API for Lead Generation

Overview

This API is used to configure products and request a proposal in the Lead Generation Channel. This API is an optional feature activated with the license.

The API works for both products based on product models and TCX configuration models but there are some extra details to consider for product models.

API General Info

Response Content Type

The API only supports the `json` response type.

Successful Responses

Response Codes

- **200** - The request was executed successfully and the server returned some data. The return content type is specified inside a response header called `content-type`. Example content types are `application/xml`, `application/json`, `application/zip` and `text/plain`.
- **204** - The request was executed successfully and the server returned no data. If the `Location` header was present then the server responded with a redirect.

Error Responses

If the API call resulted in an error then the error code will be returned along with the error object in the response body. The error object contains the attributes listed below:

- **type** - type of the error
- **cause** - the root cause of the error
- **message** - message containing details of the related error
- **uuid** - a unique identifier of the error which also may appear in the exception logs
- **timestamp** - the time of the error

Note: For chunked responses, the socket will be closed prematurely if an error occurs after the initial chunk containing the status code has been sent.

Response Codes

- **400** - The request contains invalid parameters or there are invalid settings. Consult the error message and/or check Administration for invalid settings.
- **401** - The user is not authenticated.
- **403** - The user is not authorized access to the specified endpoint.
- **404** - The specified endpoint does not exist or a request parameter is invalid. Consult the error message.
- **405** - The specified endpoint does not support the requested operation. Consult the error message.
- **409** - The specified endpoint is in a state incompatible with the requested operation. Consult the error message.
- **500** - Unknown error occurred while processing the request. Consult the CPQ logs for more information.
- **504** - The specified concurrent calls limit was exceeded and the server was not able to serve the response in a specified timeout limit. That error is always served with the xml body containing the description.

API Keys

All the operations in the application require a context of an organization and role and the API for Lead Generation is no exception. Functions in this API require an API key which the administrator sets up in advance. Each API key corresponds to an organization and role.

It is possible to have multiple API keys registered. Keys are managed at *Sales Process* > *Integration* > *Lead Generator*.

Setting	Description
Name	Name of the key.
Key	The key to use for authentication.
Role	The role in which the API acts when authenticated with this key.
Organization	The organization in which the API acts when authenticated with this key.

Note

It is recommended to send the API key as the request header `X-Key`. The support for query parameter `_key` will eventually be phased out.

Translations

Most endpoints support the optional query parameter `language` or request header `X-Language`. Both expect a language code as value. Translatable strings in the response will be translated if used.

See Languages and Translations for more details.

Configuration State

The configurator engine is stateless and requires that callers maintain the configuration session state throughout a configuration. Most functions in this API require a state and most functions provide an updated state.

The state is represented by a string of text that contains information about which product is being configured, which parameters have been committed etc. The length of the state string depends on the configuration model used, there is no hard upper limit, but typical sizes are well below 10,000 characters.

All API functions that modify (or create) a configuration session state return a new configuration state for use in following calls.

Product Reference

The start function requires that you specify which product id to use. This API does not provide any means to list products but there are two ways of finding it:

- Manually go to `/products/list` and copy the internal id of the product from the address bar.
- Use API for Direct Sales to list all product instances and get your id.

Needs and Sizing for Product Models

Product models which implement Needs or Sizing will receive corresponding steps in the configuration state, if input to any is required to advance in the configuration.

If a product model uses the concept of *contextual needs* then values for those needs must be set at start of configuration. This is achieved by assigning an alias to the need (where needs are managed on the product model) and then providing the alias as a request parameter with the desired value.

Sample

Assuming there are two needs defined on a product model, using aliases `region` and `size` , the following request parameters could be used to set the values for these needs:

```
&region=europe&size=large
```

Field Display Properties

See API for Self Service documentation.

