Modern i2b2 Plugin Development

Plugin Development for i2b2 Release v1.8+
11st October 2022

PDF Version of this document: https://doi.org/10.6084/m9.figshare.21507267

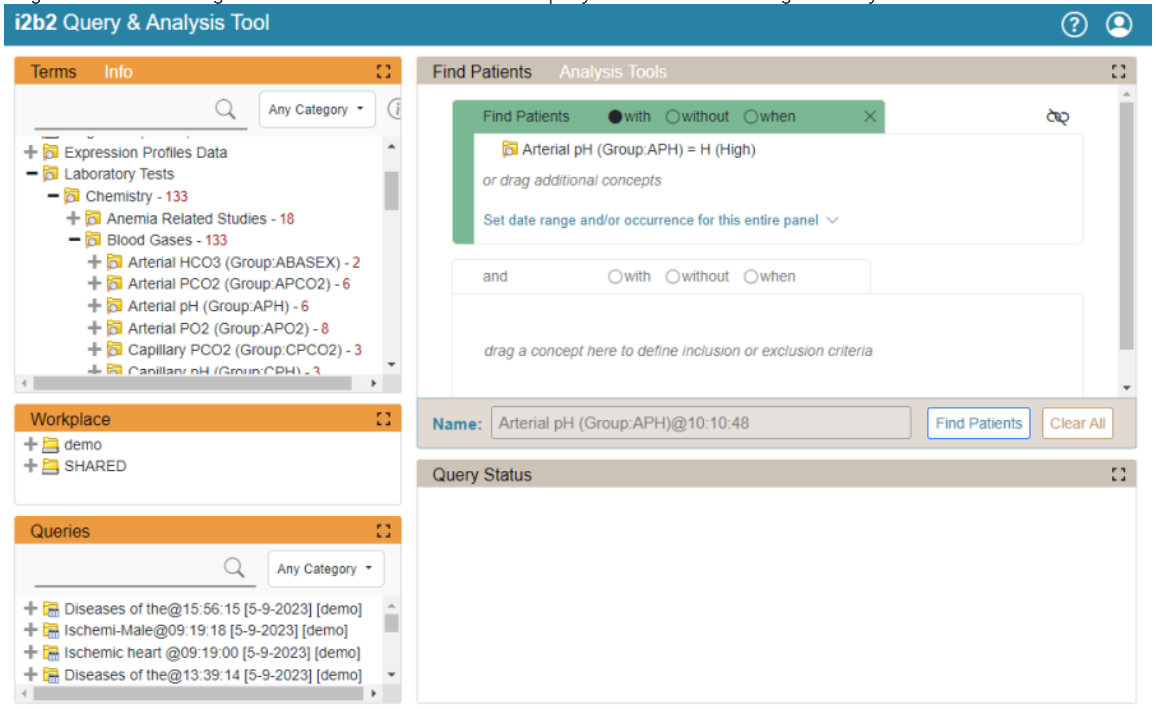
SUMMARY

This document describes the design of the Plugin Manager component and plugin support libraries that interact with the new "modern" style i2b2 plugins. This new plugin manager does not handle or interact with "legacy plugins" which are handled by the LEGACYPLUGIN manager.

