EVOTIX

API Guide

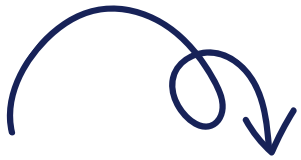
Assure Customer API

User Guide: Managing Users via API

Evotix Ltd. Revision 5.0 (April 2025)

US +1 (872) 215 5913 **UK** +44 (0) 161 521 8490 **AU** +61 3 8595 5909 **Visit** evotix.com

EVOTIX



The Assure Customer API is the means by which Evotix provides integration ‘capability to Assure for customers.

The Customer API is available to customers via the public internet and takes the form of a RESTful API. Using the Customer API you can automate processes such as managing users, org unit structure and exporting data for analysis. Making use of the Customer API requires a level of technical expertise so this is typically something that a company’s IT function would handle.

This guide focusses on how to set up the Customer API to manage users in Assure. A separate guide is available for setting up the Customer API.

EVOTIX

Contents

EVOTIX

Managing users via the Customer API

One of the interactive methods available via the Customer API is the ability to manage user accounts in Assure.

This allows you to create, update and disable user accounts in Assure.

Limitations for creating and updating users

The following lists the current limitations of the API for creating and updating users. It is expected that these will be addressed in future Assure releases:

- Can't configure dashboards for a user (although dashboards can be manually added via the Assure UI once the user has been created).
- Manual changes are allowed to API defined user data. In the Assure UI an administrator can amend user data (e.g. email address, full name, roles) which has been defined by the API. If the user is subsequently updated via the API then manual changes made in Assure will be overwritten by the API supplied data.
- No support for configuration of notification groups for the user.

There is no support for providing an initial password for the user. This is because the use of static passwords is not secure and Assure supports emailing out a password link. This method is used to allow users to set their initial password (or to reset their password if requested when updating a user via the API).

Pre-configuration

Before the APIs can be used to create and update users there is some pre-configuration required in Assure for Organisational Units (for units not created via the Customer API), Roles, and Supervisor Privileges. This is required for the API methods to be able to correctly assign users to the Organisational hierarchy and to the correct roles (and to supervisor privileges for those users who need them).

EVOTIX

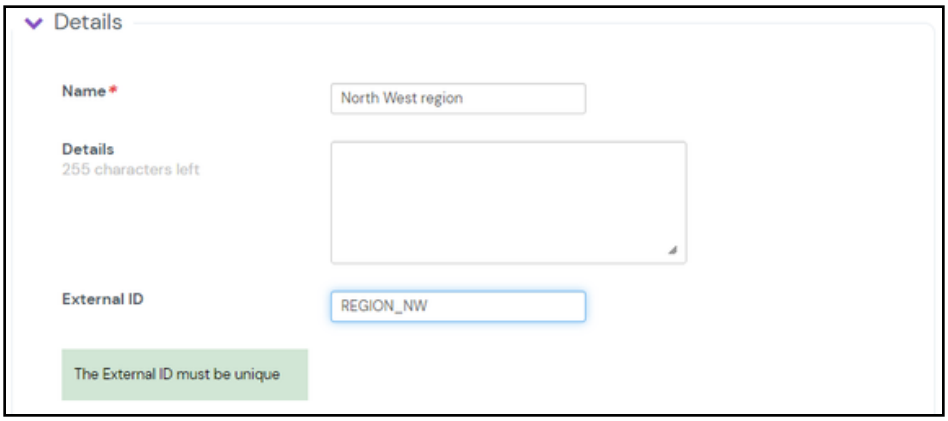
Organisational Unit external IDs

The Assure Organisational Unit hierarchy has a new attribute for each unit called 'External ID'. You will find this in the 'Edit' page of an Organisational Unit and its purpose is to allow an unique external identifier to be associated with each unit. This is necessary because:

1. The existing Organisational Unit names are not unique and therefore cannot be used with by the API method to target a specific Organisational Unit.
2. Organisational Unit names can be changed by Assure administrative users so they are not guaranteed to align with external IT systems.
3. Integration workflows should use the immutable unique identifier for Organisational Units so that changes to names (whether in Assure or the source system) do not break the integration workflow. This means Assure needs to be able to configure the unique identifier against each Organisational Unit which is what the External ID field does.

It is strongly recommended that the customer use their own identifier external ID of an Organisational Unit i.e. the identifier that their IT systems use for the Organisational Unit (e.g. for a retailer this might be the Shop ID). This removes the need for the customer to maintain a mapping between their Org Unit identifiers and the Assure internal identifier for an Org Unit.

The external IDs for existing Organisational Units can be configured manually using the Assure UI (screenshot below shows an example setting the Organisational Unit external ID for the "North West region" to be REGION_NW). This is fine for testing but for ensuring that the Organisational Unit hierarchy is in sync with the customers user management system we have a bulk import/update tool which can be used.