Configurator JSON Structure

```
{
    "configState" : "<state>",
    "steps" : [
        {
            "name" : "<name>",
            "description" : "<description>",
            "current" : true|false,
            "available" : true|false,
            <FOR CURRENT STEP ONLY:>
            "rootGroup": {
                "name" : "<name>",
                "description" : "<description>",
                "hasVisibleParameters" : true|false,
                "members": [
                    "isGroup" : true|false,
                    "isParameter" : true|false,
                    "name" : "<name>",
                    "description" : "<description>",
                    "properties" : {
                        "<prop1>" : "<val1>",
                        "<multivalprop2>" : ["<val1>", "<val2>"],
                        <...>
                    }
                    <FOR PARAMETERS ONLY:>
                    "value" : "<value>",
                    "valueDescription" : "<valueDescription>",
                    "committed" : true|false,
                    "domain" : {
                        "name" : "<name>",
                        "min" : <number>,
                        "max" : <number>,
                        "elements" : [
                            {
                                "name" : "<name>",
                                "description" : "<description>",
                                "state" : "green|orange",
                                "selected" : true|false
                            }
                        ]
                    }
                    <FOR GROUPS ONLY:>
                    "hasVisibleParameters" : true|false,
                    "members": [<recursive structure>]
                ]
            }
        },
        <more steps...>
    ],
    "response" : {
        "status" : "OK|RESOLVABLE|not OK",
        "message" : "<message>",
```

```
        "not-applied": [
            {
                "name": "<parameter name>",
                "value": "<parameter value>"
            },
            ...
        ],
        "error" : {
            "description" : "<description>",
            "info" : "<info>",
            "location" : "<location>",
            "status" : "<status>",
            "type" : "<type>",
        },
        "changed" : {
            "name" : "<description>",
            "description" : "<description>",
            "oldValue" : "<oldValue>",
            "oldValueDescription" : "<oldValueDescription>",
            "newValue" : "<newValue>",
            "newValueDescription" : "<newValueDescription>"
        }
    }
}
```

Bill of Materials JSON Structure

```
{
    "bom" : [
        {
            "name": "<name>",
            "description": "<description>",
            "qty": <number>,
            "attributes": {
                "<attr1>" : "<attr1val>",
                ...
            },
            "subItems", [
                <Recursive structure>
            ],
        }
    ],
    "status" : "OK"
}
```

Functions

Start Configuration

Setting	Description
Method	POST
Address	/configurator-api/start
Required Parameters	`X-Key` The API key header (or `_key` query parameter). `product` : Internal id of a product to configure.
Other Parameters	Additional key value pairs for setting context alias values. `language` parameter or `X-Language` header (with language code). `group` : the name of the focused configuration group. When supplied, the consistency checking is limited to parameters of the group, usually improving performance. Contextual Needs.
Usage	Starts a new configuration session.
Return	configState

Sample

Start a new configuration session for `product=3aad0606c4a14cccb2259667c906700b` .

```
POST /configurator-api/start?product=3aad0606c4a14cccb2259667c90670
X-Key: abc123
```

Commit a Parameter

Setting	Description
Method	POST
Address	/configurator-api/commit
Required Parameters	`X-Key` The API key header (or `_key` query parameter). `configState` : The current configurator state. `par` : The name of a parameter to commit. `val` : The value to commit.
Other Parameters	Additional `par` and `val` pairs for multi-commit. `language` parameter or `X-Language` header (with language

Setting	Description
	code). `group` : the name of the focused configuration group. When supplied, the consistency checking is limited to parameters of the group, usually improving performance.
Usage	Performs a commit of a parameter to a value. If this commit conflicts with other commits, the operation is cancelled and a status of RESOLVABLE is returned with a suggestion of operations that resolves the conflict. If the value specified is outside of the valid range of values for the specified parameter, the operation is cancelled and a status of UNRESOLVABLE is returned. Note It's possible to set several parameters at once (aka multi-commit) by adding several par/val pairs in the request.
Return	configState

Sample

Set `Market_field=Market_Norway` for a started configuration with `configState=H4sIAAAAAAAAAHWRUWvDIBSFf1GqJs4` .

```
POST /configurator-api/commit?configState=H4sIAAAAAAAAAHWRUWvDIBSFf
X-Key: abc123
```

Example of setting several parameters at the same time.

```
POST /configurator-api/commit?configState=H4sIAAAAAAAAAHWRUWvDIBSFf
X-Key: abc123
```

