Self-registration workflow configuration - CustomerID

Ubisecure CustomerID supports different kinds of user registrations. Multiple types of registrations can also be used at the same time. The Ubisecure CustomerID registration workflows are configured in the `eidm2.properties` configuration file. By default, the installation contains no configuration keys for registration. If you want to use self-service registrations, you must create the configuration keys yourself.

A registration flow can have the following steps and phases:

1. Registration protection (optional)
2. User verification step (optional)
3. Backend query input step (optional)
4. User details input step, optional password setup and terms of service acceptance
5. Authentication method activation step (optional)
6. Summary step
7. Confirmation page
8. Approval by the main user through the administration screen (optional)
9. User account creation

First of all a registration may be protected by a configured set of authentication methods. This functionality is mainly used to limit the use of the certain registration to existing users. So a registration can also be used to request more access rights instead of always creating new users.

Then at the beginning of the actual registration wizard they may be a user verification step using an external authentication method (like a TUPAS method for example). It is mainly used to verify the identity of the user performing the registration. The configured, available external authentication methods are presented to the user on the front page. The user must select one of these methods to proceed with the authentication and registration.

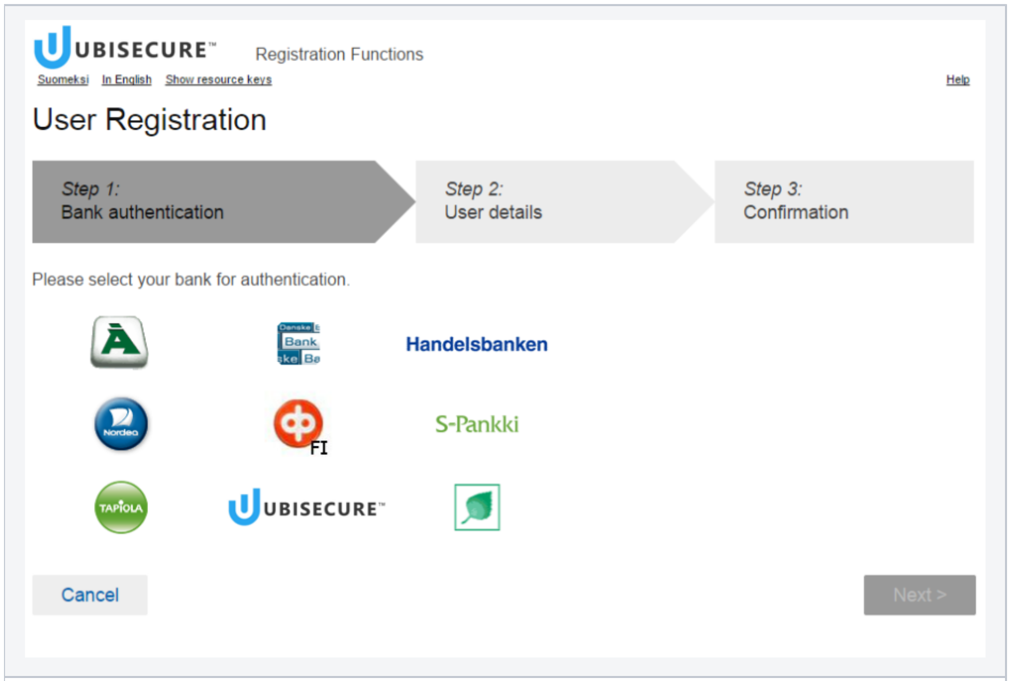

Figure 1. Example of TUPAS authentication methods in the user verification step

The registration URL address is in the form:

<url_base>`/eidm2/wf/register/`*<registration_name>*

For example:

`https://localhost:7443/eidm2/wf/register/person`

If, for example, we were configuring a simple person registration, the eidm2.properties configuration file could contain the following registration workflow configuration:

- This configuration would require that you create a registration configuration with the same value that is used as registration name – in this example: `registration.1 = person`
  (If you would use multiple registrations, they would need to have different index numbers, for example:
  `registration.2 = organization`)

- Configuration keys are allotted to a certain registration by placing the number of the registration in front of them. In this example, configuration would use keys that begin with `registration.1`. Here is an example of a functional set of person registration properties:

