Backend query configuration - CustomerID

backend.properties

Configuration of all of the backend queries can be found in the `backend.properties` file. There can be several backends and a unique name should be given to all of them. In this example the name of the backend is `customerdata`. Three configuration nodes are needed for backend queries to work: `input`, `output` and `config`. They all use a similar JSON-style notation where an attribute set of name/value pairs is included inside curly brackets. There is also a fourth configuration node for transformations, which is optional.

input

`input` describes the parameters that the backend gets in a request during the registration. For example:

```
customerdata.input = { "user.email": "Email", "user.accountnumber": "AccountNumber" }
```

In this example the lower case parameters `user.email` and `user.accountnumber` are the same that are given in registration's user input fields. The parameters `Email` and `AccountNumber` are the translated parameter names that are added to the URL in the sent REST request. The parameter `user.password` will contain the unencrypted password as entered by the user.

output

`output` describes the parameters that the backend sends back in the response. For example:

```
customerdata.output = { "/customer/lastname": "user.surname", "contract": "user.contractnumber" }
```

In this example the parameter `/customer/lastname` is an XPath expression for parsing data from the response. This data is put to a field with the name `surname`. If the left side value is not an XPath expression the value is interpreted as an element name and the contents of each element with the name in the response document are taken to Ubisecure CustomerID.

The information returned from the backend may be shown to the user by adding similarly named fields to the registration's user input fields that are used after the backend request. If the fields are not visible, the information is still saved after the registration.

config

`config` describes the configuration parameters that are needed to make the request. For example:

NOTE: Configuration keys must be written in lower case letters.

```
customerdata.config = { \
        "url": "http://localhost:7080/backend", \
        "timeout": "3000", \
        "defaultparams": { "authcode": "123" }, \
        "langcode": "locale", \
        "statuscode": "/customer/status", \
        "statusmapping": { "OK": "OK", "NOT_OK": "ERROR", "CRITICAL": "STOP" }, \
        "username": "basicAuthUsername", \
        "password": "basicAuthPassword", \
        "method": "GET", \
        "message": { "status-code" : "*", "body-or-header" : "header", "location" : "code", "use-as-key" :
"true" } \
        }
```

- `url` parameter describes the URL where backend is situated. This URL can be a HTTPS address if the issuer of the server certificate is trusted by the Java execution environment.
- `timeout` parameter describes if and how long the timeout should be for the request before showing an error message.
- `defaultparams` parameter describes the list of static parameters that are sent to the backend in every request. The key uses JSON object notation.
- `langcode` parameter describes the parameter which is the name of the language code parameter in the request.
- `statuscode` parameter describes where the status code is in response, there are two modes of configuration: *xpath* and *http-header* of which *xpath* is the default behavior. When using the *xpath* mode, all HTTP codes other than 200 will result in an error condition. When using *http-header* mode, it is possible to use the HTTP status code or message as a key for the `statusmapping` parameter. For example, consider the fragment `"statuscode": "/path/to/status"`, this will be interpreted similarly as `"statuscode" : { "xpath" : "/path/to/status" }`. If you wish to use the HTTP status code in conjunction with `statusmapping`, then you should use the configuration `"statuscode" : { "http-`

`header" : "status:code" }`, or if you wish to map the HTTP status message you should define `"statuscode" : { "http-header" : "status:message" }`.

