HUMAN HEALTH

ENVIRONMENTAL HEALTH

ChemDrawDirect 1.7.1

Developer's Guide

PerkinElmer
For the Better

Last Updated: December 02, 2016

---

title: ChemDraw Direct v1.7.1 Developer’s Guide
layout: pkidocs

1 Introduction

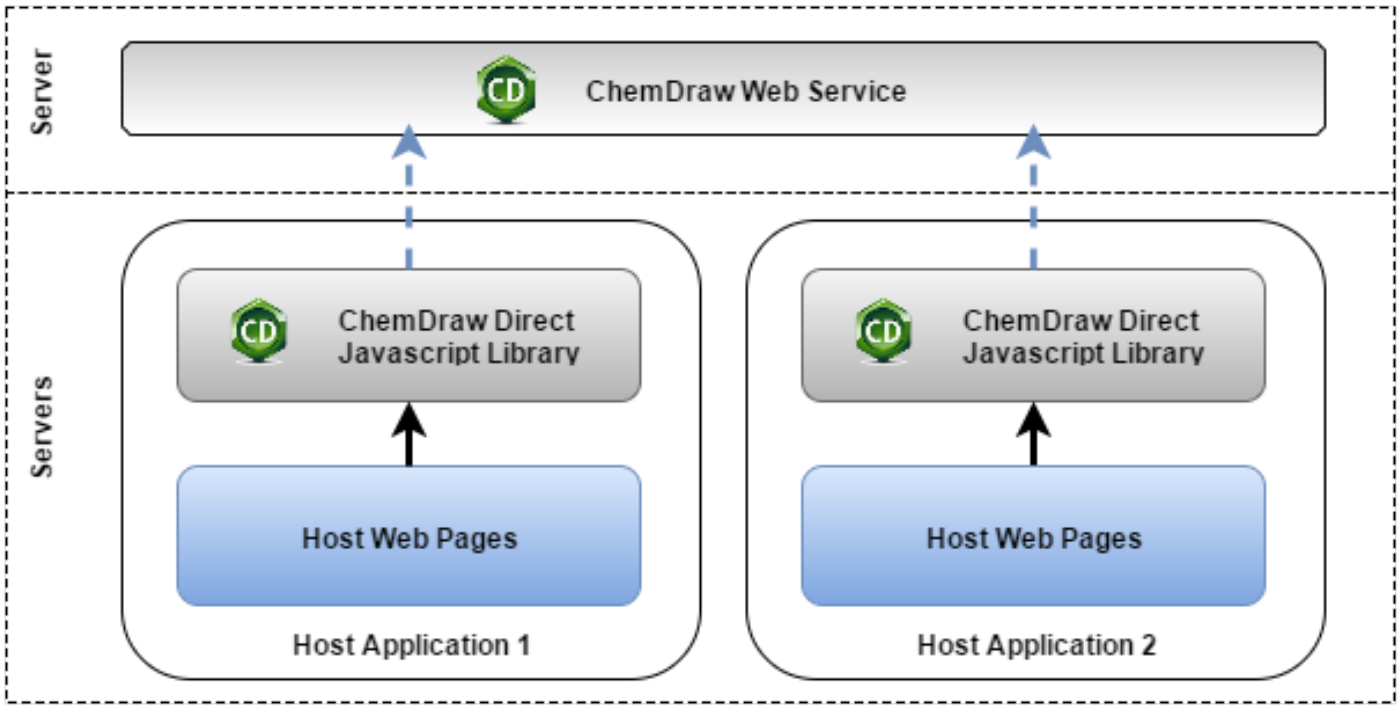
ChemDraw Direct (CDD) is a lightweight JavaScript-based chemical drawing tool built using modern web technologies that can be easily integrated into web-based applications such as PerkinElmer Elements via its published API.
This guide is designed to instruct developers on the development of web-based applications that using ChemDraw Direct. It describes the procedures for loading and attaching it, as well as API usage and functionality.

1.1 Architecture

Three typical architectures are supported by ChemDraw Direct to enable host applications to embed ChemDraw functionalities in a suitable way.
The JavaScript library is a must for the host applications to enable basic chemical drawing functions. It consists of the JavaScript source files, cursor resources, and chemical resources.
The ChemDraw Web Service is only required if the host application wants to enable advanced chemical features, such as chemical format converting, structure cleanup, stoichiometry, etc. It runs as a windows service and provides RESTful web service APIs to authorized clients.
Refer to the *PerkinElmer ChemDraw Direct 1.7.1 Installation Guide* for details about the installation steps.