NOTE: You must define certain mandatory configuration keys in order for the registration to work. The mandatory keys are those that don't have a default value.

```
registration.1 = person

registration.1.logo.key = person-registration

registration.1.enabled = true

registration.1.protectiononly = false

registration.1.inviteonly = false

registration.1.newuseronly = false

registration.1.email.confirmation = false

registration.1.approval = false

registration.1.approval.organization =

registration.1.password = Password1

registration.1.password.method = constant

registration.1.mobile.confirmation = false

registration.1.methods = [ { "name" : "password.2", "mandatory" : "true", "visible" : "false", "default" :
"true" } ]

registration.1.userinfo.fields = firstname, surname, customid, email, mobile, acceptTerms

registration.1.userinfo.optional = customid

registration.1.summary.fields = firstname, surname, customid, email, mobile, acceptTerms

registration.1.organizations = [ { "path" : "Branches/Customers", "organizationtype" : "customer" }, [ {
"path" : "CustomerNumbers/Person", “organizationtype" : "person", "virtual" : "true" } ] ]

registration.1.roles = [ "eIDM/PersonalUser", "CustomerNumbers/Person/OrganizationOwner",  "CustomerNumbers
/Person/InsuranceCustomer",  "CustomerNumbers/Person/AccountCustomer",  "CustomerNumbers/Person
/PensionCustomer",  "CustomerNumbers/Person/FundCustomer", "CustomerNumbers/Person
/LoanCustomer",  "CustomerNumbers/Person/Reviewer" ]

registration.1.roles.firstuser =
```

Furthermore, registration consists of optional and obligatory steps. The steps are:

- User verification step (optional). The external authentication methods shown in this step are configured in a protection configuration and it can be selected with registration. `<index_number>.verification.protection.configuration` key.
- Entering user information (obligatory). The collected information and the obligatory information are configurable. This can be set with registration. `<index_number>.userinfo.fields` and registration. `<index_number>.userinfo.optional` key.
- Multiple steps can be configured by adding fields of each step inside curly brackets. For example, `registration.1.userinfo.fields = {customid, email}, {firstname, surname, mobile, acceptTerms}`
- Custom attributes can be created by giving them names and adding descriptions to `messages_xx.properties`. For example, `registration.1.userinfo.fields = firstname, surname, testikentta`
- The system also supports the option to check that the user has a valid customership in the company customer information system. These are set with keys that start with `registration.<index_number>.backend`.
- Activation of authentication methods (optional). With this step you can, for example, force the user to print out a list of one time passwords. Activation step is shown if there are methods set with `registration.<index_number>.methods` key that are defined visible. The activated authentication methods are configurable and the desired authentication methods can be set as obligatory. Obligatory authentication methods must be activated before the user can proceed.

- Information summary (obligatory). This step only shows the user the entered information and asks the user to check the information before proceeding. This step requires no configuration.

If approval is not required (see below) user can be transferred directly to an application in the Ubisecure CustomerID domain. The target application can be defined using the following keys
`registration.<index_number>.targetApplication` and
`registration.<index_number>.targetPath` where targetApplication must be the SAML entityID of the application service provider and targetPath is an optional entry pointing to the entry path of the application

You can also configure for each registration workflow whether an existing user must approve the new user registration. The approval options are:

- Registrations are automatically approved. This is set by giving the `registration.<index_number>.approval` key value false.
- Registrations must be approved by the administrative user of the organization where the registered user will be created. These administrative users have approval permissions for the target organization. This is set by giving the `registration.1.approval` key value true and leaving the `registration.<index_number>.approval.organization` value empty.
- Registrations must be approved by users of a specific organization regardless of the organization where the registered user will be created. These administrative users have approval permissions for the target organization. This is set by giving the `registration.<index_number>.approval` key value true and giving the entity ID of the desired organization as the value for the `registration.<index_number>.approval.organization` key.

You can also define the branch in the user repository where the users ends up in conjunction with the registration. This is done with the first organization defined in `registration.<index_number>.organizations` key.
Let us assume, for example, that the configuration file defines that registering users will end up to the Customers branch in the repository and the approving organization is the `InternalUsers` branch. In this case, any user in the `InternalUsers` branch who has the `user.approve` permission can approve the registered user.

Forwarding SAML Attributes to a Registration Workflow

It is possible to forward SAML attributes to the input fields of the registration, given the user is not found in the system. The SAML attribute must have the name of `user.<userinfo_field_name>` or `organization.<userinfo _field_name>` in order to create the mappings to the corresponding fields defined in `registration.N.userinfo.fields`.