INTRODUCTION TO I2B2

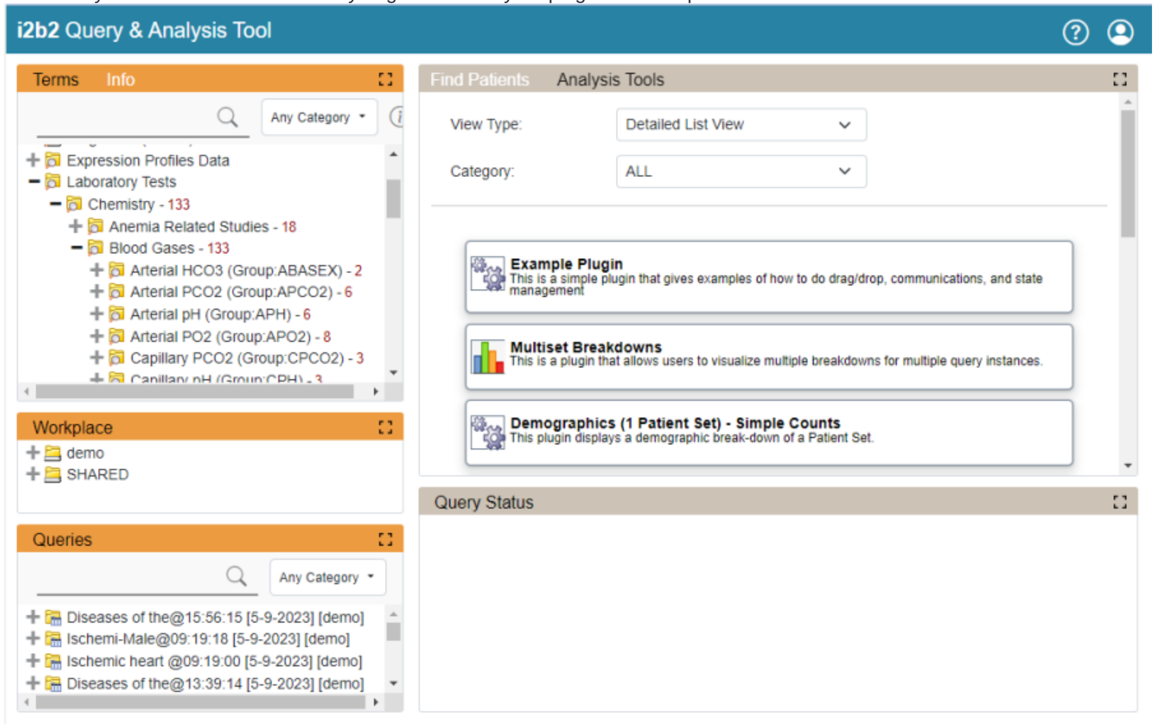
Usage

The Informatics for Integrating Biology & the Bedside ("i2b2") project is an easy to use data query and retrieval platform that is used by researchers to perform searches on electronic health records. This is done using a simple drag-and-drop interaction whereas users browse (or search) a list of medical diagnoses and then drag those terms into various areas of a query builder window. The general layout is shown below:



i2b2 Query & Analysis Tool
Terms
Info
Any Category
Expression Profiles Data
Laboratory Tests
Chemistry - 133
Anemia Related Studies - 18
Blood Gases - 133
Arterial HCO3 (Group:ABASEX) - 2
Arterial PCO2 (Group:APCO2) - 6
Arterial pH (Group:APH) - 6
Arterial PO2 (Group:APO2) - 8
Capillary PCO2 (Group:CPCO2) - 3
Workplace
demo
SHARED
Queries
Any Category
Diseases of the@15:56:15 [5-9-2023] [demo]
Ischemi-Male@09:19:18 [5-9-2023] [demo]
Ischemic heart @09:19:00 [5-9-2023] [demo]
Diseases of the@13:39:14 [5-9-2023] [demo]
Find Patients
Analysis Tools
Find Patients ● with ○ without ○ when
Arterial pH (Group:APH) = H (High)
or drag additional concepts
Set date range and/or occurrence for this entire panel
and ○ with ○ without ○ when
drag a concept here to define inclusion or exclusion criteria
Name: Arterial pH (Group:APH)@10:10:48
Find Patients
Clear All
Query Status

On the upper left is the "Terms" window which contains search terms. On the upper right is the "Find Patients" window which is used to build the query. Also on the upper right is the "Analysis Tools" window which displays a list of plugins (both new and legacy) that a user can access by clicking on the title. This "Analysis Tools" window is where you go to access your plugin in development.



i2b2 Query & Analysis Tool
Terms
Info
Any Category
Expression Profiles Data
Laboratory Tests
Chemistry - 133
Anemia Related Studies - 18
Blood Gases - 133
Arterial HCO3 (Group:ABASEX) - 2
Arterial PCO2 (Group:APCO2) - 6
Arterial pH (Group:APH) - 6
Arterial PO2 (Group:APO2) - 8
Capillary PCO2 (Group:CPCO2) - 3
Workplace
demo
SHARED
Queries
Any Category
Diseases of the@15:56:15 [5-9-2023] [demo]
Ischemi-Male@09:19:18 [5-9-2023] [demo]
Ischemic heart @09:19:00 [5-9-2023] [demo]
Diseases of the@13:39:14 [5-9-2023] [demo]
Find Patients
Analysis Tools
Query Status

View Type: Detailed List View

Category: ALL

Example Plugin
This is a simple plugin that gives examples of how to do drag/drop, communications, and state management

Multiset Breakdowns
This is a plugin that allows users to visualize multiple breakdowns for multiple query instances.

Demographics (1 Patient Set) - Simple Counts
This plugin displays a demographic break-down of a Patient Set.