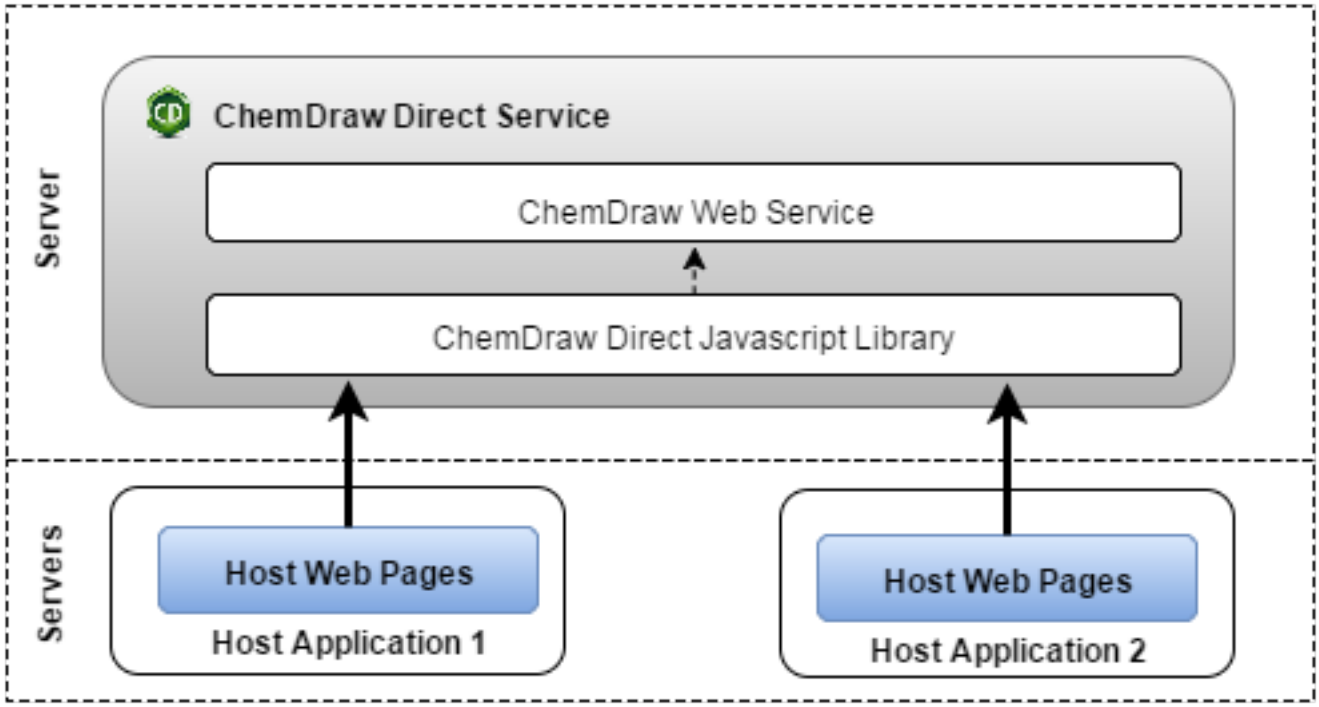
1.1.1 Separated Libraries Embedded in Host Applications

The JavaScript library can be included in any web-based applications. The host application developers have to make sure the scripts and resource files are deployed correctly with the host application. Each host application needs to maintain one copy of the library. An upgrade on the JavaScript library may cause an upgrade of the entire host application.
If the ChemDraw Web Service is needed, a separate web service has to be deployed in the network. The address of the web service has to be configured manually in the JavaScript library. The web service can be installed on the same server which has the host application running.