Accept a Conflict

Setting	Description
Method	POST
Address	/configurator-api/accept

Setting	Description
Required Parameters	`X-Key` The API key header (or `_key` query parameter). `configState` : The current configurator state. `par` : The name of a parameter to commit. `val` : The value to commit.
Other Parameters	`language` parameter or `X-Language` header (with language code). `group` : the name of the focused configuration group. When supplied, the consistency checking is limited to parameters of the group, usually improving performance.
Usage	Performs a commit of a parameter to a value without cancelling if the commit is in conflict. The parameter and value from the 'response' block of a response to /commit call which caused a conflict should be used in the /accept call.; instead performs the operations as suggested by the configurator to resolve the conflict. If the value specified is outside of the valid range of values for the specified parameter, the operation is cancelled and a status of UNRESOLVABLE is returned.
Return	configState

Sample

Set `Market_field=Market_Norway` for a started configuration with `configState=H4sIAAAAAAAAAHWRUWvDIBSFf1GqJs4` even if the given value conflicts with another value.

```
POST /configurator-api/accept?configState=H4sIAAAAAAAAAHWRUWvDIBSFf
X-Key: abc123
```

Uncommit a Parameter

Setting	Description
Method	POST
Address	/configurator-api/uncommit
Required Parameters	`X-Key` The API key header (or `_key` query parameter). `configState` : The current configurator state.

Setting	Description
	`par` : The name of a parameter to uncommit.
Other Parameters	`language` parameter or `X-Language` header (with language code). `group` : the name of the focused configuration group. When supplied, the consistency checking is limited to parameters of the group, usually improving performance.
Usage	Uncommits a parameter. Does nothing if parameter is not committed.
Return	configState

Sample

Unset parameter `Market_field` .

```
POST /configurator-api/uncommit?configState=H4sIAAAAAAAAAHWRUWvDIBS
X-Key: abc123
```

Change Step

Setting	Description
Method	POST
Address	/configurator-api/step
Required Parameters	`X-Key` The API key header (or `_key` query parameter). `configState` : The current configurator state. `step` : The name of the step to switch to.
Other Parameters	`language` parameter or `X-Language` header (with language code). `group` : the name of the focused configuration group. When supplied, the consistency checking is limited to parameters of the group, usually improving performance.
Usage	Jump between steps.
Return	configState

Sample

Change step to `Detailed`.

```
POST /configurator-api/step?configState=H4sIAAAAAAAAAHWRUWvDIBSFf1G
X-Key: abc123
```

Get BoM

Setting	Description
Method	POST
Address	/configurator-api/bom
Required Parameters	`X-Key` The API key header (or `_key` query parameter). `configState` : The current configurator state.
Other Parameters	`language` parameter or `X-Language` header (with language code).
Usage	Returns a bill-of-materials for the current state of the configuration.
Return	bom

Sample

Get the BoM for the current configuration.

```
POST /configurator-api/bom?configState=H4sIAAAAAAAAAHWRUWvDIBSFf1GC
X-Key: abc123
```

Get Visualization

Setting	Description
Method	POST
Address	/configurator-api/visualization

Setting	Description
Required Parameters	`X-Key` The API key header (or `_key` query parameter). `configState` : The current configurator state.
Other Parameters	`group` : the name of the focused configuration group. When supplied, the consistency checking is limited to parameters of the group, usually improving performance.
Usage	Returns a visualization object for the current state of the configuration.
Return	The visualizationObject in JSON format to be passed on to the Visualization.

Sample

Get the visualization object for the current configuration.

```
POST /configurator-api/visualization?configState=H4sIAAAAAAAAAHWRUW
X-Key: abc123
```

Create Lead

Setting	Description
Method	POST
Address	/configurator-api/lead
Required Parameters	`X-Key` The API key header (or `_key` query parameter). `configState` : The current configurator state.
Other Parameters	Populate attributes on the lead object with any parameters named like the attributes.
Usage	Create an instance in *Lead* object with the configuration state. This lead can be converted to a solution with a single configured product.
Return	Returns a reference of the created lead.