Calling Registration Wizard with Redirect to Specific URL After Completion

If user should be redirected to a specific URL (e.g. back to related portal application) upon completion or after pressing the *cancel* button in the registration wizard, the registration wizard can be given a query parameter `returnurl` containing the URL to which the user will thereafter be redirected to. The same parameter can also be given to a protection URL if that protection later redirects to a registration.

Example:

```
https://customerid.mydomain/eidm2/wf/register/person?returnurl=https://www.my.portal.mydomain
```

If you use SAML AP in connection with the return URL and you want to create the returnurl parameter content statically, then here are the instructions how to create the correct value:

- Define what is the path in the target application where you want to redirect the user. If you have previous used the old `registration.N.targetPath` property then this is the same value.
- URL encode the target application path value so that you get the <URL encoded target path>.
- Find out (for example using Ubisecure SSO Management) the entity ID of the target application. If you have previously used the old `registration.N.targetApplication` property then this is the same value.
- URL encode the target application entity ID value so that you get the <URL encoded target application>.
- Use the following template and insert the previously mentioned application specific values to the correct places:
  - /uas/saml2/SessionRelayService?entityID=<URL encoded target application>&RelayState=<URL encoded target path>
- Finally URL encode the whole string that you got by using and filling out the template. This is then the correct value to be used as the value of the `returnurl` parameter.

Registration Properties

NOTE: To save space, the place-holder for *<index_number>* is replaced with *N* in the property descriptions in this chapter.

registration.*N*

This property defines the registration name and this name is used as part of the service URL. This property is mandatory in all registrations. The maximum length for this value is 255 characters.

Example:

```
registration.1 = person
```

registration.*N*.logo.key

This property defines the logo for the specific registration service. This property is optional.

Example:

```
registration.1.logo.key = person-registration
```

registration.*N*.enabled

This property defines if the specific type of registration is possible. There are two possible values:

- `true:` This registration is enabled.
- `false:` This registration is disabled.

Default is `false.` This property is optional but the registration is not enabled without using this property.

Example:

```
registration.1.enabled = true
```

registration.*N*.protection.configuration

This property defines which protection configuration (if any) is used in the registration. Valid values are protection configuration numbers.

Example:

```
registration.1.protection.configuration = 1
```

registration.*N*.protectiononly

This property defines if this registration is only available for those users that have logged in via the protection mechanism. There are two possible values:

- `true:` This registration workflow can only be used when the user has been through the protection mechanism.
- `false:` This registration can be used by going to the URL of the registration.

Default is `false`.

Example:

```
registration.1.protectiononly = true
```

registration.*N*.verification.protection.configuration

This configuration option adds an authentication step to the beginning of the registration workflow with methods defined in protection.properties.

Please note that the used protection configuration must have the following configuration option set: "customeriduseronly = false".
Default is `<not set>`. This property is optional.
Example:

```
registration.1.verification.protection.configuration = 1
```

registration.*N*.inviteonly

This property defines if only invited users are able to register to the system. This property is mandatory in all registrations. There are two possible values:

- `true:` This registration workflow can only be used when the user has been invited.
- `false:` This registration can be used by going to the URL of the registration.

Default is `false`.

Example:

```
registration.1.inviteonly = true
```

registration.*N*.newuseronly

This property defines if only new users are able to register to the system using this registration workflow. There are two possible values:

- `true:` This registration workflow can only be used by new users.
- `false:` This registration can be also used by existing users.

Default is true.

Example:

```
registration.1.newuseronly = false
```

registration.*N*.email.confirmation

This property defines if the user email address must be confirmed by sending an email to the given address. There are two possible values:

- `true:` Email verification is required.
- `false:` No email confirmation email is sent to the user.

The default is not to use email confirmation (false). This property is optional.

Note that you cannot use email confirmation if you have disabled the email field.

Example:

```
registration.1.email.confirmation = false
```

registration.*N*.approval

This property defines if the registration requires the approval from the organization admin user. There are two possible values:

- `true:` An approval is required.
- `false:` No approval is required.

The default value is true. This property is optional.

Example:

```
registration.1.approval = true
```

registration.*N*.approval.organization

This property defines the organization where the approvals are sent if approval is set to true. This property is optional.