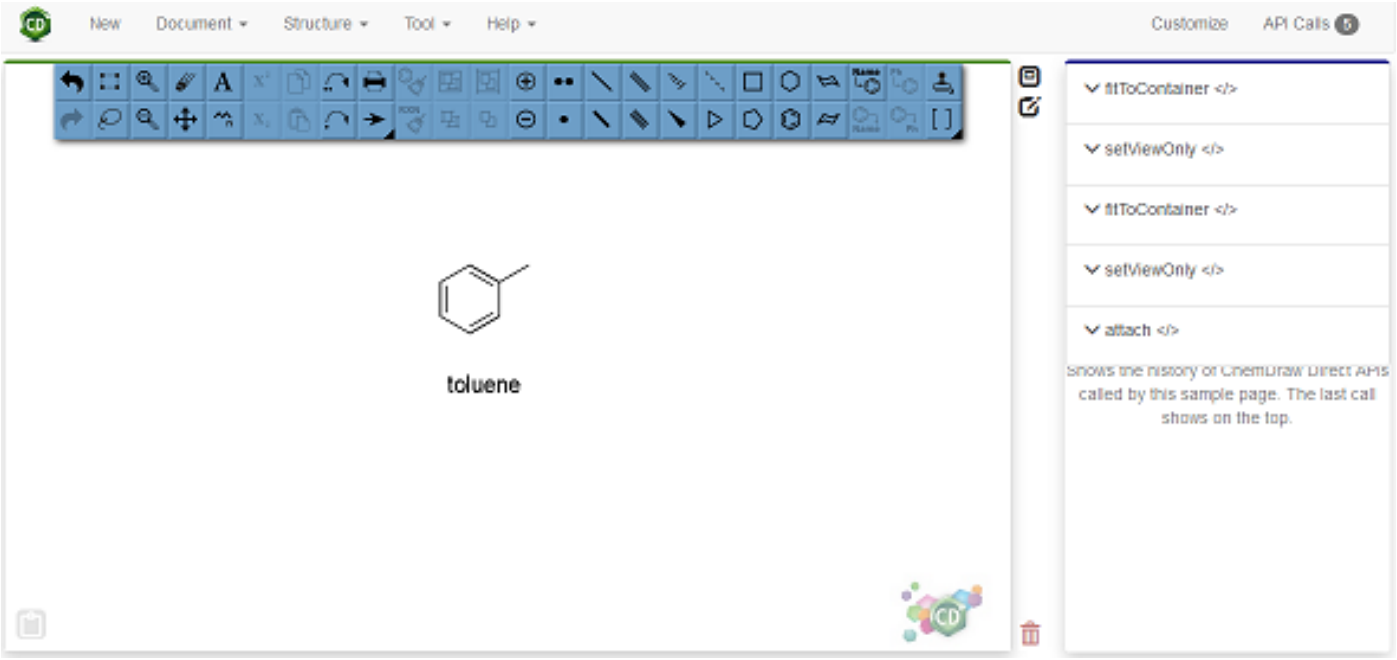
1.1.2 Single ChemDraw Direct Service

The library and the RESTful web service can be deployed within a single service using the installer of the ChemDraw Direct Service. The service provides the capability of Cross Origin Resource Sharing (CORS) so that the libraries can be accessed from the applications that running in different domains.
The address of the web service is automatically configured in the library by the installer, so that all the host applications can share the same configuration to provide ChemDraw functions to end users.
The ChemDraw Direct Service needs to be maintained by the service professionals or administrators of the host applications. Ideally, an upgrade of ChemDraw Direct Service and host applications has no impact to each other.
The service can be deployed in the same local network with the host applications.



1.2 ChemDraw Direct Sample Page

A sample page of ChemDraw Direct is distributed with the release package or the installer.



This page demonstrates how ChemDraw Direct can be embedded in a web based application and all functions of ChemDraw Direct.

- The menus on the top call the ChemDraw Direct JavaScript APIs to create new ChemDraw tools and execute document or structure related commands.
- The panel on the right shows the history of the API calls that have been triggered by this sample page.
- Multiple ChemDraw documents can be created on the page by clicking the New button. This demonstrates the right way to handle multiple ChemDraw instances within the same web page.
- The panel that showing the ChemDraw document can be resized, maximized or restored to demonstrate the impact of the containerâ€™s size change.
- Customized configuration can be applied to the current active ChemDraw drawing tool easily by using the “Customize” tool.

With the help of this sample page, it will be easier to understand the APIs that describes in the following sections.

***Note**: The sample page is using ZeroClipboard to provide “copy to clipboard” function. It requires Adobe Flash plugin enabled on the browsers.*

2 Loading the ChemDraw Direct Module

2.1 Loading the Required Scripts

To gain access to the classes you need to use ChemDraw Direct, you first need to load its module. To do this, you add the required `<script>` tag:

Once this script tag is evaluated, the loading of the module will begin. The loading of the module is asynchronous and the classes in the module will not be available until the loading is complete.

