Supporting Subscriptions for openGear Protocol (OGP) JSON Devices

The DashBoard platform provides support for a new subscription protocol that device developers who are part of Ross Video's openGear partner program can implement on OGP JSON devices to get the most out of the DashBoard Connect™ ecosystem.

Once a device developer adds support for the subscription protocol to an OGP device, then any DashBoard device panel can specify which parameter(s) it would like to subscribe to from that OGP device. DashBoard device panels that only subscribe to the required parameters will benefit from the reduced message size and minimal communication between the DashBoard Client and the OGP device.

Using subscriptions is recommended to:

- Optimize memory usage and communication
- Increase panel efficiency
- Load device panels faster

Important: Subscriptions is a protocol extension to OGP JSON, and does not support OGP binary devices.

Software Version

- The subscription protocol is only available in DashBoard v9.4 and later.
- The prerequisite Minimal Mode protocol must also be supported.

Assumptions

For the purposes of this document, a device panel refers to any DashBoard CustomPanel with a device context that is running in the DashBoard client, or a device user interface (UI) page that is served up by a device directly. This document assumes that the user is an openGear partner or Ross device developer. More general instructions for how a DashBoard user can work with the OGP devices that support subscriptions can be found in the DashBoard built-in Help or in the "**DashBoard PanelBuilder Features**" section.

For More Information on...

- About openGear Partners, see the opengear.tv website
- More important terms, see the "**Glossary**"
- Accessing the DashBoard built-in help, see the DashBoard top menu and navigate to **Help > Help Contents**.

Table of Contents

How Subscriptions Work

Subscriptions is a protocol built on top of minimal mode, and as such requires that support for minimal mode has already been implemented before implementing support for subscriptions.

But wait, what is Minimal Mode?

Minimal Mode allows you to specify which device parameters updates each device panel subscribes to. When you provide support for Minimal Mode in your device communication, it greatly decreases the response times for panels that require a large amount of parameter updates. Instead of the DashBoard panel receiving all of a device's parameter updates in **`full`**, the panel will only receive the **`minimal`** set of parameter updates that you have defined.

You will find more details on how to support Minimal Mode in the "**Before You Begin**" section.

Once I support Minimal Mode, how can I support Subscriptions?

To take advantage of subscriptions, all device panels need to provide their list of subscriptions using one of the ways in which subscriptions are declared in the panel's OGLML document structure.

To add subscriptions support in the OGLML Document Structure

1. You must add **`subscriptions="true"`** in the layout container or device context to support subscriptions.
2. You must add **`<subscription oids=""/>`** to support subscriptions.

✶ **Important**: You can use the device context in the topmost container, as shown in this example, or any other device context in the OGLML document structure. The **`<subscription oids=""/>`** tag does not need to be nested within **`<meta>`** tags, but it is recommended.

You can see an example of the correct OGLML document syntax below:

Example 1: **OGLML Document Syntax for a Layout Container**
Where the syntax shows an absolute container <abs/> with an opengear **contexttype** attribute. Note that any layout or container type attribute could be used.

```
<abs contexttype="opengear" id="_top" keepalive="false" objectid="MyUltritouch..."
subscriptions="true">
   <meta>
      <subscription oids="oid1, oid2, oid3*"/>
   </meta>
</abs>
```

Example 2: OGLML Document Syntax for a Device Context

Where the syntax shows a Subscriptions Panel with a device <context/> Tag. Using this format makes it possible to organize a panel that has multiple device contexts (or device sources).

```
<context contexttype="opengear" objectid="DeviceID..." subscriptions="true">
   <meta>
      <subscription oids="oid1, oid2, oid3*"/>
   </meta>
```

```
</context>
```

For More Information on...

- OGLML document structure, see: ***DashBoard CustomPanel Development Guide (8351DR-007)***
- Device contexts, see: "Glossary"