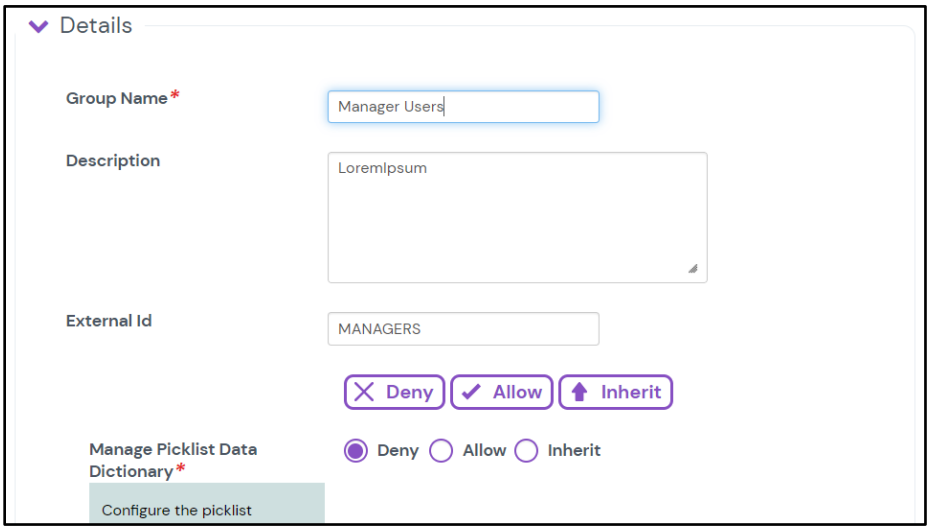
EVOTIX

Role external IDs

Assure roles also now have a new attribute for each role called 'External ID'. You will find this in the 'Edit' page of a Role and its purpose is to allow an unique external identifier to be associated with each role. This is necessary for the same reasons as detail above for the Organisational Units. The external IDs for the roles can ONLY be configured manually using the Assure UI (screenshot below shows an example setting the Role external ID for the "Sales users" to be SALES).

Supervisor Privilege external IDs

Assure Supervisor Privileges also now have a new attribute for each Supervisor Privilege called 'External ID'. You will find this in the 'Edit' page of the Supervisor Privilege and its purpose is to allow an unique external identifier to be associated with the Supervisor Privilege. This is necessary for the same reasons as detail above for the Organisational Units. The external IDs for the Supervisor Privileges can ONLY be configured manually using the Assure UI (screenshot below shows an example setting the Role external ID for the "Manager Users" to be MANAGERS).



User POST JSON object

When creating or updating a user the user's details need to be provided in the form of a JSON object. JSON is the most commonly used syntax for describing data objects in RESTful APIs. The OpenAPI schema for the Customer API contains the formal definition of the JSON structure i.e. the userPOSTRequest object. The OpenAPI schema is the master definition of the API methods and data objects, it should always be consulted to understand the required fields, field types, max data lengths and string patterns, plus descriptions of the behaviour associated with the use of each field (or its omission). Some software tools and platforms can consume the OpenAPI schema to automate the process of generating the correct JSON and if this is available it should be used. To aid understanding of how the JSON fields in the User JSON object relate to the resulting User setup in Assure the following sections contain some worked examples.

The text encoding to be used for all interactions with the Customer API is UTF-8. This is pretty much the standard today for software tools and platforms however it is important to check that you are using UTF-8 as if not then when you get foreign characters in the data or other symbols like emoticons these will not appear correctly in Assure.

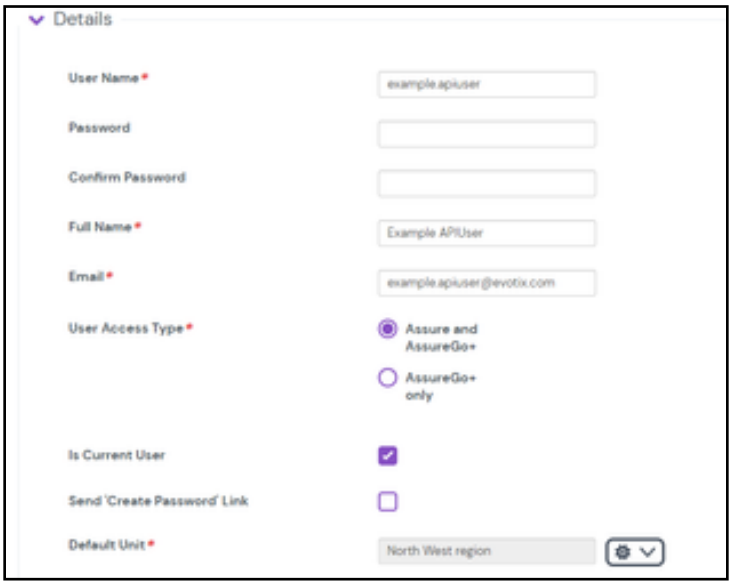
As JSON objects are described using plain text it is possible to hand craft these objects for initial testing. However, it is VERY IMPORTANT to ensure that when generating JSON objects to send to the Customer API you use a proper JSON library or a tool with native JSON support. This is because JSON relies on 'escaping' for certain data values, if this escaping is not done correctly it will lead to API errors and can also be a source of security vulnerabilities.

EVOTIX

Minimum user data

This example shows the minimum possible user data which can be used to create/update a user. A user created with this data would not be very useful as they have no roles defined however it is a good place to start for testing the ability to create or user a user.

```
{
  "username": "example.apiuser",
  "fullname": "Example APIUser",
  "email": "example.apiuser@evotix.com",
  "defaultOrgUnitExternalId": "REGION_NW"
}
```



User with a role

This example shows the same user with a role which would allow them to actually meaningfully use Assure. The user details will be the same as the 'minimum user data' screen shot above. The difference in this case is that the role is present in the Permissions section.