2.2 Attaching ChemDraw Direct to the DOM Tree

perkinelmer.ChemdrawWebManager.attach(options);
The attach function may be used to attach several instances of ChemDraw to the DOM and may be called at any time. The id or the element parameter has to be specified in the options.

Options

Name	Type	Description
id	string	The id of the element in the DOM tree the tool will be attached to.
element	DOM node	The DOM element the tool will be attached to.
viewonly	boolean	Whether the drawings are editable or not.
config	JSON object (configuration)	The configuration object that will be used when attaching ChemDraw.
configUrl	string (URL)	The configuration file URL that will be used when attaching ChemDraw.
license	string	The license content string.
licenseUrl	string (URL)	The license file URL.
callback	function (drawingTool) {}	The callback that will be called after the tool is attached. The ChemDraw instance will be passed back as the first parameter of the callback, which can be used to call the APIs.

2.3 Basic Template

Start with this basic HTML template below. It loads the required scripts and attaches the tool to the HTML element with the id “drawingTool”.

```
<script type="text/javascript" src="js/chemdrawweb/ChemdrawWebManager.js"></script> <div id="drawingTool" style="position: absolute; width: 100%; height: 100%; left: 0; top: 0;"></div> <script> var toolAttached = function(chemdraw) { //ChemDraw drawingTool is ready for use now. Use chemdraw instance to call the APIs }; perkinelmer.ChemdrawWebManager.attach({id: "drawingTool", callback: toolAttached, license: "â€¦"}); </script>
```

2.4 License File

The license file is required to initialize ChemDraw Direct. There is a license file embedded in the ChemDraw Direct installer. This license file has limited features enabled, and some features are not allowed to use, e.g. calling the getSVG, getEPS, getWMF, getEMF APIs.
To use those disabled APIs or PerkinElmer hosted ChemDraw web service, please contact PerkinElmer to acquire a new license file.
The license can be passed to ChemDraw Direct in the attach function by the property “licenseUrl” (the license file URL) or “license” (the content of the license file).
The license can also be passed to ChemDraw Direct by setting the variable *window.chemdrawweb_licenseUrl* (the license file URL) or *window.chemdrawweb_license* (the license file content).
***Note**: The license file is hosted by the application server. If the licenseUrl is a relative path, it is relative to the current webpage.*

3 Customizing ChemDraw Direct

3.1 Configuration File

ChemDraw Direct JavaScript library contains a configuration file named “configuration.json”. This file is the preferred configuration to be loaded when loading ChemDraw Direct JavaScript files.
The content of the configuration file is in JSON format. The JSON configuration consists of three parts:

- theme: defines the custom style of toolbar size, icons, info images, dialogs. The url of theme can be relative path or absolute path;
- features: defines the features to enable and the features to disable;
- layout: defines the orientation and the order of the tools or tool groups;
- properties: defines the application properties which is used in the library. Currently, the address of the ChemDraw Web Service is the only property can be defined in the configuration.

The sample page can be used to generate a pretty configuration text in JSON format. Refer to section *3.3. Generate Configuration using the Sample Page* for details.


```
{
theme: { name: "chemdraw", [url: "theme/chemdraw"]
},
features: {enabled: [â€¦], disabled: [â€¦]}, layout: {orientation: "â€¦", tools: [â€¦]},
properties: { chemservice: "â€¦", "StyleSheet": "ACS Document 1996", }, authentication: { type:
"Basic" }
}
```


3.1.1 Theme

- **name**: This is CDD internal theme name. It is a folder inside CDD library. Developers can add their own theme folder on their own CDD server.
- **url**: This is the external theme address, which is the link of a folder outside of CDD. In this way, developers can define customized theme without modifying CDD library. The theme folder is deployed on developersâ€™ own application server, which can be a different server from CDD server. The URL can be an absolute link or a relative link to the webpage.

name and **url** cannot exist at the same time. If developers set both name and url, CDD will only apply name theme.

In the theme folder, developers need several files to configure the theme such as index.html, xxx.css, and some image files (xxx.png, xxx.gif, xxx.jpg). Index.html is the theme configuration file. It configures the css name and which images should be replaced in CDD. xxx.css is a css style file. Its name is based on configuration in index.html. Images files, their name are based on the configuration in the index.html.