System Design

In general, i2b2 is designed to operate using a services-oriented architecture. An i2b2 "hive" is a collection of services called "cells". The primary i2b2 services are: *PM / Project Management* which manages projects/users in the hive, *ONT / Ontology* which stores all the terms that are used to select patients, *CRC / Clinical Research Chart* which is the central repository of patient data within the hive, *WORK / Workplace* which allows users to store information for later use. The i2b2 web client segments the functionality of its libraries in a similar way. In addition the core of the v1.8 web client has the PLUGIN and LEGACYPLUGIN components which handle the web client's plugins.

PLUGIN OVERVIEW

Plugins

All "modern" style plugins (release version 1.8 and later) will be loaded and operate inside of their own HTML iframe within the larger i2b2 web client. They will be allowed to use any and all Javascript libraries that they wish (ex. jQuery, D3js, React, Angular, Vue, etc). The only thing needed to be integrated into the main application is for the plugin's HTML to reference an i2b2-loader.js file which is used to load several plugin-specific support libraries that allows for drag-drop operations, AJAX communication with the i2b2 servers, state management and privileged access to the main UI's functions and variables.

i2b2-loader.js

This file loads and waits for the plugin's HTML file to fully load. On load completion, it uses the Window.postMessage function to send an initialization message to the owner of the iframe (the main i2b2 web application) which then replies with a list of file locations for the rest of the plugin-side i2b2 framework script files. In addition to the file locations it also sends configuration information used to configure the plugin and its injected support libraries into the plugin's global namespace.

Injected i2b2 Support Libraries

The plugin-side support libraries are hosted in the /js-i2b2/cells/PLUGIN/libs directory and are injected into your plugin by the i2b2-loader.js file:
i2b2-sdx.js - This support library makes up the "Standard Data eXchange" (SDX) layer that enables drag drop handling of i2b2 objects.
i2b2-ajax.js - This support library will populate the i2b2.ajax namespace within your plugin with function calls that map to the AJAX calls of all the cells in the web client's main UI.
i2b2-state.js - This support library manages the i2b2.model and i2b2.state namespaces within your plugin. It saves the state of this namespace throughout subsequent reloads of your plugin that happens when the user rearranges UI tabs.
i2b2-auth-tunnel.js - This support library builds the i2b2.authorizedTunnel namespace which gives you the ability to access variables and functions that are within the main UI window.

PLUGIN MANAGER

File Locations

The plugin manager for plugins exists within the /js-i2b2/cells/PLUGIN directory. The support libraries that are loaded into the plugins themselves will be hosted from the /js-i2b2/cells/PLUGIN/libs directory.

Operation

The plugin manager will operate as a singleton cell within the larger application. However, the plugin manager will have the ability to have multiple view instances (and thus plugin iframes). This is to allow multiple plugins to be loaded and operate simultaneously without interference.
The plugin manager communicates back and forth to your plugin's support libraries using the browser's window.postMessage() function call. The support libraries hide the complexity of this communication modality for you.

PLUGINS

File Locations

All plugins will exist entirely within the /plugins directory. Each plugin will be hosted in a single directory that follows Java-style library paths such as/edu/harvard/catalyst/example with the resulting plugin name being defined as "edu.harvard.catalyst.example". All files and libraries needed by the plugin will be contained solely within its own directory. This also includes a copy of the i2b2-loader.js script file. There will also be a file called plugins.json that exists within the root /plugins directory containing an array of plugin names as shown below.

```
[
"edu.harvard.catalyst.example",
"edu.harvard.WeberLab.ExportPatientset"
]
```

If you are using the i2b2-webclient-proxy service to host your web client UI it will automatically generate the plugins.json file based on the directory structure if the file does not exist. The Github repository for this proxy server is at _https://github.com/hms-dbmi/i2b2-webclient-proxy_

Required Loader Script

All plugins that interact with the main i2b2 UI must load the support libraries hosted in the /js-i2b2/cells/PLUGIN/libs directory. The easiest way to do this is to incorporate the i2b2-loader.js file. The loader script also handles initialization of the i2b2 namespace in your plugin's code environment.

Required Configuration File

All plugins will have a configuration file named plugin.json within the root of the plugin's main directory. This file should at minimum have the following data:

title	A short title for your plugin
description	A description for your plugin
icons.size32x32	The filename of the icon graphic (that exists within the assets directory)
assetDir	The name of the assets directory for your plugin
base	The filename of your plugin's main HTML page

Here is an example file:

```
{
"title": "Example Plugin",
"description": "This is a simple plugin...",
"category"

["example"],
"icons": { "size32x32": "exampleplugin_32x32.gif" },
"assetDir": "assets",
"plugin_version" : "1.0",
"base": "index.html"
}
```

If you need to have access to variables or to execute function calls that exist within the main i2b2 UI then you would want to configure an authorizedTunnel variable. More details are in the section describing the Authorized Tunnel functionality.