Example:

```
registration.1.approval.organization = Org1/SubOrg2
```

registration.*N*.password

This property defines the default password that is assigned to the user during the registration if password setting is used and the method to set it is to use a constant password. This property is optional.

Example:

```
registration.1.password = Password1
```

registration.*N*.password.method

This property defines the method for setting the user password value in the registration. There are two possible values:

- `random:` The user is given a random password.
- `constant:` The user is given a constant password.

This property is optional.

Example:

```
registration.1.password.method = constant
```

registration.*N*.mobile.confirmation

This property defines if the mobile number the user gives at registration must be confirmed by responding to an SMS message sent to the number. There are two possible values:

- `true:` A confirmation is needed for the given mobile number.
- `false:` No confirmation is needed for the given mobile number.

The default value is `false.` This property is optional.

Note that you cannot use mobile confirmation if you have disabled the mobile field.

Example:

```
registration.1.mobile.confirmation = false
```

registration.*N*.methods

This property defines the list of available authentication methods the user can take into use during the registration. It may also contain authentication methods that are activated for the user without asking the user. Note that you cannot separately disable the password method when Active Directory is used as the main user storing repository so it will be activated anyway when AD is used. The format is a JSON array of authentication methods with their parameters. The possible parameters are:

- `name:` The technical name of the authentication method.
- `mandatory:` A boolean value that defines whether it is mandatory to activate the defined authentication method.
- `visible:` A boolean value that defines whether the defined authentication method will be presented to the user in the authentication method activation step.
- `default:` A boolean value that defines whether the state for the defined authentication method will be by default activated or disabled.

The default value is `an empty list`. This property is optional but often needed.

Example:

```
registration.1.methods = [ { "name" : "password.2", "mandatory" : "true", "visible" : "false", "default" :
"true" }, { "name" : "ubikey.sms.1", "mandatory" : "false", "visible" : "true", "default" : "true" }, { "name"
: "ubikey.otp.1", "mandatory" : "true", "visible" : "true", "default" : "false" } ]
```

registration.*N*.userinfo.fields

This property defines all the user input fields in the registration service. Example of one phase user info:

```
registration.1.userinfo.fields = firstname, surname, customerid, email, mobile, acceptTerms
```

Since version 3.1 user input fields can be divided to several steps. When this is needed the input field keys for each step must be grouped inside curly brackets {} as follows:

```
registration.1.userinfo.fields = {email, customerid}, {firstname, surname, email, mobile, acceptTerms}
```

Since version 3.1 custom attributes can be created by giving them names and adding descriptions to messages_xx.properties. For example,

```
registration.1.userinfo.fields = firstname, surname, testikentta
```

As of version 3.5 also organization attributes can be requested in registration. These can be defined with the prefix `organization.` and should be used in conjunction with the registration organization's storeattributes key. Respective message keys should be defined in `messages_xx.properties`. For example,

```
registration.1.userinfo.fields = firstname, surname, companyname, organization.identifier, password
registration.1.organizations = { "path" : \
"Customers/${companyname}", "storeattributes" : "true" }
```

The above would store the `organization.identifier` value given in registration to the Ubisecure CustomerID organization *Customers /${companyname}*, where *${companyname}* would get its value from `companyname` given in registration.

registration.*N*.userinfo.optional

This property defines the user input fields that are optional in the registration service.

Example:

```
registration.1.userinfo.optional = customerid
```

registration.*N*.userinfo.disabled

This property defines the user input fields that are disabled in the registration service (visible but not editable).

Example:

```
registration.1.userinfo.disabled = ssn
registration.N.summary.enabled
```

This property defines if the summary step is present in the registration wizard. There are two possible values:

- `true:` Summary step is present.
- `false:` Summary step is not shown.

The default value is `true`.

Example:

```
registration.1.summary.enabled = true
```

registration.*N*.summary.fields

This property defines all the fields shown in the summary step of the registration wizard.

Example:

```
registration.1.summary.fields = firstname, surname, customerid, email
```

registration.N.summary.backend

This property defines the backend call that is made after the summary step.

Example:

```
registration.1.summary.backend = summaryBackend
```

registration.N.summary.requirePrinting

If set to true the summary step must be printed before it can be sent. This may cause usability problems. It is not recommended to use this feature. There are two possible values:

- `true:` Printing is required.
- `false:` Printing is not required.

The default value is `false`.