- index.html: This is a json format file, it configure the css and images information. CDD will accord this configuration to apply the css style and replace the icon in CDD.
- css: Setting the name of css file
- images: Setting the custom images
- name: Image name
- url: Image file URL


```
{
"css":"chemdraw.css", "images" : [ { "name":"ToolBackground_Down",
"url":"ToolBackground_Down.png" }, { "name":"ToolBackground_Normal",
"url":"ToolBackground_Normal.png" }, { "name":"Clipboard_Error", "url":"Clipboard_Error.png" },
{ "name":"Clipboard_Normal", "url":"Clipboard_Normal.png" } ]
}
```


- image files: A set of replaced images files. The name follows the rule. If developers input unsupported name to replace, CDD wonâ€™t apply it to replace the icon. The url is a relative link in the theme folder, this is placed with index.html on the same folder on server.

The images reference table, if developers want to replace those images, they should use the images name to input in index.html.

Image file name Default image

- chemdraw.css: The css style file overrides the style of cdd. We support developers to replace the CDD tool bar style (size, background, hover and active affect). These css style names are fixed, developers canâ€™t remove or rename them, but developers can add their expected css style name. For example if developers want to change GWT or other css names which are found through web debugger tool, they can add what they want in this css file.

```
.cdd-custom-ToolButton {
background-image: none; outline-width: 0px; position: relative; border: 0px; border-radius: 0px;
text-align: center; height: 32px; width: 32px; padding: 0px;
}
.cdd-custom-ToolBackground, .cdd-custom-ToolExpandable {
border: 1px solid transparent; height: 30px; width: 30px; margin: 1px;
}
.gwt-ToggleButton-up-hovering .cdd-custom-ToolBackground:hover,
.gwt-ToggleButton-down-hovering .cdd-custom-ToolBackground:hover,
.gwt-PushButton-up-hovering .cdd-custom-ToolBackground:hover,
.gwt-PushButton-down-hovering .cdd-custom-ToolBackground:hover,
.gwt-ToggleButton-up-hovering .cdd-custom-ToolExpandable:hover,
.gwt-ToggleButton-down-hovering .cdd-custom-ToolExpandable:hover,
.gwt-PushButton-up-hovering .cdd-custom-ToolExpandable:hover,
.gwt-PushButton-down-hovering .cdd-custom-ToolExpandable:hover {
border-color: black;
}
.cdd-custom-ToolExpandable {
position: absolute; left: 0px;
}
.cdd-custom-ToolIcon {
height: 24px; width: 24px; position: absolute; margin-top: 4px; margin-left: 4px; pointer-
events: none;
}
.cdd-custom-ToolBar {
background-color: #EDEDED;
}
```

3.1.2 Features

Following two lists are used to define the enabled or disabled features in the configuration.

- Configuration.features.enabled
- Configuration.features.disabled

The following features can be enabled or disabled in the configuration.

- HydrogenCalculation: Controls whether the hydrogen calculation should be performed when loading or modifying a chemical structure or not.
- ChemTranslation: Controls whether the API calls for translating CDXML to other formats, such as SMILES, MOL, etc., should be enabled or not.
- CopyPaste: Controls whether the copy & paste tools should be displayed or not, and controls whether the hotkeys should be enabled or not.
- GroupUngroup: Controls whether the group/ungroup tools should be displayed and the hotkeys are all disabled.
- FileDrop: Controls whether the CDXML files can be loaded to the ChemDraw Direct by dragging and dropping a file on the canvas.
- LogoLink: Controls whether a hyperlink should be added on the logo which shows on the right bottom of the drawing tool. Clicking the hyperlink will open PerkinElmerâ€™s web site.

- WarningsForQueries: Controls whether the warnings for queries should be shown.
- WarningsForLonePairDivalentConversion: Controls whether the warnings for long pair divalent conversion should be shown.
- AutoUpdateStructureName: Controls whether the generated name should be updated automatically once the linked structure is modified.
- ContractExpand: Controls whether the expand/contract tools should be displayed and the features are enabled.
- ExtendedCopyPaste: Controls whether the clipboard extension should be loaded or not, and controls whether the features should be enabled or not.
- WarningsForDataLoss: Controls whether the warnings for queries should be shown.
- MessageCenter: Controls whether the information, warnings, errors icon and details should be shown on the left bottom or not.
- Arrows: Controls whether the arrows group tool should be displayed or not.