Sample

Create an instance in *Lead* object and set parameters `customerid` and `email`.

```
POST /configurator-api/lead?configState=H4sIAAAAAAAAAHWRUWvDIBSFf1C
X-Key: abc123
```

Visualization

It is recommended to use the Tacton Visualization JavaScript Library when integrating the visualization in a web application utilizing the API for Lead Generation. This library contains the basic functionality of the iFrame hosting the 3D visualization and is available as part of every tenant.

Example

Load the javascript library provided by your tenant.

```
<script src="https://NAME.tactoncpq.com/js/tacton.vis.js"></script>
```

Initialize Visualization

The main function provided by this javascript library is called *tactonVis* and sets up the visualization based on the current configuration result. Subsequent calls are needed to keep the visualization updated with the current configuration.

```
tactonVisObject = tactonVis(containerHTMLElement, visualizationObject, conf
```

Parameter	Details
containerHTMLElement	The parent HTMLElement into which the visualization iframe will be inserted or where it currently exists in case of a subsequent call. Typically a DIV container on the web site where the visualization should be displayed.
visualizationObject	The JSON object returned when requesting visualization-relevant information Get Visualization.

<table>
<tr><th>Parameter</th><th>Details</th></tr>
<tr><td></td><td>It contains the location of all files needed to initialize the visualization:

<table>
<tr><td><code>assetsPath</code></td><td>String</td><td>URL where the visual assets are located.</td></tr>
<tr><td><code>customImagesRootPath</code></td><td>String</td><td>Optionally URL where the custom visualization images are located.</td></tr>
<tr><td><code>iframePath</code></td><td>String</td><td>URL of the visualization version being used.</td></tr>
<tr><td><code>screenshotRenderer</code></td><td>String</td><td>URL of the server processing screenshot generation requests.</td></tr>
<tr><td><code>sessionServer</code></td><td>String</td><td>Location of the web socket server processing session sharing requests.</td></tr>
<tr><td><code>tvkbPath</code></td><td>String</td><td>URL where the visual logic is located.</td></tr>
<tr><td><code>tvkbReference</code></td><td>String</td><td>Name of the visual logic file being used.</td></tr>
</table>

It contains information about available buttons on the visualization UI:</td></tr>
</table>

<table>
<tr><th>Parameter</th><th>Details</th></tr>
<tr><td></td><td>
<table>
<tr><td><code>debugMode</code></td><td>Boolean</td><td>If the button to copy the current configuration to the clipboard is available.</td></tr>
<tr><td><code>fullscreen</code></td><td>Boolean</td><td>If the button to switch the visualization to fullscreen is available.</td></tr>
<tr><td><code>share</code></td><td>Boolean</td><td>If the button to open the sharing dialog is available.</td></tr>
<tr><td><code>visaction</code></td><td>Boolean</td><td>If the buttons executing project specific functionality are available.</td></tr>
</table>
It contains visualization relevant configuration information:
<table>
<tr><td><code>currentStep</code></td><td>String</td><td>Name of the configuration step that is currently selected.</td></tr>
<tr><td><code>customImageData</code></td><td>String</td><td>Available and selected custom visualization images.</td></tr>
<tr><td><code>messagingData</code></td><td>[{String,String}]</td><td>Array of visualization relevant configuration attributes and their current values.</td></tr>
</table>
It contains information about the visualization:
</td></tr>
</table>

<table>
<tr><th>Parameter</th><th>Details</th></tr>
<tr><td></td><td>
<table>
<tr><td><code>name</code></td><td>String</td><td>Name of the visualization object in CPQ.</td></tr>
<tr><td><code>screenshots</code></td><td>[{Object}]</td><td>Array of objects containing information about available screenshots.</td></tr>
</table>
It contains style information:
<table>
<tr><td><code>aspectRatio</code></td><td>String</td><td>Aspect ratio of the Visualization (e.g. "1:1").</td></tr>
<tr><td><code>primaryColor</code></td><td>String</td><td>Color used for loading spinner (e.g. "#ff00ff").</td></tr>
</table>
It is possible to adjust the JSON object before passing it on, to adjust the visualization according to the lead generation use case. For Example removing buttons on the visualization UI.
And it is also possible to add additional parameters to the visualizationObject that are not provided by the configurator in a lead generation use case:
<table>
<tr><td><code>currentGroup</code></td><td>String</td><td>The current configuration group name being displayed. This parameter may be required by a visualization if it was designed to use it.</td></tr>
<tr><td><code>useCSP</code></td><td>Boolean</td><td>Determines if the Content Security Policy meta tags will be rendered in the visualization iframe. By default, the lack of this</td></tr>
</table>
</td></tr>
</table>