Ultritouch Panel Example with Subscriptions Support

```
<abs contexttype="opengear" id="_top" keepalive="false" objectid="My_Ultritouch"
objecttype="MyUltritouchDevice" subscriptions="true">
    <meta>
       <subscription oids="db.uiselectormapping, db.touch*, device.memoryusage"/>
    </meta>
    <table height="112" left="103" oid="0xFF01" top="102" width="295">
       <tr>
<label anchor="east" fill="none" insets="0,0,0,5" name="Display Name" weightx="0.0"/>
<param anchor="west" expand="true" fill="both" mid="-1" oid="0xFF01" showlabel="false"
weightx="1.0" weighty="1.0"/>
      </tr>
    </table>
    <table height="68" left="102" oid="device.memoryusage" top="247" width="298">
       <tr>
                  <label anchor="east" fill="none" insets="0,0,0,5" name="Device Memory Usage"
weightx="0.0"/>
                   <param anchor="west" expand="true" fill="both" mid="-1"
oid="device.memoryusage" showlabel="false" weightx="1.0" weighty="1.0"/>
            </tr>
    </table>
    <table height="86" left="103" oid="db.touch.type" top="380" width="301">
            <tr>
                   <label anchor="east" fill="none" insets="0,0,0,5" name="Type"
weightx="0.0"/>
                  <param anchor="west" expand="true" fill="both" mid="-1" oid="db.touch.type"
showlabel="false" weightx="1.0" weighty="1.0"/>
        </tr>
    </table>
    <table height="79" left="105" oid="db.touch.serial" top="523" width="303">
      <tr>
                   <label anchor="east" fill="none" insets="0,0,0,5" name="Serial"
weightx="0.0"/>
                   <param anchor="west" expand="true" fill="both" mid="-1"
oid="db.touch.serial" showlabel="false" weightx="1.0" weighty="1.0"/>
      </tr>
    </table>
```

```
</abs>
```

Explanation

The example above shows the OGLML document structure of an Ultritouch 4RU panel with subscriptions support. If you look at the structure, the subscriptions tag and subscription OIDs tag are highlighted in red in the device context.

As you can see the subscription tag lists the OIDs of some of the parameters used in this OGLML document:

```
<meta>
        <subscription oids=" db.uiselectormapping, db.touch*, device.memoryusage"/>
<meta/>
```

To make sure all the device parameters referenced in this OGLML document are updated and have their full description available, you need to add them to the subscription list(s) in the OGLML document. In the example above we are using the device's UI mapping, type, and memory usage.

The subscription tag above shows that the OIDs of these parameters are in subscription list:

```
<subscription oids="db.uiselectormapping, db.touch*, device.memoryusage"/>
```

Even though device's display name is used in the panel, explicit subscription to it is not required. This is because device's display name is part of reserved OIDs, which is considered a parameter in device's minimal set. Subscription to items in device's minimal set is done by default. If a parameter that is not part of a device's minimal set is used in the OGLML document file while its oid is not in any subscription tag, then the value for that parameter may not be accessible.

Each subscription tag will trigger a request to the device. We recommend combining all subscription OIDs into a single tag for a device context in an OGLML document, as shown above. The **<meta/>** tags are not required, but the **<subscription oids=""/>** tag must be in the device context.

In this case a wildcard is not required, but is used for **"db.touch*"** to ensure that any other OIDs that begin with the prefix **db.touch** will automatically be included. For example, **"db.touchserial"** is included.

To support subscriptions, in addition to updating the device panel's OGLML document structure, as shown above, the device itself must also support subscriptions.

Overview of Subscriptions

The steps to fulfill the requirements for subscriptions support are summarized below:

1. Each DashBoard device panel must indicate its own subscription list.
2. The DashBoard client collects the subscription list of each active panel and submits the entire subscription list to the device.
3. On the device side, the device must provide a list of all the parameters in a subscription list in addition to parameters in its minimal set.

4. Once implemented, the device should only send updates to the values of the items that the device panel has subscribed to in its subscription list.
5. When a panel is closed in DashBoard, the DashBoard client will send the list of items that are no longer in use to the device to unsubscribe from the updates.
6. The device uses the updated list to filter out which updates are required by the DashBoard client.

✶ **Note:** If the device sends updates to parameters that a DashBoard client has not subscribed to, the message will be ignored by the DashBoard client.

FAQ

How does message handling differ with subscriptions?

• If a DashBoard client supports subscriptions, it only requires change updates from the specified parameters that DashBoard subscribes to. This increases the performance significantly, particularly for devices with many parameters. Previously, devices without subscriptions were required to broadcast all parameter changes that occur via other clients via an asynchronous parameter reporting message to all connected DashBoard clients.

• With support for subscriptions, the device message is not required to provide the descriptors for all parameters. The only parameter descriptors that are required are those from the minimal set and the list of subscription parameters.

What if a subscriptions tag is not included?