The tools can also be controlled by adding the name of the tool into the enabled or disabled feature list in the configuration. Once a tool is disabled in the configuration, it will not be visible to the end users.

3.1.3 Tools Orientation

The orientation of the tools can be set by Configuration.layout.orientation.
â€ƒ
Orientation: Horizontal
Orientation: Vertical
Orientation: Mixed

3.1.4 Tool Group and Shrink

When the container of the attached drawing tool is not large enough to show the entire toolbar, the tool groups shrink.
Following are the three tool groups already supported by ChemDraw Direct.

- Symbols: CirclePlus, CircleMinus, LonePair, Radical
- Bonds: SolidBond, BoldBond, DoubleBond, TripleBond, HashedWedgeBond, WedgeBond, DashedBond
- Rings: Propane, Butane, Pentane, Hexane, Benzene, CyclohexaneChairRight, CyclohexaneChairLeft
- Arrows: SolidArrow, DashedArrow, NoGoArrow, ResonanceArrow, HollowArrow

When the tools are not defined in the groups, shrinking of these tools is disabled.

3.1.5 ChemDraw Web Service

The address of the ChemDraw Web Service can be set by Configuration.properties.chemservice .

3.2 Loading Customized Configuration

The default configuration is stored in the library. It can be modified to be the customized configuration, if developers have write access to that file. Then all web pages using the same library can load this customized configuration.
Customizing the central default configuration is not applicable to for the host application that loads the scripts from the CloudFront service that hosted by PerkinElmer.

3.2.1 Loading Customized Configuration in attach()

The section 2.3 Attaching ChemDraw Direct to the DOM Tree described how to attach a ChemDraw drawing tool to DOM. Two options can be used to specify the configuration to load when attaching.
Using the config option
var configJson = { features: {enabled: [], disabled: []}, layout: {orientation: “â€¦”, tools: []}, properties: {chemservice: “â€¦”}};
perkinelmer.ChemdrawWebManager.attach({id: “cddContainer”, config: configJson);
Using the configUrl option
var configUrl = “https://fakeurl/sample.conf”;
perkinelmer.ChemdrawWebManager.attach({id: “cddContainer”, configUrl: configUrl);
The configuration loaded in the attach() can only be applied to the drawing tool that being attached. The other drawing tools on the same web page are not affected.

3.2.2 Loading Customized Configuration in chemdrawweb_config

The ChemdrawWebManager will try to read the variable window.chemdrawweb_config to see if it is set before attaching the drawing tool to the DOM. If valid value exists, the drawing tool will apply the configuration to all the drawing tools that attached to the current page.
A JavaScript file can be written to include the script below.
// e.g. Save the script below as conf/configuration.js
window.chemdrawweb_config = {
```
features: {enabled: [â€¦], disabled: [â€¦]}, layout: {orientation: "â€¦", tools: [â€¦]},
properties: {chemservice: "â€¦"}};
```
Add the script tag in the head of the HTML pages.
```
<body></body>
```
Using the variable chemdrawweb_config developers do not need to specify the config options when attaching a ChemDraw drawing tool to the DOM. It will be helpful when there are several places in the host web page calling the attach() function.

3.2.3 Loading Customized Configuration in chemdrawweb_configUrl

The ChemdrawWebManager will try to read the variable window.chemdrawweb_configUrl (after the window.chemdrawweb_config) to see if it is set before attaching the drawing tool to the DOM. If valid value exists, the drawing tool will apply the configuration to all the drawing tools that attached to the current page.

3.3 Generate Configuration using the Sample Page

The sample page provides a customize tool which allows user to try different configurations on the current drawing tool. It also allows user to generate the configuration text in JSON, so that the content can be copied and written to the configuration file of the library.
The dialog of customizing ChemDraw Direct shows after the Customize button is clicked.
Clicking the Apply button will make the current ChemDraw load the customized configuration immediately. The modified values will not be kept after the dialog hides.
Clicking the Generate Configuration button, the configuration text in JSON will be displayed on another popup dialog which enables user to copy the whole content of the configuration.

4 Using the API

ChemDraw Direct publishes several API functions, which lets developers interact with ChemDraw programmatically. All API calls are made against the ChemDraw Direct Object. The API functions are detailed below.