Your Plugin's i2b2 Namespace

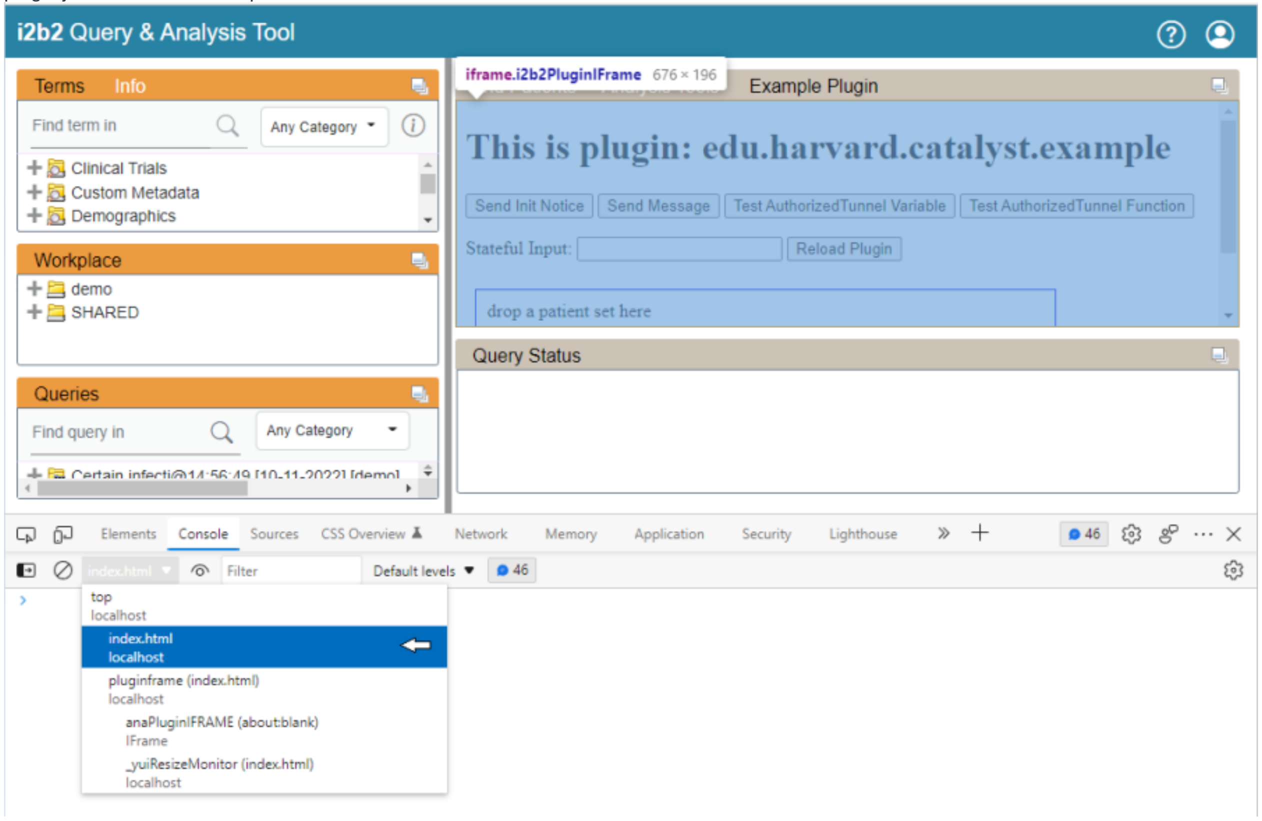
Before your plugin is activated by an "I2B2_READY" window event, the i2b2-loader.js file injects several support files into your plugin. In doing this it builds the i2b2 namespace as follows:

```
i2b2.
MSG_TYPES: {AJAX: {…}, STATE: {…}, SDX: {…}, TUNNEL: {…}}
ajax: {PM: {…}, ONT: {…}, CRC: {…}, WORK: {…}}
authorizedTunnel: {authorizedList: {…}, function: …, variable: …}
h: {initPlugin: ƒ, getScript: ƒ}
model: {}
state: {save: ƒ}
sdx: {AttachType: ƒ, setHandlerCustom: ƒ}
```