EVOTIX

```
{
  "username": "example.apiuser",
  "fullname": "Example APIUser",
  "email": "example.apiuser@evotix.com",
  "defaultOrgUnitExternalId": "REGION_NW",
  "roles": [
    {
      "orgUnitExternalId": "REGION_NW",
      "roleExternalId": "SALES"
    }
  ]
}
```

EVOTIX

User with multiple roles

This example shows the same user with multiple roles (including a role which includes all children). The user details will be the same as the 'minimum user data' screen shot above. The difference in this case is that the two roles are present in the Permissions section (with the Read only role including children).

```
{
  "username": "example.apiuser",
  "fullname": "Example APIUser",
  "email": "example.apiuser@evotix.com",
  "defaultOrgUnitExternalId": "REGION_NW",
  "roles": [
    {
"orgUnitExternalId": "REGION_NW",
      "roleExternalId": "SALES"
    },
{
      "orgUnitExternalId": "UK",
                          "roleExternalId": "VIEWER",
      "includeChildUnits": true
    }
                                      ]
}
```

EVOTIX

User with linked person

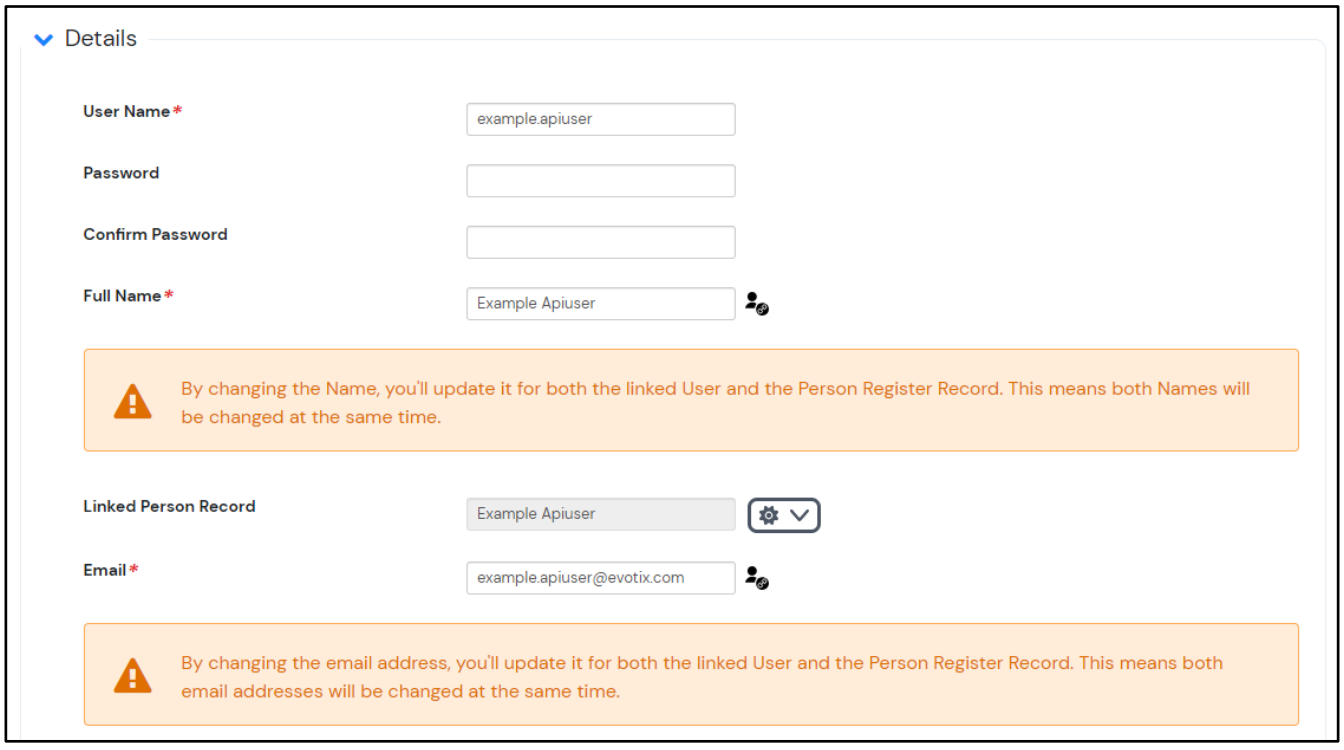
This example shows the same user with a linked person.

In order to link a User with a Person Record, you must have the feature enabled in System Settings.

In order to link a User to a Person record, the request will include the same fields as in the 'minimum person register data' section but it will also require the 'linkedPersonRecordReference' field.

Note: The email and fullname field values of the User will also update the linked Person's record's person email, forename and surname as well.

```
{
  "username": "example.apiuser",
  "fullname": "Example APIUser",
  "email": "example.apiuser@evotix.com",
  "defaultOrgUnitExternalId": "REGION_NW",
  "linkedPersonRecordReference": "ExamplePersonRecordReference"
}
```



Specific Behaviour:

- Create endpoint:
  - If People/User linking is not enabled and a linked Person Register reference is provided, then it will return error 400. Error 400 means no changes have been made at all.
  - Person Register reference needs to be provided to link to a User.
  - If a Person Register reference is provided it will validate if the given Person can be linked, if they cannot be linked it will return an error 400.
    - A Person cannot be linked if:
      - The feature is not enabled
      - The Person is linked to another User (this mirrors the behaviour in the UI)
      - The update will create duplicate Person emails
- Edit endpoint:
  - If People/User linking is not enabled and a linked Person Register reference is provided then it will return error 400.
  - Peron Register reference needs to be provided to link to a User.
  - If a Person Register reference is provided it will validate if the given Person can be linked, if they cannot be linked return error 400.
    - A Person cannot be linked if:
      - The feature is not enabled
      - The Person is linked to another User (same as through the UI)
      - The update will create duplicate Person emails
  - If the User is already linked to a Person and a different Person Register reference is provided then the User will be re-link user to a new Person. Shared fields will be updated to match the users.
    - If the Person cannot be linked then it will return an error 400. A Person cannot be linked if:
      - The feature is not enabled
      - The Person is linked to another User (same as through the UI)
      - The update will create duplicate Person emails
  - If the User is already linked to a Person and the Person Register reference is empty/null the User will be unlinked from the Person.
  - If the User is not current / made not current, the API will follow the same approach regarding linking/unlinking/re-linking depending on what information is included for the Person Register reference.
- Delete endpoint
  - Deleting a User will un-link it from a person.