4.1 API Functions

4.1.1 getInChI(callback)

This function translates the current drawing into an InChI representation. This call is asynchronous and a callback function must be provided that is informed when the results are available. A second parameter is also provided in the callback function, which is a String that describes errors that occurred during the translation.
Example:

```
function chemdrawLoaded() { cdw = new perkinelmer.ChemdrawWeb(); cdw.attach("cdwAnchor"); }
function showInChI() { cdw.getInChI(function(inchiStr, error) { if(error) {alert("Error during translation: " + error);} else {alert("The InChI String is: " + inchiStr);} }); }
```

4.1.2 getInChIKey(callback)

This function is identical to the other translation methods except that it returns an InChIKey representation of the current drawing.

4.1.3 getSMILES(callback)

This function is identical to the other translation methods except that it returns a SMILES representation of the current drawing.

4.1.4 getRXN(callback)

This function is identical to the other translation methods except that it returns a RXN representation of the current drawing.

4.1.5 getMOL(callback)

This function is identical to the other translation methods except that it returns a MOL representation of the current drawing.

4.1.6 loadCDXML(cdxml)

This function reads a CDXML string and loads its contents into ChemDraw.

4.1.7 getCDXML()

This function returns a CDXML string representing the current drawing.

4.1.8 clear()

This function clears the current contents of ChemDraw.
This operation is not undoable. And ChemDraw will not prompt the user to confirm the operation.

4.1.9 getImgUrl()

This function returns a URL string which points to a PNG representation of the current drawing. The image contents are encoded in the URL so it may be used offline.
Example:

```
function chemdrawLoaded() { cdw = new perkinelmer.ChemDrawWeb(); cdw.attach("cdwAnchor"); }
function showDrawingImage() { var url = cdw.getImgUrl(); var newWindow = window.open("");
newWindow.document.write("<img src=â€™" + url + "â€™/>"); }
```

4.1.10 getVersion()

This function returns the current version of ChemDraw as a string.

4.1.11 isBlankStructure()

This function returns true if nothing has been drawn in ChemDraw and false if something has been drawn.

4.1.12 loadConfig(configObject)

This function reads a configuration object in JSON and loads the new configuration into ChemDraw.

4.1.13 loadConfigFromUrl(url)

This function reads a remote configuration file and loads the new configuration to ChemDraw.

4.1.14 setViewOnly(viewonly)

This function sets whether the user is able to interact with ChemDraw.

4.1.15 fitToContainer()

The tool automatically resizes to fit its container when the window resizes, but if its container resizes under other circumstances, this function must be called manually to resize ChemDraw to the new container size.

4.1.16 getErrors()

This function returns an array of strings describing the chemical warnings that are present in the current drawing.

4.1.17 labelReactions()

Deprecated, use labelFragments(IDs) instead.
Calling this function will add Roman numeral labels to any reactions present in the current drawing. Calling it with labels already present will remove the existing labels and reapply new ones.

4.1.18 labelFragments(IDs)

This API is added in CDD 1.7 to replace labelReactions(). It needs an int array of IDs to label every fragment on canvas. ID refers to “id” attribute of fragment node in CDXML.
Example:

labelFragments([30, 31, 32]);

4.1.19 addReactant(cdxml)

This function appends a new reactant to the reaction present in the current drawing. If there are reactants present, the new reactant is added to the left of the left-most reactant. If no reaction is present, a new arrow is added to the center of the canvas along with the reactant.

4.1.20 addProduct(cdxml)

This function appends a new product to the reaction present in the current drawing. If there are products present, the new product will be added to the right of the right-most product. If no reaction is present, a new arrow will be added to the center of the canvas along with the product.

4.1.21 addReagent(cdxml)

This function appends a new reagent to the reaction present in the current drawing. The reagent is added to the top of the reaction arrow and above the top-most reagent if there are any already. If no reaction is present, a new arrow is added to the center of the canvas, along with the reagent.

4.1.22 getProperties(callback)

This API makes an AJAX call to service, and need a callback as the parameter. The callback function is same with APIs getInChI, getSMIKES, getRXN or getMOL.
The first argument in the callback function is a string of JSON format containing some properties of structures drawn on canvas.
The returned JSON string is something similar to the string shown below (following is formatted, the actual return value is a one-line string)

```
{
“NAME” : “benzene”,
“FORMULA” : “C6H6”,
“MW” : “78.1140000006 g/mol”,
“ID” : “24”,
“INCHI” : “InChI=1S/C6H6/c1-2-4-6-5-3-1/h1-6H”,
“SMILES” : “C1=CC=CC=C1”
}
```

From top to bottom, the fields mean structure name, formula (html format, having tags like <sub>, <sup>), molecule weight, Fragment ID (same with id attribute in CDXML), InChI, SMILES.