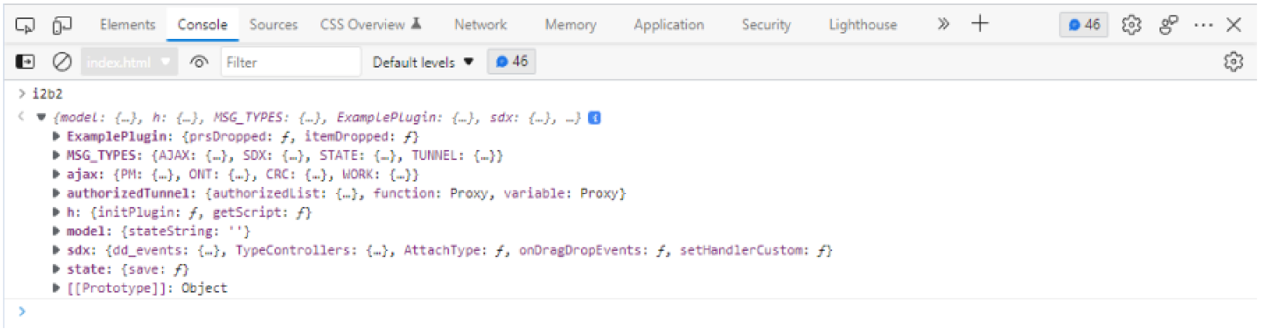
i2b2.MSG_TYPES	Contains the "magic strings" used to communicate with the plugin manager in the main UI via Window.postMessage().
i2b2.ajax	Contains Promise-returning functions that return a string of XML which was returned by the i2b2 server for your particular API call.
i2b2. authorizedTunnel	Contains the "function[]" and "variable[]" accessors your plugin can use to programmatically access resources within the main UI.
i2b2.h	Some helper functions used by the i2b2 loader file.
i2b2.model	This is where your plugin should store its data that it wants preserved when the plugin reloads. Use i2b2.state.save() function to save it.
i2b2.state	Contains the save() function used to save the i2b2.model namespace.
i2b2.sdx	Contains two functions used to make HTML elements drop targets for the i2b2 SDX subsystem.

Accessing Your Plugin's i2b2 Environment

To access the code environment of your i2b2 plugin you should load your plugin from the "Analysis Tools" tab on the right side of the i2b2 web client. After your plugin is loaded and displayed, you should open your browser's debug tools and go to the "Console" tab. Once in the console window you should see a dropdown menu on the left hand side of the screen. Use this dropdown to select your plugin's HTML file that was defined using the "base" variable in the plugin.json file. In this example it is called "index.html".



Once the proper file is selected in the drop down menu you will be able to view the i2b2 namespace and access your plugin's code from the console as shown below.



```
> i2b2
< ▼ {model: {…}, h: {…}, MSG_TYPES: {…}, ExamplePlugin: {…}, sdx: {…}, …}
    ▶ ExamplePlugin: {prsDropped: ƒ, itemDropped: ƒ}
    ▶ MSG_TYPES: {AJAX: {…}, SDX: {…}, STATE: {…}, TUNNEL: {…}}
    ▶ ajax: {PM: {…}, ONT: {…}, CRC: {…}, WORK: {…}}
    ▶ authorizedTunnel: {authorizedList: {…}, function: Proxy, variable: Proxy}
    ▶ h: {initPlugin: ƒ, getScript: ƒ}
    ▶ model: {stateString: ''}
    ▶ sdx: {dd_events: {…}, TypeControllers: {…}, AttachType: ƒ, onDragDropEvents: ƒ, setHandlerCustom: ƒ}
    ▶ state: {save: ƒ}
    ▶ [[Prototype]]: Object
>
```

Initialization of Your Plugin

Your plugin should wait until the loading and support file injection process is finished before attempting to access anything in the i2b2 namespace. This can be done safely by waiting for a window event called "I2B2_READY". This event occurs after the DOM is ready and after all i2b2 support libraries are also loaded. Some example code follows:

```
window.addEventListener("I2B2_READY", ()=> {
// i2b2 framework is loaded and ready (including population of i2b2.model)
// populate the UI based on a previously saved state (if it exists, defaults to "")
if (i2b2.model.stateString === undefined) i2b2.model.stateString = "";
document.getElementById("stateString").value = i2b2.model.stateString;
});
```

During the initialization of the i2b2 framework within your plugin, you can also monitor or react to the initialization of each plugin support library. For each support library, there is an initial configuration event that is sent by the i2b2 loader process to the support library followed by an event signaling that the support library is initialized. The initialization event may contain a data payload for the support library. Both of these custom events are issued via window.dispatchEvent() and can be captured using window. addEventListener().