EVOTIX

User with everything defined

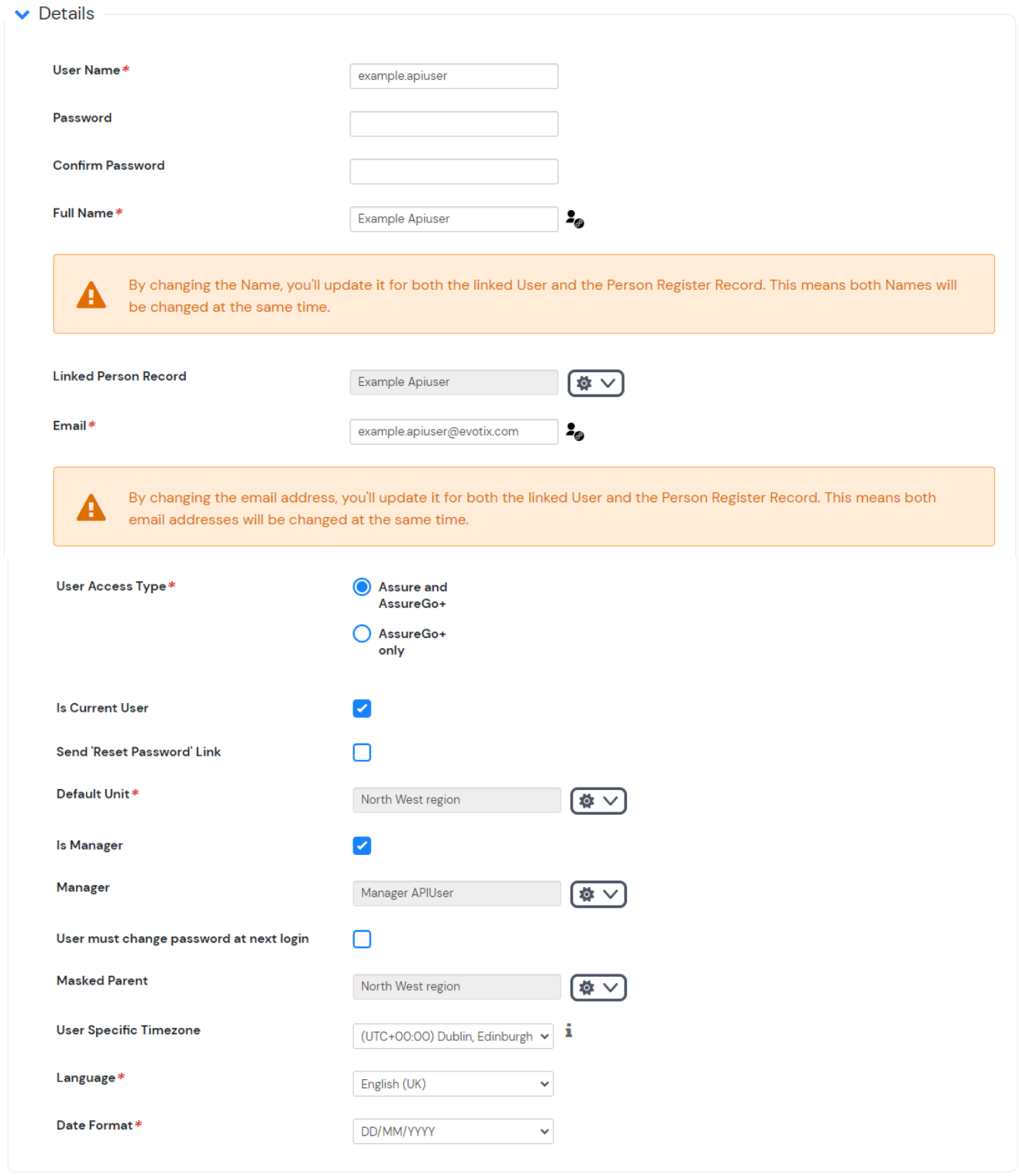
This example shows the same user with multiple roles (including a role which includes all children) and all the possible fields defined. Note that as per the OpenAPI schema the sendPasswordReset and requirePasswordChange fields only take effect when updating a user (you can include them when creating a user but they will be ignored).

It is not necessary to set all the fields when creating/updating a user. Any of the fields which are not required can be omitted in which case the default value for the field will be applied. See the description against each field in the OpenAPI schema for details of the default behaviour.


```
{
    "username": "example.apiuser",
    "fullname": "Example APIUser",
    "email": "example.apiuser@evotix.com",
    "defaultOrgUnitExternalId": "REGION_NW",
    "linkedPersonRecordReference": "ExamplePersonRecordReference",
    "maskedOrgUnitExternalId": "REGION_NW",
    "roles": [
      {
        "orgUnitExternalId": "REGION_NW",
        "roleExternalId": "SALES"
      },
      {
        "orgUnitExternalId": "UK",
        "roleExternalId": "VIEWER",
        "includeChildUnits": true
      }
  ],
  "assureGoPlusOnly": false,
  "isManager": true,
  "managerUsername": "manager.apiuser",
  "dateFormat": "MONTH_FIRST",
  "languageCode": "en-gb",
  "timezoneName": "GMT Standard Time",
  "supervisorPrivilegeExternalId": "MANAGER",
  "sendPasswordReset": true,
  "requirePasswordChange": true,
  "sisenseRole": "VIEWER"
}
```


The user permissions will be the same as the ‘user with multiple roles’ screen shot above, the user will have permission to view insights.