4.1.23 markAsSaved(), isSaved()

These two APIs need to work together.
markAsSaved() does not return anything. By invoking this API, it tells CDD that the current drawings are saved.
isSaved() returns a boolean value. True, if the current drawings are not modified since last call to markAsSaved(); false, if changes are made after markAsSaved() is called.

4.1.24 getFragmentsInfo()

It returns a JavaScript array containing information of the fragments in canvas. Only the fragments having chemical meanings (structures, formulas, etc.) are included in the result. Captions (literal text), reaction arrow, brackets, etc. are not include.

Each entry in the array containing the bellow properties:

- cdxml: a string, the CDXML string of this fragment.
- image: a string, the image in base64 string of this fragment.
- width: a number, the width of this fragment.
- height: a number, the height of this fragment.
- x: a number, the x position (left-top point of this fragment).
- y: a number, the y position (left-top point of this fragment).

***Note**: x, y, width, height are measured in CDDâ€™s coordinate system, which is not always same with the HTML position or size.*

4.1.25 nameToStructure(name, callback)

It reads a structure name and returns a CDXML string representing the structure. This API in CDD 1.6 and later will not return result, the result will be returned in callback which as the second parameter when invoke this API.
Note: This function does not read the selected text from the drawing tool. And the generated CDXML will not be rendered on the canvas.

4.1.26 structureToName(structure, callback)

It reads a CDXML string and returns the name of the structure. This API in CDD 1.6 and later wonâ€™t return result, the result will be returned in callback which as the second parameter when invoke this API.
Note: This function does not read the CDXML from the drawing tool. And the generated name will not be pasted to the canvas.

4.1.27 detectDataLoss()

When loading or pasting a CDXML string into CDD, or dragging & dropping a CDXML file into CDD, CDD will detect unsupported objects in the CDXML. This function returns an array of strings describing the detailed information that may cause data loss, e.g.
[
“This CDXML is created by program: ChemDraw 14.0.1.9, which is not fully supported.”,
“NMR Predictions are not supported yet.”
]
These unsupported objects are added to the canvas, but they are invisible. After calling the API clear(), these objects will be removed together with other objects.
If no objects are unsupported by CDD, this function returns an empty array.

4.1.28 getSVG(callback)

This API is used to retrieve the Scalable Vector Graphics (SVG) representation of current drawing. The SVG string is passed as the first parameter of the callback. If any error happens, the error string is passed as the second parameter of the callback.

4.1.29 getEPS(callback)

This API is used to retrieve the Encapsulated PostScript (EPS) representation of current drawing. The EPS string is passed as the first parameter of the callback. If any error happens, the error string is passed as the second parameter of the callback.

4.1.30 getWMF(callback)

This API is used to retrieve the Windows Metafile (WMF) representation of current drawing. The WMF blob is passed as the first parameter of the callback. If any error happens, the error string is passed as the second parameter of the callback.

4.1.31 getEMF(callback)

This API is used to retrieve the Enhanced Metafile (EMF) representation of current drawing. The EMF blob is passed as the first parameter of the callback. If any error happens, the error string is passed as the second parameter of the callback.

4.1.32 loadMOL(mol, callback)

This API is used to load a MOL string to the canvas. The callback is called when the loading is completed. If any error happens, the error string is passed as the second parameter of the callback.

4.1.33 loadSMILES(smiles, callback)

This API is used to load a SMILES string to the canvas. The callback is called when the loading is completed. If any error happens, the error string is passed as the second parameter of the callback.

4.1.34 loadB64CDX(base64cdx, callback)

This API is used to load a base64 encoded CDX string to the canvas. The callback is called when the loading is completed. If any error happens, the error string is passed as the second parameter of the callback.

4.1.35 shrinkToFit()

This API can adjust the zoom level and viewport position to make all the drawing objects visible and centered on the canvas. If the canvas is not large enough to show all drawings, this API zooms out the drawings, and the drawings become smaller and centered. If the canvas is large enough to show all drawings, this API only centers the drawings.

4.1.36 setContentChangedHandler(callback)