Support Library	***Initialization Event***	***Initialized Event***
SDX (Drag and Drop)	"I2B2_INIT_SDX"	"I2B2_SDX_READY"
AJAX Communication	"I2B2_INIT_AJAX"	"I2B2_AJAX_READY"
State Management	"I2B2_INIT_STATE"	"I2B2_STATE_READY"
Authorized Tunnel	"I2B2_INIT_TUNNEL"	"I2B2_TUNNEL_READY"

PLUGIN SUPPORT LIBRARIES

Drag and Drop (SDX)

The Standard Data eXchange layer ("SDX") lives in the i2b2-sdx.js file. When this support library is fully loaded it emits the event "I2B2_SDX_READY". The new i2b2 web client has changed its drag and drop operations to use native browsers' Drag and Drop API functions. All of the complexity of the drag and drop operations has been abstracted away by this library. To simplify adoption, new plugins can use basically the same code as the older plugin style (pre-v1.8) to register a DOM element for drag drop operations. To accept a drag drop you need to: 1) register the DOM element for drops of specific SDX types, 2) register a handler function for the drop operation of that specific SDX type.

```
window.addEventListener("I2B2_SDX_READY", (event) => {
i2b2.sdx.AttachType("ExamplePlugin-psmaindiv", "PRS"); // Patient Record Set
i2b2.sdx.AttachType("ExamplePlugin-maindiv", "CONCPT"); // Ontology Concept
i2b2.sdx.AttachType("ExamplePlugin-maindiv", "PRS"); // Patient Record Set
i2b2.sdx.AttachType("ExamplePlugin-maindiv", "ENS"); // Encounter Set
i2b2.sdx.AttachType("ExamplePlugin-maindiv", "PRC"); // Patient Record Count
i2b2.sdx.AttachType("ExamplePlugin-maindiv", "QDEF"); // Query Definition
i2b2.sdx.AttachType("ExamplePlugin-maindiv", "QGDEF"); // Query Group Definition
i2b2.sdx.AttachType("ExamplePlugin-maindiv", "QI"); // Query Instance
i2b2.sdx.AttachType("ExamplePlugin-maindiv", "QM"); // Query Master
i2b2.sdx.AttachType("ExamplePlugin-maindiv", "WRK"); // Workspace Entry
// drop event handlers used by this plugin
i2b2.sdx.setHandlerCustom("ExamplePlugin-psmaindiv", "PRS", "DropHandler",
i2b2.ExamplePlugin.prsDropped);
i2b2.sdx.setHandlerCustom("ExamplePlugin-maindiv", "CONCPT", "DropHandler",
i2b2.ExamplePlugin.itemDropped);
i2b2.sdx.setHandlerCustom("ExamplePlugin-maindiv", "PRS", "DropHandler",
i2b2.ExamplePlugin.itemDropped);
i2b2.sdx.setHandlerCustom("ExamplePlugin-maindiv", "ENS", "DropHandler",
i2b2.ExamplePlugin.itemDropped);
i2b2.sdx.setHandlerCustom("ExamplePlugin-maindiv", "PRC", "DropHandler",
i2b2.ExamplePlugin.itemDropped);
i2b2.sdx.setHandlerCustom("ExamplePlugin-maindiv", "QDEF", "DropHandler",
i2b2.ExamplePlugin.itemDropped);
i2b2.sdx.setHandlerCustom("ExamplePlugin-maindiv", "QGDEF", "DropHandler",
i2b2.ExamplePlugin.itemDropped);
i2b2.sdx.setHandlerCustom("ExamplePlugin-maindiv", "QI", "DropHandler",
i2b2.ExamplePlugin.itemDropped);
i2b2.sdx.setHandlerCustom("ExamplePlugin-maindiv", "QM", "DropHandler",
i2b2.ExamplePlugin.itemDropped);
i2b2.sdx.setHandlerCustom("ExamplePlugin-maindiv", "WRK", "DropHandler",
i2b2.ExamplePlugin.itemDropped);
i2b2.sdx.setHandlerCustom("ExamplePlugin-maindiv", "XML", "DropHandler",
i2b2.ExamplePlugin.itemDropped);
});
```

With the registered DropHandler function receiving a standard SDX data packet as shown below:

```
i2b2.ExamplePlugin.itemDropped = function(sdxData) {
console.dir("example plugin received item drop " + JSON.stringify(sdxData));
let mainDiv = document.getElementsByClassName("maindiv-content")[0];
mainDiv.innerHTML= JSON.stringify(sdxData);
};
```

