Use POST messages for IFrame Integration

To facilitate advanced desktop-based integrations, the MAX (My Agent eXperience) agent client supports event subscription by custom-embedded web pages. MAX has the ability to embed (IFrame) web pages using either Contact Panels (which open and close in conjunction with their associated contact) or Custom Workspaces (which are always open and available irrespective of individual contacts). A common use case for Contact Panels would be a customer-specific CRM web page being opened as a screen pop to a phone call (and then closed when the call is completed). A common use case for Custom Workspaces would be a knowledge base page or other site that are not directly associated with a contact or customer interaction.

In either case (Contact Panel or Custom Workspace), the embedded “child” web page can subscribe to system events received by the “parent” MAX window via POST messages. MAX routinely receives information from the ACD platform with details about the agent state or about individual contacts. By subscribing to these events, the IFramed web page can implement customized logic that responds to behavior in MAX. For example, as the agent changes state from available to working and from working to unavailable, the custom-embedded web page may choose to respond based on business rules. Or, as a new call is received, the web page can be notified of the new call and respond accordingly. For more information, see additional documentation at https://developer.niceincontact.com/API, https://developer.niceincontact.com/Documentation/AgentSessionEvents, and https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage.

Data Object Variables

MAX is set up to receive a post message from the customer in a data object containing the following values. All of the property keys are case sensitive. Where specified, the values are also case sensitive.

- **contactCardData**: This is the data object that contains customer name and the image of the phone contact. It has two keys, name and userImg. It is case sensitive.
- **contactId**: This is an optional int. If you know the contact ID, you can provide it here. If you don't have the contact ID and it is loaded in a contact panel, we will locate the associated panel's contact ID. General custom workspaces will not have a contact ID that can be located this way, so they will need to be provided if a contact-specific entry is desired.

- **issuer**: This is an optional string. It works as an identifier name for the sender, though it does not have to be unique. This string helps with logging to add context to console logs. In testing, you might find that your post message listener picks up the registration message that you send. You can use this field to check if the message came from you. MAX will send the response events with 'MAX' as the issuer.
- **messageType**: This is a required string. It is case sensitive. There are two valid entries:

ENTRY VALUE	RESULT
"RegisterForClientEvents"	This is how MAX knows that you are looking to set up a client event subscription.
"UnregisterFromClientEvents"	This will disconnect your window reference from MAX and end the client event sending. If you want to change the subscription types or contactId, you will need to call this before re-registering. Not doing this and using the same window with a new registration in the contactPanel will cause bugs.
"ContactCardData"	This is how MAX knows it's getting the customer name and image of the phone contact.

- **subscriptionTypes**: This is a required array of strings. It is not case sensitive, since everything in the array will be normalized to lowercase. The values in the array are additive. Each option specifies a type of message that you will receive. There are a few possible valid entries:

ENTRY VALUE	RESULT
"all"	This will return everything.
"agent"	This will return anything that doesn't have a contact ID.
"contact"	This will return events limited either by the contact ID requested or the contact ID for the associated panel if the IFrame is connected to a skill or contact.
"contacts"	This will return all events that have a contact ID. If you are in a contact panel and want additional information with the panel's contact, add this field.
"sessioninfo"	This will return an agent's session token. If you are looking to make agent session specific API calls, register for this event.

Note:

Using an empty array for **subscriptionTypes** will result in an error.

When MAX receives the subscription, along with the acknowledge message, we will send the current states of the requested subscription types to help set up the workspace appropriately. These events are guaranteed at start.

ENTRY VALUE	CURRENT STATES
"all"	Will return current **AgentState** and all current contacts that are within the scope of MAX.

ENTRY VALUE	CURRENT STATES
"agent"	Will return current **AgentState**. This does not include the agent's Session Token.
"contact"	Will return current contact state if the contact exists within the scope of MAX.
"contacts"	Will return all current contacts that exist within the scope of MAX.

It is possible that the registration will occasionally receive more than the base **AgentState** and contacts on initialization. These events could include the AgentLegEvent, AgentSessionStart, and MchAgentSettingsChangeEvent. You will see these extra events if your subscription request comes in before we have established our local references to this data. If that happens, we will store and pass through all of the events that get sent to MAX through the get-next-event like you would be normally receiving during use. The only difference is we aren't able to pick out the select core events for a basic initialization. The same filters will apply for data you are registered for so no abnormal results should come from this.

Implementation Example

```
// Find the parent window (MAX) to register for events

var opener = window.opener || window.parent;

// Set up your subscriptions

var subscriptionTypes = ['agent', 'contacts'];


// Start listening for response messages

var doSomething = function (events) {
        var event = null;

        var eventIndex = null;

        for(eventIndex in events) {

                if (events.hasOwnProperty(eventIndex)) {
```

```
                              event = events[eventIndex];
                    }
          }
};
var listenForPostMessage = function (event) {
if (event.data && event.data.events && event.data.issuer === 'MAX') {
          logToConsole('=== received a post message with [' + event.data.events.length + ']
events ===');
          doSomething(event.data.events);
          }
};

// Add the listener for MAX client events.
window.addEventListener('message', listenForPostMessage);

// Send the registration message to MAX
opener.postMessage({ contactId: contactId, issuer: 'MyTestSite', messageType: 'Register-
ForClientEvents', subscriptionTypes: subscriptionTypes }, '*');
```

The Response Object

THE RESPONSE OBJECT

{

issuer: 'MAX',

contactId: int (Nullable) - the contact ID that was found as the persistent panel or was passed in. This will be null if no contactId was passed in or found.

events: [object, object, object ...] - All of the events that were returned in this set of events that matched the subscription types or contact filter.

}

Response for "SessionInfo" subscription's Message Shape:

{

messageType: "SessionInfo"

sessionToken: "mysessiontoken=="

}

THE SUBSCRIPTION ACKNOWLEDGMENT EVENT

If there is at least one **subscriptionType**, the subscription is allowed. If there are no valid **subscriptionTypes**, a console warning is logged that displays the invalid type. The first event in the set of events returned will be an event with the message type ClientEventSubscriptionAcknowledge. This event has the following structure:

{

contactId: (int) - If null, that means no contactId is associated with the subscription. This is a way for the client to see if their "contact" subscription worked.

messageType: ClientEventSubscriptionAcknowledge - (string). This is their specific acknowledge event message type.

reason: "Success" | "Invalid Contact Id" | "Invalid Subscription Types" - If it responds with the reason for the ERROR status code. This won't be super detailed in order to not add too much complexity. It will only return one failure reason. We check contact ID for validity first (not ZERO or not a non integer string). If that fails but there are also invalid subscription types, then we won't show the sub errors until they try again after fixing the contact id. That means we didn't keep a reference to the window, so the subscription failed.

status: "OK" or "ERROR" - If this returns Error, then we did not successfully create the post message subscription connection. They will need to fix these errors and try again.

}

In some cases, no subscription acknowledgment is sent. When the same window requests a second subscription, without first removing the existing subscription, no response will be sent and no additional subscription will be recognized. We will print a console warning that has the following structure:

{

console.warn('Error processing Client Even Subscription. Issuer: [' + subscriberObject.data.issuer + '] has already subscribed.')

}