EVOTIX



EVOTIX

User JSON PATCH Object

When sending a patch request to update a user's details they need to be provided in the form of a JSON object. JSON is the most commonly used syntax for describing data objects in RESTful APIs, for an introduction to JSON see this guide. The OpenAPI schema for the Customer API contains the formal definition of the JSON structure i.e. the userPATCHRequestobject. The OpenAPI schema is the master definition of the API methods and data objects, it should always be consulted to understand the required fields, field types, max data lengths and string patterns, plus descriptions of the behaviour associated with the use of each field (or its omission). Some software tools and platforms can consume the OpenAPI schema to automate the process of generating the correct JSON and if this is available it should be used.

The below section of JSON shows all the fields in the User JSON PATCH Object.
Note. The object contains almost all the same fields as the JSON POST object but it does not contain the username field in the body.

EVOTIX

```
{
    "fullname": "Example APIUser",
    "email": "example.apiuser@evotix.com",
    "defaultOrgUnitExternalId": "REGION_NW",
    "linkedPersonRecordReference": "ExamplePersonRecordReference",
    "maskedOrgUnitExternalId": "REGION_NW",
    "roles": [
      {
        "orgUnitExternalId": "REGION_NW",
        "roleExternalId": "SALES"
      },
      {
        "orgUnitExternalId": "UK",
        "roleExternalId": "VIEWER",
        "includeChildUnits": true
      }
  ],
  "assureGoPlusOnly": false,
  "isManager": true,
  "isCurrent": true,
  "managerUsername": "manager.apiuser",
  "dateFormat": "MONTH_FIRST",
  "languageCode": "en-gb",
  "timezoneName": "GMT Standard Time",
  "supervisorPrivilegeExternalId": "MANAGER",
  "sendPasswordReset": true,
  "requirePasswordChange": true,
  "sisenseRole": "VIEWER"
}
```

The text encoding to be used for all interactions with the Customer API is UTF-8. This is pretty much the standard today for software tools and platforms however it is important to check that you are using UTF-8 as if not then when you get foreign characters in the data or other symbols like emoticons these will not appear correctly in Assure.

As JSON objects are described using plain text it is possible to hand craft these objects for initial testing. However, it is VERY IMPORTANT to ensure that when generating JSON objects to send to the Customer API you use a proper JSON library or a tool with native JSON support. This is because JSON relies on 'escaping' for certain data values, if this escaping is not done correctly it will lead to API errors and can also be a source of security vulnerabilities.

EVOTIX

Create a user

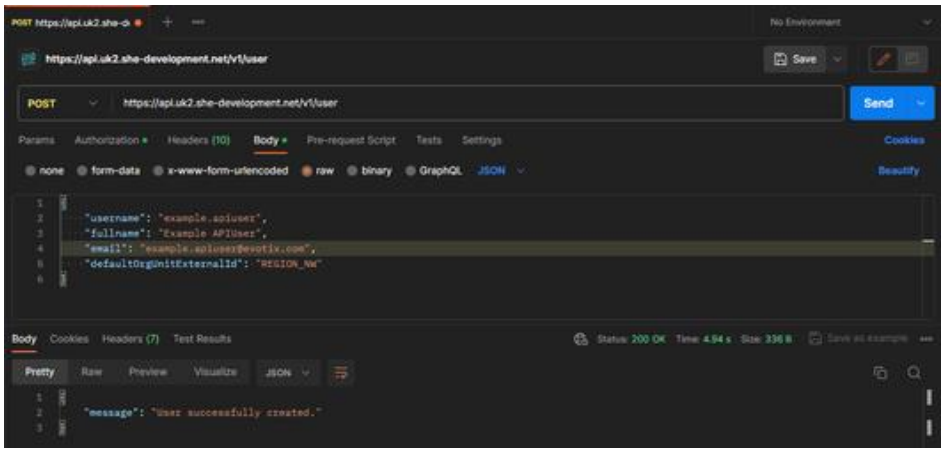
The Customer API allows the creation of users in Assure using the /v1/user API method with the POST verb. The users details are supplied in the body of the API request using the User JSON object (see section above). Users are uniquely identified by their login user name, which is the value in the username field of the User JSON object. If there is no user record for the login user name then the /v1/user API method will create the user using the details in the User JSON object. If the customer's licence limit has been reached then attempting to create a user which would require another licence will result in an error response (HTTP error 400).

If there is an existing user record for the login user name then the /v1/user API method will update the existing user record to match the supplied details (see the 'Update a user' section below for details on the behaviour when updating an existing user).

Before attempting to use the /v1/user API method make sure you understand the User JSON object (see the section above) and that you have the required details to access the API (i.e. the URL prefix, API key, etc). The following table sections show how to create a user using the /v1/user API method in a variety of software tools / platforms:

Any HTTP response code is the range 200-299 should be treated as success (do not use the contents of the response body for identifying success or failure as these may change).