Example:

```
registration.1.summary.requirePrinting = false
```

registration.*N*.temporary.fields

This property defines those fields that are not stored to the database after the registration is finished (user is activated). Only user fields can be used here.

Example:

```
registration.1.temporary.fields = ssn
```

Using User Input in Organization and Role Definition

The user details input during the registration in question can be used as a part of several configuration parameters below. The syntax used is based on Java Expression Language and thus allows combination of parameter values to resolve a value.

Examples of usage:
If user's name is *Test User* with a customer identifier *56789* given during the registration, the following configuration

```
registration.N.organizations = { "path" : "Customers/${customerid}", "name" : "${firstname + ' ' + surname +
'\\'s organization'" }
```

would cause the path to finally resolve to *Customers/56789* and the organization residing there be called *Test User's organization*. Each string where the syntax is supported can contain several references to registration parameters.
Organization attributes in the registration can be accessed with prefix `organization.`

```
registration.N.organizations = { "path" : "Customers/${organization.customerid}", "name" : "${organization.
name}" }
```

If organization's technical name, i.e. the last part of the path should be generated, a special attribute `generatedTechnicalName` is available. This feature is as such very limited: each reference to the `generatedTechnicalName` in all organizations return the same generated value. Usage example:

```
registration.N.organizations = { "path" : "Customers/${organization.generatedTechnicalName}", "name" :
"Organization with generated technical name" }
```

registration.*N*.organizations

This property defines the organization where the user will be stored and any other organizations that should be created as part of the registration process. The first organization defined is where the user will be stored. The organization(s) must be defined using JSON notation in either of two ways; if only an organization where the user is to be stored needs to be define this can be achieved by defining a single organization as a JSON object.

The JSON object notation of an organization contains one or more of the following keys.
Keys definable for user organization:

- `path`: The entity name of the organization e.g. `"path" : "Customers/${customerid}"`. This key supports user input notation described above. If no *customerid* or similar attribute uniquely defining the organization in the registration can be used a special attribute `organization.generatedTechnicalName` is available as described above.
- `name`: The friendly name of the organization created e.g. `"name" : "Customer ${customerid}"`. If not defined the last part of the entity name (relative name) is used. This key supports user input notation described above.
- `organizationtype`: The organization type of the user organization e.g. `"organizationtype" : "customer"`. This key is mandatory.
- `storeattributes`: If set to *true* the organization attributes provided during the registration will be stored in the respective organization. If only certain attributes should be stored then the names of the respective attributes can be given in a JSON array object, e.g. "storeattributes" : ["orgattribute1", "orgattribute2"].

Additional organizations can be defined with the same key set found above as the user organizations. In addition following keys are definable for additional organizations:

- `virtual`: Set to *true* if the organization should be virtual, i.e. `"virtual" : "true"`.

The JSON notation of the user organization must at minimum contain the key path which defines the entity path of the organization where the user is stored.

Example:

```
registration.1.organizations = { "path" : "all_users", "organizationtype" : "", "storeattributes" : "false" }
```

If organizations are not wanted to be created during registration, this property can be left empty. In this case the users' organization has to be predefined in some other way, for example in role invitation. Example:

```
registration.1.organizations = { }
```

If one or more additional organizations should be created upon user registration the value of the key should be a JSON array where first element contains a JSON object definition of the user organization and the second object a JSON array of additional organizations. If the path cannot be resolved, when creating additional organizations (i.e. there are unresolved variables in the path) the creation of additional organizations are ignored.

Example:

```
registration.2.organizations = [ { "path" : "all_users", "organizationtype" : "", "storeattributes" : "false"
}, [ { "path" : "Customers/${customerid}/", "organizationtype" : "customer", "virtual" : "true",
"storeattributes" : [ "customerid" ] }, { "path" : "Other/Organization" } ]]
```

registration.*N*.roles

This property defines roles that are assigned to the user during the registration. These roles should be defined using the whole entity name of the role. This property supports user input notation described above. If the entity of the role cannot be resolved due to unresolved attributes, the assignation of the role is ignored.

If you want to grant the user roles from the organization where the user is being registered to, then you can refer to this organization with the variable ${user_organization} and append the desired role after this, separated by a slash character. Note that these roles must already be configured to the target organization.