- `statusmapping` parameter maps the status code value in response to a result action in Ubisecure CustomerID. Left side of each mapping corresponds to the value returned by backend while right side corresponds to values used to guide Ubisecure CustomerID actions. *OK* means that registration is continued, *ERROR* means that an error message is shown in the current wizard step and *STOP* means that registration is stopped and user is redirected to an error page with an error message. The key uses JSON object notation.
- `username` and `password` parameters describe the credentials used in the HTTP Basic Authentication. These parameters are optional. If they are not included, HTTP Basic Authentication is not used.
- `method` parameter describes the HTTP method used in the request. This parameter is optional and if not specified, GET will be used as default.
- `message` parameter describes what messages are shown to the user depending on the backend response. Each map specifies how the message will be produced based on the status code of the response. All of the four parameters must be specified for each mapping.
    - `status-code`: Specifying the HTTP status code on which the mapping is based on. Possible values:
        - HTTP status code, for instance *404* or *500*.
        - "*": Corresponds to all the response status codes that can't be mapped to a specific status code in the configuration.
    - `body-or-header`: Whether the message will be based on content found in the HTTP header or the actual HTTP XML message content. Possible values:
        - *body*
        - *header*
    - `location`: The location where the message will be searched for. Possible values:
        - XPath addressing the message location in the XML response. Can be used only if the `body-or-header` is set to *body*.
        - *code*: The message will be read from the status code of the HTTP header. Can be used only the `body-or-header` is set to *header*.
        - *message*: The message will be read from the message of the HTTP header. Can be used only the `body-or-header` is set to *header*.
    - `use-as-key`: Defines if the message will be used as is or as a language key value.

      If message is defined to be used as key, then the retrieved message will be interpreted as part of a localization key that will be mapped to a language specific value specified in `messages_xx.properties`. Addressing the language specific value in the localization file follows the following notation:

      ```
      backend.<backend-name>.message.<key value from backend response>
      ```

An example of a message configuration:

```
"message" : \
[ { "status-code" : "400", "body-or-header" : "body", "location" : "/error/message" } , \
{ "status-code" : "500", "body-or-header" : "body", "location" : "/error/message" } , \
{ "status-code" : "201", "body-or-header" : "header", "location" : "code" } , \
{ "status-code" : "200", "body-or-header" : "header", "location" : "code" } ]
```


NOTE: For backward compatibility, the message parameter can also be a single parameter describing an XPath. In this less versatile configuration, displayed message is always mapped to the value found in the XPath.

transform

`transform` points to the XSLT file. For example:

```
customerdata.transform = transform.xslt
```

See Backend queries - CustomerID for instructions concerning how to transform backend responses to the Ubisecure CustomerID XML schema in order to be able to make more detailed data modifications.

Backend related language keys in messages_xx.properties

If registration is stopped, user is redirected to feedback page.

```
backend.feedbackPage.title = Registration was interrupted
backend.feedbackPage.feedback = Continuing of registration is not possible. Please, contact customer support.
```

This error is shown if backend response don't include one.

```
backend.feedbackPage.error = Backend connection received an error.
```

These errors are shown if timeout occurs or response status code is other than HTTP 200 when requesting backend.

```
backend.internalerror = Backend connection failed.
backend.timeout = Backend connection failed, try again later.
```

An example of the localization of the messages returned and addressed by the message parameter.

```
backend.mybackend.message.500 = Internal failure. Try again later.
backend.mybackend.message.usernotfound = User not found.
backend.mybackend.message.invalidid = The provided id was erronous.
```

For instance, if the query of the backend named *mybackend* returns a HTTP 500 status, the message mapping would map to the `backend.mybackend.message.500` key according to the following configuration:

```
{ "status-code" : "500", "body-or-header" : "header", "location" : "code" }
```

Another example: The following backend response with status code 400

```
<error><message>invalidid</message></error>
```

would map to the `backend.mybackend.invalidid` key according to the following configuration:

```
{ "status-code" : "500", "body-or-header" : "body", "location" \
: "/error/message" }
```

Backend related parameters in eidm2.properties

The `registration.N.userinfo.backend` property defines when and which backend request is made. First parameter describes after which user information field step the backend is called. The second parameter describes the name of the backend configuration (in `backend.properties`). The `registration.N.summary.backend` property defines the backend that is used in the summary step.

```
registration.N.userinfo.backend = 1:customerdata
registration.N.summary.backend = summaryBackend
```