EVOTIX

Postman API platform	The screenshot below shows a successful 'create user' request for a customer on the uk2 stack. The API key is supplied via the 'Authorization' tab. The response section at the bottom shows the result Status: 200 OK which indicates that the user creation was successful (also displayed is the response message from the Customer API confirming the successful creation). NB: With tools like this where the User JSON object is manually entered you must ensure the contents of any string values are properly escaped. 
Windows Powershell	The code block below shows the few lines of Powershell script required to setup the User Object JSON and to make the 'create user' request to the Customer API for a customer on the uk2 stack. The xxxxxxxxxxxxxxx is where the API key needs to be placed. `$UserObjectJSON = @{` `  "username" = "example.apiuser"` `  "fullname" = "Example APIUser"` `  "email" = "example.apiuser@evotix.com"` `  "defaultOrgUnitExternalId" = "REGION_NW"` `} | ConvertTo-Json` `Invoke-WebRequest `` `  -Headers @{'x-api-key' = 'XXXXXXXXXXXXXXX' } `` ` -Uri https://api.uk2.sheassure.net/v1/user `` `  -Method Post `` ` -ContentType 'application/json' `` `  -Body $UserObjectJSON`` The following shows the output from the above Powershell script code being run where a successful response is generated. The StatusCode: 200 indicates that the create user request was successful. Some response lines have been removed for brevity. `StatusCode : 200` `StatusDescription : OK` `Content : {"message":"User successfully created."}`

Python	The code block below shows the few lines of Python code required to setup the User Object JSON and to make the 'create user' request to the Customer API for a customer on the uk2 stack. The xxxxxxxxxxxxxxx is where the API key needs to be placed.
	
	`# pip install requests`
	`import requests`
	
	`user_object = {`
	`"username": "example.apiuser",`
	`"fullname": "Example APIUser",`
	`"email": "example.apiuser@evotix.com",`
	`"defaultOrgUnitExternalId": "REGION_NW"`
	`}`
	
	`headers = { "x-api-key": "XXXXXXXXXXXXXXX"}`
	`r = requests.post("https://api.uk2.sheassure.net/v1/user", headers=headers, json = user_object)`
	
	`r.raise_for_status()`
	`print(f"StatusCode={r.status_code}")`
	`print(f"Body={r.content}")`
	
	The following shows the output from the above Python code being run where a successful response is generated. The StatusCode=200 indicates that the user creation was successful.
	
	StatusCode=200
	Body=b'{"message": "User successfully created."}'

Update a user with a POST request

The Customer API allows the updating of users in Assure using the /v1/user API method with the POST verb. The users details are supplied in the body of the API request using the User JSON object (see section above). Users are uniquely identified by their login user name, which is the value in the username field of the User JSON object. If there is an existing user record for the login user name then the /v1/user API method will update the existing user record to match the supplied details (see the 'Update a user' section below for details on the behaviour when updating an existing user). If the existing user record is disabled then the user will be re-enabled.

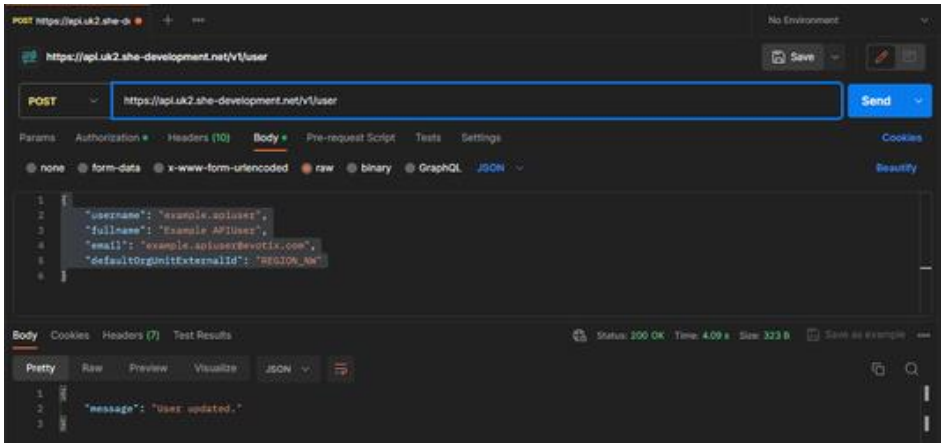
If there is no user record for the login user name then the /v1/user API method will create the user using the details in the User JSON object (see the 'Create a user' section above for details on the behaviour when creating a user).

Before attempting to use the /v1/user API method make sure you understand the User JSON object (see the section above) and that you have the required details to access the API (i.e. the URL, API key, etc). The following table sections show how to update a user using the /v1/user API method in a variety of software tools / platforms:

Partial updates are not supported with a POST request. If you wish to perform a Partial Update please see the section Partial update of a user record with PATCH request.

If you try to update a record with using the POST request Assure will update the person register record to match the details supplied, this includes applying the default values for any fields that are not provided in the User JSON POST object. See the OpenAPI schema for details of how the user data will be defaulted for each field.

Any HTTP response code is the range 200-299 should be treated as success (do not use the contents of the response body for identifying success or failure as these may change).

Postman API platform	The screenshot below shows a successful 'update user' request for a customer on the uk2 stack (NB: the Postman setup is identical to that used for creating a user). The API key is supplied via the 'Authorization' tab. The response section at the bottom shows the result Status: 200 OK which indicates that the user update was successful (also displayed is the response message from the Customer API confirming the successful update). NB: With tools like this where the User JSON object is manually entered you must ensure the contents of any string values are properly escaped. 