Example:

```
registration.1.roles = [ "all_users/User", "Yritysasiakkaat/${companyid}/Corporate_user" ]
registration.1.roles = [ "${user_organization}/OrganizationUser" ]
```

It is also possible to configure a separate approval workflow for roles, this requires a slightly different method of configuration, where the role definition is and approval requirements are defined as a JSON Object, such as in the example below.

Example:

```
registration.1.roles = [ { "path" : "all_users/User" },{ "path" : "all_users/RestrictedUser", "approval" :
"true" } ]
```

In the example above, the path key defines the path of the role and the approval key defines whether the role requires a separate approval process. If the value of approval is set to *true*, the role requires a separate approval process and will not automatically be approved when the user registration is approved. The user registration will, however, result in a functional user account, but the role will not be available until it has been separately approved. If the approval key is set with a *false* attribute, or omitted entirely, the role will be in effect when the user registration itself is approved.

registration.*N*.roles.firstuser

This property defines roles that are assigned to the first user that registers to the organization. Organization membership is defined if the user has a role from the organization. This property supports user input notation described above.

Example:

```
registration.1.roles.firstuser = [ "all_users/main_user" ]
```

It is also possible to configure a separate approval workflow for roles, this requires a slightly different method of configuration, where the role definition is and approval requirement are defined as a JSON Object, such as in the example below.

Example:

```
registration.1.roles.firstuser = [ { "path" : "all_users/User" },{ "path" : "all_users/RestrictedUser",
"approval" : "true" } ]
```

registration.*N*.userinfo.backend

This property defines when and which backend request should be made for the registration. Each backend can be attached to an input step (i.e. the backend will be called after the corresponding input step) by defining a list of comma separated values formatted *<stepnumber>:<backendname>*. A single backend configuration in each input step is allowed.

```
registration.1.userinfo.backend = 1:customerdata
```

An example of using several backends in one registration:

```
registration.2.userinfo.backend = 1:customerdata, 2:accountdata
```

Customerdata backend is called after the first user input phase, the *accountdata* backend after the second.

registration.*N*.targetApplication

NOTE: registration.*N*.targetApplication and registration.*N*.targetPath only work if registration.*N*.approval is set to *false* so that user accounts are usable directly after registration.

This property defines the application to which user should be optionally transferred after the registration. The value of the parameter should be the SAML SP entity identifier of the target application. This value can be found from Ubisecure SSO Management. This configuration can only be used in connection with appplications integrated with the SAML protocol.

Example (if using entityId generated by SAML SP):

```
registration.1.targetApplication = urn:uuid:359ec8f8-b9ce-38f4-8680-5861a38d2473
```

registration.*N*.targetPath

This property defines the whole path on the target host to which the receiving SAML SP will direct the user.

Example:

```
registration.1.targetPath = /myApplication/entryPath
```

registration.*N*.wizard.back.enabled

This property defines if the back button can be used in the registration wizard. There are two possible values:

- `true:` Back button can be used.
- `false:` Back button cannot be used and is not even shown to the user.

The default value is `true.` This property is optional.

Example:

```
registration.1.wizard.back.enabled = true
```

registration.*N*.use.samlap.with.returnurl

This property defines if SAML AP authentication method is used when returning to the URL defined in `returnurl` parameter. There are two possible values:

- `true:` SAML AP method is used.
- `false:` SAML AP method is not used.

The default value is `false.` This property is optional.

Example:

```
registration.1.use.samlap.with.returnurl = true
```

registration.*N*.logout.when.cancel

This property defines if logout is issued when pressing "**Cancel**" in registration.

Return page is defined in order:

1. Gained cancelurl URL parameter
2. Configured cancel.returnurl registration configuration
3. Gained returnurl URL parameter
4. Configured general.default.logoutReturnUrl configuration
5. Configured general.default.returnUrl configuration

The default value is false. This property is optional.

Example:

```
registration.1.logout.when.cancel = true
```

registration.*N*.cancel.returnurl

This property defines returnURL page initiated by "**Cancel**" button in Registration page.

Return page is defined in order:

1. Gained cancelurl URL parameter
2. Configured cancel.returnurl registration configuration
3. Gained returnurl URL parameter
4. Configured general.default.logoutReturnUrl configuration
    a. If `registration.N.logout.when.cancel` is `true`
5. Configured general.default.returnUrl configuration

Example:

```
registration.1.cancel.returnurl = https://www.ubisecure.com
```