Uninstall Applications

This action occurs after an application is uninstalled from a sandbox.

<table>
<thead>
<tr>
<th>Action Type</th>
<th>Embedded</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Action ID</td>
<td>embedded.platform.applications.uninstall</td>
</tr>
<tr>
<td>Runs multiple custom functions?</td>
<td>Yes</td>
</tr>
</tbody>
</table>

JavaScript File Structure

Action files share the following basic structure:

```
module.exports = function(context, callback) {
    // Your custom code here
callback();
};
```

When you code the custom function for an action, you have access to two arguments:

- **callback** — This argument follows the established JavaScript callback pattern: it takes an error as the first argument (or null if there is no error) and a result as the second argument (if required).

- **context** — This argument provides the function access to relevant objects and methods that interface with Kibo.

Context: Platform

The following methods and objects are available to this action through the use of the `context` argument.

Microservice Operation

This action corresponds to the microservice that installs and uninstalls applications.

Get Methods

- `get.applicationKey`
- `get.exports`
- `get.installationState`
- `get.nameSpace`

Exec Methods

- `exec.saveInstallationState`

Context Objects Available to All Actions
Get

get.applicationKey

Returns the application key.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Example:

context.get.applicationKey();

Response:

"string"

get.exports

Returns the exports for the application.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Example:

callback.get.exports();

Response:

"object"

get.installationState

Returns the installation state of the application.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Example:
context.get.installationState();
Response:
"object"

get.nameSpace

Returns the namespace of the application.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Example:
context.get.nameSpace();
Response:
"string"

**Exec**

eexec.saveInstallationState

Saves the supplied application installation state to Kibo.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>installationState</td>
<td>object</td>
<td>The application installation state. You can retrieve this object using the corresponding GET method.</td>
</tr>
</tbody>
</table>

Example:
context.exec.saveInstallationState(installationState);
Response: N/A

**Context Objects Available to All Actions**

**apiContext**

Accesses tenant information.
<table>
<thead>
<tr>
<th>Properties</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseUrl</td>
<td>string</td>
<td>The base URL for the site.</td>
</tr>
<tr>
<td>basePciUrl</td>
<td>string</td>
<td>The base PCI URL for the site.</td>
</tr>
<tr>
<td>tenantPod</td>
<td>string</td>
<td>The name of the tenant pod in which the tenant resides.</td>
</tr>
<tr>
<td>appClaims</td>
<td>string</td>
<td>The application claims token.</td>
</tr>
<tr>
<td>appKey</td>
<td>string</td>
<td>The application key.</td>
</tr>
<tr>
<td>tenantId</td>
<td>integer</td>
<td>Unique identifier for the tenant.</td>
</tr>
<tr>
<td>siteId</td>
<td>integer</td>
<td>Unique identifier for the site. This ID is used at all levels of a store,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>catalog, and tenant to associate objects to a site.</td>
</tr>
<tr>
<td>masterCatalogId</td>
<td>integer</td>
<td>Unique identifier for the master catalog.</td>
</tr>
<tr>
<td>catalogId</td>
<td>integer</td>
<td>The unique identifier for the product catalog. Catalogs are part of a master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>catalog.</td>
</tr>
<tr>
<td>currencyCode</td>
<td>string</td>
<td>The default three-letter ISO currency code for monetary amounts.</td>
</tr>
<tr>
<td>previewDate</td>
<td>date/time</td>
<td>The date and time that the content is being viewed. This might be a future</td>
</tr>
<tr>
<td></td>
<td></td>
<td>date if the content is previewed with an active date range set in the future.</td>
</tr>
<tr>
<td>localeCode</td>
<td>string</td>
<td>The locale code per the country code provided. This code determines the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>localized content to use and display.</td>
</tr>
<tr>
<td>correlationId</td>
<td>string</td>
<td>The unique identifier of the API request associated with the event action,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>which might contain multiple actions.</td>
</tr>
<tr>
<td>isAuthorizedAsAdmin</td>
<td>Boolean</td>
<td>Indicates whether the Dev Account user is authorized as an admin.</td>
</tr>
<tr>
<td>userClaims</td>
<td>string</td>
<td>The user claims token.</td>
</tr>
</tbody>
</table>

Example:

```javascript
context.apiContext.baseUrl;
```

**configuration**

Receives a JSON response that contains information about the configuration data set in the Action Management JSON editor.
<table>
<thead>
<tr>
<th>Properties</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Varies</td>
<td>object</td>
<td>Custom fields and values that you can set in the Action Management JSON Editor.</td>
</tr>
</tbody>
</table>

Example:

```javascript
context.configuration.customData;
```