EVOTIX

Windows Powershell	The code block below shows the few lines of Powershell script required to setup the User Object JSON and to make the 'update user' request to the Customer API for a customer on the uk2. The XXXXXXXXXXXXXXX is where the API key goes. `$UserObjectJSON = @{` `  "username" = "example.apiuser"` `  "fullname" = "Example APIUser"` `  "email" = "example.apiuser@evotix.com"` `  "defaultOrgUnitExternalId" = "REGION_NW"` `} | ConvertTo-Json` `Invoke-WebRequest `` `  -Headers @{'x-api-key' = 'XXXXXXXXXXXXXXX' } `` ` -Uri https://api.uk2.sheassure.net/v1/user `` `  -Method Post `` ` -ContentType 'application/json' `` `  -Body $UserObjectJSON`` The following shows the output from the above Powershell script code being run where a successful response is generated. The StatusCode: 200 indicates that the update user request was successful. `StatusCode : 200` `StatusDescription : OK` `Content : {"message":"User updated."}`

Python	The code block below shows the few lines of Python code required to setup the User Object JSON and to make the 'updateuser' request to the Customer API for a customer on the uk2 stack. The XXXXXXXXXXXXXXX is where the API key goes.
	
	`# pip install requests`
	`import requests`
	
	`user_object = {`
	`"username": "example.apiuser",`
	`"fullname": "Example APIUser",`
	`"email": "example.apiuser@evotix.com",`
	`"defaultOrgUnitExternalId": "REGION_NW"`
	`}`
	
	`headers = { "x-api-key": "XXXXXXXXXXXXXXX"}`
	`r = requests.post("https://api.uk2.sheassure.net/v1/user", headers=headers, json = user_object)`
	
	`r.raise_for_status()`
	`print(f"StatusCode={r.status_code}")`
	`print(f"Body={r.content}")`
	
	The following shows the output from the above Python code being run where a successful response is generated. The StatusCode=200 indicates that the user update was successful.
	
	`StatusCode=200`
	`Body=b'{"message": "User updated."}'`

Partial update of a user record with PATCH request

The Customer API allows the partial update of Users in Assure using the /v1/user/{username} API method with the PATCH verb. A PATCH request will only update the fields provided in the request and any other fields will remain unchanged.

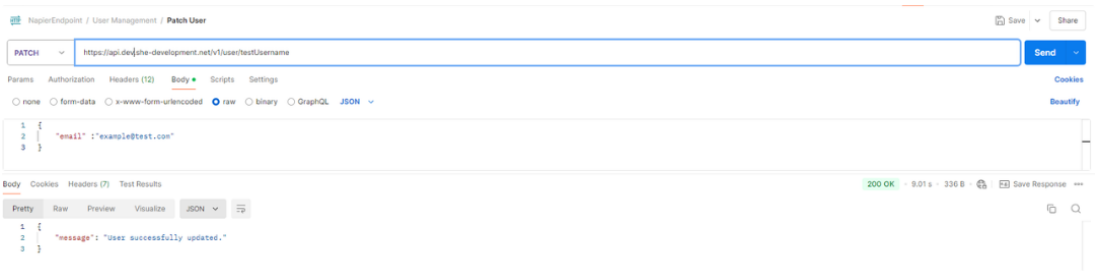
The record that will be updated will be identified by its reference provided from the {username} path parameter. The user details to be updated are supplied in the body of the API request using the user JSON PATCH object. If there is an existing user for the username provided then the /v1/user/{username}API method will update the existing user with the supplied details. Note: If there is no user existing with the username that is provided in the request this will result in an error response (HTTP error 404).

Before attempting to use the /v1/user/{username} API method make sure you understand the user JSON PATCH object (see the section above) and that you have the required details to access the API (i.e. the URL, API key, etc), follow the Getting Started guide if you don't have these details already. The following table sections show how to perform a

partial update of a user using the /v1/user/{username} API method in a variety of software tools / platforms.

Note: All examples shown below are for a partial update of the 'email' field. All fields in the JSON PATCH object can also be used for a partial update.

Any HTTP response code is the range 200-299 should be treated as success (do not use the contents of the response body for identifying success or failure as these may change). Implement effective error handling as per the guide for error handling.

Postman API platform	The screenshot below shows a successful 'partial update of a user' request for a customer. (NB: the Postman setup is very similar to that used for creating a user). The API key is supplied via the 'Authorization' tab (see the Getting Started guide for how to setup the API key). The response section at the bottom shows the result Status: 200 OK which indicates that the user update was successful (also displayed is the response message from the Customer API confirming the successful update). NB: With tools like this where the User JSON object is manually entered you must ensure the contents of any string values are properly escaped. 

Windows Powershell	The code block below shows the few lines of Powershell script required to setup a partial update using the User Object Patch JSON. It also shows how to make the Patch partial update request to the Customer API for a customer on the uk2 stack. The XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX is where the API key needs to be placed. **IMPORTANT** - Note the use of [System.Net.WebUtility]::UrlEncode to ensure that the reference value is properly escaped for inclusion in the URL. `$UserObjectJSON = @{` `  "email"  = "example@test.com"` `} | ConvertTo-Json` `Invoke-WebRequest `` `  -Headers @{'x-api-key' =` `'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'} `` `  -Uri https://api.uk2.sheassure.net/v1/user/ +` `[System.Net.WebUtility]::UrlEncode("testUsername")` ` -Method Patch `` `  -ContentType 'application/json' `` ` -Body $UserObjectJSON`` The following shows the output from the above Powershell script code being run where a successful response is generated. The StatusCode: 200 indicates that the update user request was successful. `StatusCode  : 200` `StatusDescription : OK` `Content  : {"message":"User updated."}`

Python	The code block below shows the few lines of Python code required to setup the User JSON PATCH Object and to make the 'update user request' request to the Customer API for a customer on the uk2 stack (NB: this is identical to the script for creating a user record). The XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX is where the API key needs to be placed.
	
