin developer role.

Test Steps:

1. The user, aNewUser1, logs in.
2. The user accesses the password reset functionality.
3. The user enters a new non-compliant password and tries to reset it.
4. Verify that the system fails the user password reset.

## Fail password reset by OAGi developer due the non-compliant password

Requirement document reference: Section 5.7 (**Optional**)

Pre-condition: A user with the username, aNewUser1, already exists with the OAGi developer role.

Test Steps:

1. The user, aNewUser1, logs in.
2. The user accesses the password reset functionality.
3. The user enters a new non-compliant password and tries to reset it.
4. Verify that the system fails the user password reset.

## Fail password reset by Enterprise tenant admin user due the non-compliant password

Requirement document reference: Section 5.7 (**Optional**)

Pre-condition: A user with the username, aNewUser1, already exists with the Enterprise tenant admin role.

Test Steps:

1. The user, aNewUser1, logs in.
2. The user accesses the password reset functionality.
3. The user enters a new non-compliant password and tries to reset it.
4. Verify that the system fails the user password reset.

## Fail password reset by Enterprise tenant end user due the non-compliant password

Requirement document reference: Section 5.7 (**Optional**)

Pre-condition: A user with the username, aNewUser1, already exists with the Enterprise tenant end user role.

Test Steps:

1. The user, aNewUser1, logs in.
2. The user accesses the password reset functionality.
3. The user enters a new non-compliant password and tries to reset it.
4. Verify that the system fails the user password reset.

## Fail password reset by Free user due the non-compliant password

Requirement document reference: Section 5.7 (**Optional**)

Pre-condition: A user with the username, serm, already exists with the Free user role.

Test Steps:

1. The user, serm, logs in.
2. The user accesses the password reset functionality.
3. The user enters a new non-compliant password and tries to reset it.
4. Verify that the system fails the user password reset.

## Fail password reset by Root user due the non-compliant password

Requirement document reference: Section 5.7 (**Optional**)

Pre-condition: A user with Root role already exists.

Test Steps:

1. The user logs in.
2. The user accesses the password reset functionality.
3. The user enters a new non-compliant password and tries to reset it.
4. Verify that the system fails the user password reset.

1. OAGi admin is short for OAGi admin developer. [↑](#footnote-ref-1)
2. SRT Core functions refer to all functions other than user access, user information, billing management functions. [↑](#footnote-ref-2)
3. Enterprise users mean enterprise end users and enterprise admin users. [↑](#footnote-ref-3)
4. OAGi tenant roles mean OAGi developers and OAGi admin developers. [↑](#footnote-ref-4)
5. In the context of SRT core functions, “any change” always includes editing, deleting, changing state, and changing ownership. [↑](#footnote-ref-5)
6. Enterprise users mean enterprise end users and enterprise admin users. [↑](#footnote-ref-6)
7. Enterprise users mean enterprise end users and enterprise admin users. [↑](#footnote-ref-7)
8. Enterprise users mean enterprise end users and enterprise admin users. [↑](#footnote-ref-8)
9. Enterprise users mean enterprise end users and enterprise admin users. [↑](#footnote-ref-9)
10. In the context of SRT core functions, “any change” always includes editing, deleting, changing state, and changing ownership. [↑](#footnote-ref-10)
11. Enterprise users mean enterprise end users and enterprise admin users. [↑](#footnote-ref-11)
12. Enterprise tenant roles mean Enterprise end users and Enterprise admin users. [↑](#footnote-ref-12)
13. Enterprise users mean enterprise end users and enterprise admin users. [↑](#footnote-ref-13)
14. Enterprise tenant roles mean Enterprise end users and Enterprise admin users. [↑](#footnote-ref-14)