State Management

This functionality lives in the i2b2-state.js file. When this support library is fully loaded it emits the event "I2B2_STATE_READY". The new i2b2 web client uses HTML iframes to fully isolate plugin code from the main UI's environment. This allows plugins to use any libraries they want without having code conflicts with the main UI or other plugins. Unfortunately, browser standards dictate that when an iframe is manipulated within the DOM tree its contents is destroyed and reloaded. This results in destruction of the plugin's internal state. This can occur when a user drags and reorganizes the tabs in the main UI, or if the user expands the plugin to full size, or pops the plugin out into a floating window. To solve this problem, the i2b2 plugin framework can save the i2b2.model namespace to main UI memory. Also, upon loading of the state functionality library, any previous state is retrieved and automatically loaded into the i2b2.model namespace. In order to save the i2b2.model namespace you will need to call the i2b2.state.save() function whenever you update it as shown below:

```
function saveState() {
i2b2.model.stateString = document.getElementById("stateString").value;
i2b2.state.save();
}
```

To properly implement state management, you should drive your plugin from the data contained in the i2b2.model namespace. Upon loading your plugin, you should render the screen based on this persistent storage location.

```
window.addEventListener("I2B2_READY", ()=> {
// i2b2 is loaded and ready (including population of i2b2.model namespace)
if (i2b2.model.stateString === undefined) i2b2.model.stateString = "";
// populate the UI based on a previously saved state
document.getElementById("stateString").value = i2b2.model.stateString;
});
```

AJAX Calls to the i2b2 Server

This functionality lives in the i2b2-ajax.js file. When this support library is fully loaded it emits the event "I2B2_AJAX_READY". This library will populate the i2b2.ajax namespace with the loaded cells along with the AJAX calls that can be made to access server-side functionality. These function calls accept the same inputs as their original i2b2 function calls do. However, these functions will return a Javascript Promise that will resolve or reject and can be further processed using a .then() or .catch() as shown below:

```
function sendTestMsg() {
console.log("sending test message...");
let ajaxData = {
ont_synonym_records: "N",
ont_hidden_records: "N"
};
i2b2.ajax.ONT.GetCategories(ajaxData).then((data)=> {
console.log("Got response base from ONT.GetCategories()");
console.log(data);
}).catch((error) => {
console.error(error);
});
}
```

There is also a function called _RawSend() which is added to each cell's AJAX namespace and enables you to send a raw XML message to the cell as a string. The format you use to pass the information is:

```
let ResultPromise = i2b2.ajax.ONT._RawSend(serviceURL, rawMessage);
```

Where serviceURL is the path to the service endpoint when it is combined with the cell's base path. The variable rawMessage is a string containing the full XML message that is passed to the server. The _RawSend function will automatically populate the following template tokens:

`{proxy_info}`	Proxy redirection information.
`{sec_user}`	User that is currently logged in.
`{sec_pass_node}`	The password/session credential used to authenticate.
`{sec_domain}`	The domain that the user is logged into.
`{sec_project}`	The project that the user is logged into.
`{header_msg_id}`	A unique identifier for the message.
`{header_msg_datetime}`	The date and time that the message was set at.
`{result_wait_time}`	How long the request should stay open (defaults to 180 seconds).

The **Ontology** cell found at i2b2.ajax.ONT contains the following AJAX calls:
GetCategories(), GetChildConcepts(), GetChildModifiers(), GetCodeInfo(), GetModifierCodeInfo(), GetModifierInfo(), GetModifierNameInfo(), GetModifiers(), GetNameInfo(), GetSchemes(), GetTermInfo() and _RawSent().

The **CRC** cell found at i2b2.ajax.CRC contains the following AJAX calls: deleteQueryMaster(), getIbservationfact_byPrimaryKey(), getNameInfo(), getPDO_fromInputList(), getQRY_getResultType(), getQueryInstanceList_fromQueryMasterId(), getQueryMasterList_fromUserId(), getQueryResultInstanceList_fromQueryInstanceId(), getQueryResultInstanceList_fromQueryResultInstanceId(), getRequestXml_fromQueryMasterId(), renameQueryMaster(), runQueryInstance_fromQueryDefinition() and _RawSent().

The **Workplace** cell found at i2b2.ajax.WORK contains the following AJAX calls:
addChild(), annotateChild(), deleteChild(), getChildren(), getFoldersByProject(), getFoldersByUserId(), moveChild(), renameChild() and _RawSent().