	`# pip install requests`
	`# pip install urllib`
	`import requests`
	`import urllib`
	`user_object = {`
	`"email": "example@test.com"`
	`}`
	`headers = { "x-api-key": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"}`
	`url = "https://api.uk2.sheassure.net/v1/user/" +`
	`urllib.parse.quote('testUsername', safe='')`
	`r = requests.patch(url, headers=headers, json = user_object)`
	`r.raise_for_status()`
	`print(f"StatusCode={r.status_code}")`
	`print(f"Body={r.content}")`
	
	The following shows the output from the above Python code being run where a successful response is generated. The StatusCode=200 indicates that the user update was successful.
	
	`StatusCode=200`
	`Body=b'{"message": "User updated."}'`

Delete a user

The Customer API allows the deleting of users in Assure using the /v1/user/{username} API method with the DELETE verb. The user to be deleted is identified by their login username supplied via the {username} path parameter. The User Object is NOT required for this method.

If there is no user record for the login user name or the user is already deleted then the /v1/user/{username} API method will still return successfully response (i.e. HTTP status code in the range 200-299).

Before attempting to use the /v1/user/{username} API method make sure you have the required details to access the API (i.e. the URL, API key, etc). The following table sections show how to disable a user using the /v1/user/{username} API method in a variety of software tools / platforms:

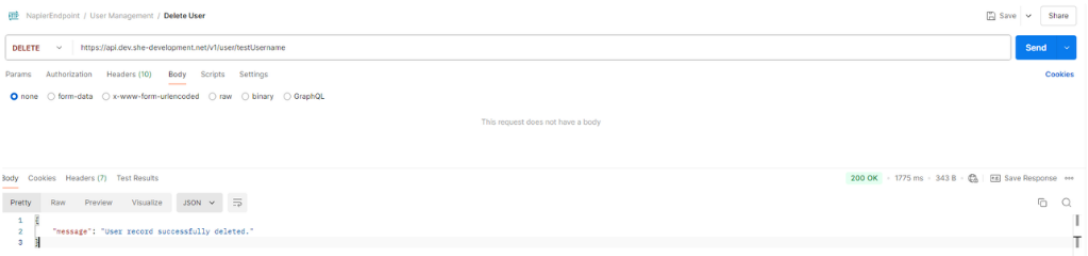
There are some scenarios where you cannot delete users, these are the same as in the UI and an appropriate error message will be returned explaining the reason if this is the case.

These scenarios include:

- User is a System user
- User is set as an approver
- User is set as a reviewer
- User is assigner to tasks
- User is a notification user
- User is a portal notification user
- User is a Auto Archive Recipient
- User is a Seven Day Recipient
- User is a HR resource for an org unit
- User is a portal user
- User is a dashboard owner
- User has any HR records
- User has any rules assigned to them
- User has any notifications assigned to them
- User has any outstanding tasks
- AGO+ action users (assignees or raisers)

Any HTTP response code in the range 200-299 should be treated as success (do not use the contents of the response body for identifying success or failure as these may change).

IMPORTANT - The login user name value needs to have URL escaping applied before inclusion in the URL.

Postman API platform	The screenshot below shows a successful 'delete user' request for a customer on the uk2 stack. Note the use of the 'Authorization' tab to configure the API key header. The XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX value is where the API key would be placed. The response section at the bottom shows the result Status: 200 OK which indicates that the user disable was successful (also displayed is the response message from the Customer API confirming the successful disable). NB: With tools like this where the login user name to be deleted is manually entered you must ensure that URL escaping is applied to the username. 

Windows Powershell	The code block below shows the few lines of Powershell script required to make the 'delete user' request to the Customer API for a customer on the uk2 stack. The XXXXXXXXXXXXXXXXXXXXXXXX is where the API key needs to be placed.
	
	IMPORTANT - Note the use of [System.Net.WebUtility]::UrlEncode to ensure that the login username value is properly escaped for inclusion in the URL.
	
	`$URL = "https://api.uk2.sheassure.net/v1/user/" + [System.Net.WebUtility]::UrlEncode("example?user")`
	`Invoke-WebRequest ` `-Headers @{'x-api-key' = 'XXXXXXXXXXXXXXXXXXXXXXXX'} ` `-Uri $URLEscaped ` `-Method Delete ``
	
	The following shows the output from the above Powershell script code being run where a successful response is generated. The StatusCode: 200 indicates that the disable user request was successful. Some response lines have been removed for brevity.
	
	`StatusCode : 200`
	`StatusDescription : OK`
	`Content : {"message": "User successfully deactivated."}`

Python	The code block below shows the few lines of Python code required to make the 'delete user' request to the Customer API for a customer on the uk2 stack. The XXXXXXXXXXXXXXXXXXXXXXXX is where the API key needs to be placed.
	
	IMPORTANT - Note the use of urllib.parse.quote to ensure that the login username value is properly escaped for inclusion in the URL.
	
	`# pip install requests`
	`# pip install urllib`
	`import requests`
	`import urllib`
	
	`url = "https://api.uk2.sheassure.net/v1/user/" +`
	`urllib.parse.quote('example.apiuser', safe='')`
	`headers = { "x-api-key": "XXXXXXXXXXXXXXXXXXXXXXXX"}`
	`r = requests.delete(url, headers=headers)`
	
	`r.raise_for_status()`
	`print(f"StatusCode={r.status_code}")`
	`print(f"Body={r.content}")`
	
	The following shows the output from the above Python code being run where a successful response is generated. The StatusCode=200 indicates that the user disable was successful.
	
	`StatusCode=200`
	`Body=b'{"message": "User successfully deactivated."}'`