Parameter	Details
	parameter evaluates to true.
configResponseObject	*Optionally* The response sub-object returned in every configuration result (JSON). It contains information to display configuration errors or conflicts. This information will not be passed if parameter is omitted.
triggerObject	*Optionally* The object to send builtin triggers ("toggleFullscreen", "share", "copyconfig") or vis actions (button names) to the visualization.

Return value	Details
tactonVisObject	*tactonVis* returns this object on which the visualization messaging handlers can be registered, see Send Messages to VIZ for more details.

Order of processing inside the visualization:

1. Configuration update
2. Update group
3. Handling of triggers

Example

```
// initialize the visualization
var data = {};
data["configState"] = response.configState;
// response as returned from /configurator-api/start, /configurator
var containerHTMLElement;
var visualizationObject;
var xhttp = new XMLHttpRequest();
xhttp.open("POST", "https://NAME.tactoncpq.com/configurator-api/vis
xhttp.setRequestHeader("X-Key", "abc123");
xhttp.setRequestHeader('Content-type', 'application/x-www-form-urle
xhttp.onreadystatechange = function(event) {
if (xhttp.readyState == XMLHttpRequest.DONE && xhttp.status == 200)
visualizationObject = JSON.parse(xhttp.responseText);
containerHTMLElement = document.getElementById(visualizationObject.
}
};
xhttp.onerror = function(event) {
//error code here
};
let body = "configState=" + encodeURIComponent(data["configState"])
xhttp.send(body);
```

Trigger Visualization events

To trigger specific visualization events, a subsequent call of the tactonVis object is needed.

Event	Type	Description
toggleFullscreen	triggers	Toggles the fullscreen mode
share	triggers	Shows the share dialog including QR code
copyconfig	triggers	Copies the current configuration into the clipboard to be used in the VIZstudio preview
<buttonname>	buttonTriggers	Trigger action button

Example

```
// send a fullscreen trigger
function sendFullscreenTrigger() {
tactonVis(containerHTMLElement, visualizationObject, {}, {"triggers
}
// send a share trigger
function sendShareTrigger() {
tactonVis(containerHTMLElement, visualizationObject, {}, {"triggers
}
// send a copyconfig trigger
function sendCopyConfigTrigger() {
tactonVis(containerHTMLElement, visualizationObject, {}, {"triggers
}
// send a vis action trigger
function sendVisActionTrigger(name) {
tactonVis(containerHTMLElement, visualizationObject, {}, {"buttonTr
}
```

VIZ Message Handling

The "tactonVisObject", provided as a result of calling the "tactonVis" function, is required to update the external web application, or the configuration when interacting directly with visualization. The messageHandler callback function will be called every time the visualization detects relevant user interaction or other backend events. The callback function will be called with a data object parameter containing information about the event. For example, the current configuration attributes set on the visualization.

Note

Subsequent calls to the *onMessage* method will register multiple callbacks.

Types of messages sent to the messageHandler:

Action	Details	Example message
multicommit	Configuration change request sent from visualization.	{"action": "multicommit", parameters: {"button1": true, "button2": false}}
group	Group change request sent from visualization.	{"action": "group", "value": "options"}
button_visibility	Vis action visibility change notification sent from visualization.	{"action": "button_visibility", parameters: {"button1": true, "button2": false}}

Example

```
tactonVisObject = tactonVis(containerHTMLElement, visualizationObje
tactonVisObject.onMessage(messageHandler);
funtion messageHandler(message, visObject) {
switch(message.action) {
case "multicommit":       console.log(message.action + " parameters
case "group":             console.log(message.action + " value: "
case "button_visibility": console.log(message.action + " parameters
};
}
```