The **Project Management** cell found at i2b2.ajax.PM contains the following AJAX calls:
deleteApproval(), deleteCell(), deleteDBLookup(), deleteGlobal(), deleteHive(), deleteParam(), deleteProject(), deleteRole(), deleteUser(), getAllApproval(), getAllCell(), getAllDBLookup(), getAllGlobal(), getAllHive(), getAllParam(), getAllProject(), getAllProjectRequest(), getAllRole(), getAllRoleUser(), getAllUser(), getApproval(), getCell(), getDBLookup(), getGlobal(), getParam(), getProject(), getProjectRequest(), getUser(), getUserAuth(), setApproval(), setCell(), setDBLookup(), setGlobal(), setHive(), setParam(), setPassword(), setProject(), setProjectRequest(), setRole(), setUser(), and _RawSent().

Authorized Tunnel Access

This functionality lives in the i2b2-auth-tunnel.js file. When this support library is fully loaded it emits the event "I2B2_TUNNEL_READY". The new i2b2 web client uses HTML iframes to fully isolate the plugin code environment from the code environment of the main i2b2 UI. In some situations, you may need to access a variable or execute a function that exists within the main UI. A logical tunnel was created to accomplish this which provides secure access to these resources.
In order to access a variable or function within the main UI you will first need to add a configuration to your plugin's plugin.json file as shown below:

```
{
"title": "Example Plugin",
"description": "This is a simple plugin that gives examples of how to do drag/drop, communications, and state management",
"category": ["example"],
"icons": { "size32x32": "exampleplugin_32x32.gif" },
"assetDir": "assets",
"base": "index.html",
"authorizedTunnel": {
"variables" :{
"i2b2.PM.model.isAdmin": "R",
"i2b2.ONT.cfg": "RO"
},
"functions":[

"i2b2.h.getDomain",

"i2b2.h.getProject",
"i2b2.ONT.view.nav.doRefreshAll"
]
}
}
```

Within the authorizedTunnel namespace exists an object named variables having its keys be the path used to access the variables within the main i2b2 UI code environment and its values being a string that contains the access permissions to that variable. The **access string** consists of a combination of the following 3 characters in any order: **"R"** signifying the read permission, **"W"** signifying the write permission, and **"O"** signifying that the targeted variable is an object to be read or written. If the targeted variable is found to be an object but the "O" flag is not used then a security error is thrown in the main UI and the request is ignored.
Within the authorizedTunnel namespace also exists an array named functions containing strings which are the path used to access the functions within the main i2b2 UI code environment.

To *read a variable's value* you will need to use i2b2.authorizedTunnel.variable as shown below. Notice how the accessor method uses square brackets [] to access the variable and not parentheses as in a function. Also, due to the asynchronous nature of window.postMessage() communications a Javascript Promise is returned instead of the actual value of the variable. This Promise can be rejected and an error thrown in the main UI if unauthorized /unconfigured access is attempted.

```
i2b2.authorizedTunnel.variable["i2b2.PM.model.isAdmin"].then((isReallyAnAdmin) => {
if (isReallyAnAdmin) {
alert("You ARE logged in as an Administrator");
} else {
alert("You are NOT logged in as an Administrator");
}
});
```

To **set a variable's value** you will also use i2b2.authorizedTunnel.variable as but you will simply use it on the left side of an assignment statement as shown below. If your plugin has not been configured or is otherwise not allowed write permissions to the variable your plugin will not be notified that the value was not assigned however an error will be thrown in the main UI.

```
i2b2.authorizedTunnel.variable["i2b2.PM.model.isAdmin"] = true;
```

To *access a function* you will use i2b2.authorizedTunnel.function. Once again, you will use square brackets \[\] to access the function as shown below. When accessing the function an anonymous function will be returned which accepts any number of parameters that you can send to the real function within the main UI. When this anonymous function is executed it also returns a Javascript Promise which will resolve with the value returned by the function, or it may be rejected and an error is thrown within the main UI.

```
let mainUI_function = i2b2.authorizedTunnel.function["i2b2.h.getDomain"];
let mainUI_promise = mainUI_function(my, parameters, here);
mainUI_promise.then((domain) => {
alert("The i2b2 domain is: "+domain);
});
```

This can be written more succinctly as follows:

```
i2b2.authorizedTunnel.function["i2b2.h.getDomain"](my, parameters, here).then(
(domain) => {
alert("The i2b2 domain is: "+domain);
}
);
```

i2b2 Web Client Repository

https://github.com/hms-dbmi/i2b2v2-webclient

Acknowledgments

Nick Benik, Marc-Danie Nazaire, Marc Ciriello, Anupama Maram, Perminder Singh, Griffin Weber