Even if a device supports subscriptions, if a subscriptions tag with the attribute set to **`subscriptions=true`** is *not* included in the DashBoard panel's device context, or if the attribute is set to **`subscriptions=false`**, then it is assumed to be a legacy panel that receives *all* parameter updates. In this case it does not support subscriptions.

Essentially, if subscriptions is not supported, or the attribute is undefined or set to false, then subscriptions are not enabled for the OGLML document and the full device descriptor is requested.

What if a list of the required OIDs is not provided?

✶ **Important**: If you add **`subscriptions=true`** but fail to provide a subscriptions list of the parameters you wish to subscribe to, using the **`<subscription oids=""/>`** list, then you will only be subscribed to the minimal set.

How can I update my legacy panels (created before DashBoard v9.4)?

Legacy panels do not include the **`subscriptions="true"`** tag, and will be assumed to be false (**`subscriptions="false"`**). You must add a **`subscriptions="true"`** tag to indicate support for subscriptions. Updating your legacy panel will only produce the desired result when working with a device that supports minimal mode and subscriptions protocol.

For More Information on...

- OGLML document structure, see: ***DashBoard CustomPanel Development Guide (8351DR-007)***
- Subscriptions, see: ***openGear Software Development Guide*** **(8200DR-006)**
- Wildcard usage, see "**About Wildcards**" section in this guide.

Before You Begin

In this section, you can verify that you meet the software and device requirements. If you wish to use the DashBoard Proxy Server, additional requirements are also listed below. You will then learn about Minimal Mode requirements, and how to enable subscriptions features in the DashBoard Application.

Software and Device Requirements

Ensure that your software and devices meet the requirements listed below.

Software

- DashBoard Version 9.4 and later supports subscriptions

✶ **Important**: Minimal Mode support must be implemented before adding subscriptions support. For more information see, the ***Minimal Mode Application Note***.

Devices

- Devices must support OGP JSON protocol or devices must be shared through the DashBoard Proxy Server. Refer to the section "**DashBoard Proxy Server Requirements**" below.

✶ **Important**: Devices that use OGP binary must switch to use OGP JSON before implementing subscriptions.

DashBoard Proxy Server Requirements

When a DashBoard Client that supports subscription connects to a device through the DashBoard proxy server, the communication between DashBoard client and the proxy server can use subscriptions if the proxy server also supports subscriptions. Both the DashBoard client and DashBoard Proxy Server support subscriptions in DashBoard v9.4 and later. It is important to note that even if the device supports subscriptions, the proxy server connected to the device will always subscribe to all of device parameters. The proxy server can communicate with any device that supports either OGP JSON or OGP binary. The Proxy Server automatically converts any devices to OGP-JSON with subscription support regardless of which version of the protocol the device uses natively for its communication.

Implementing Minimal Mode

Minimal Mode is supported and enabled by default in DashBoard Version 8.7.1 and later.

✶ The protocol changes do not impact CustomPanels at all, and do not require any changes on the CustomPanel side.

When you provide support for minimal mode in your device communication, it takes the first step to make your device communication more efficient. With minimal mode, any connected devices will send only the basic required communication and only parameter updates from a defined minimal set of OIDs. Implementing minimal mode greatly reduces the memory consumption in DashBoard when a device is not actively in use. The benefits will be widely felt for any users with multiple connected devices that are on standby, and who wish to increase performance. Instead of DashBoard receiving all of a device's parameter updates in `full`, the panel will only receive the `minimal` set of parameter updates that you have defined.

You must follow the requirements listed here to enable it and follow the ***openGear Software Development Guide*** instructions to update your device communication to support minimal mode (details can be found under **JSON Reference > OGP Minimal Mode Support**):

- Minimal Mode must be enabled (default behavior)
- Device and Client messages must indicate support for the level of `detail` as `minimal` or `full`.

How to Implement Subscriptions

In this section you will learn how to implement subscriptions support in the DashBoard Client and OGP device.

✶ **Note**: This document assumes that support for minimal mode has already been implemented and that you are using a version of DashBoard that supports subscriptions (DashBoard version 9.4 or later). You can find instructions on how to implement minimal mode in the ***openGear Software Development Guide*** under **JSON Reference > OGP Minimal Mode Support**.

1. Verify that the DashBoard Client Supports Subscriptions

To verify that the DashBoard client supports subscriptions, you'll need to verify that the `"subscriptions"` flag and level of `"detail"` are present in the device `"payload"`.

Verify that the subscriptions flag is present in the device `"handshake"` message and that the `"subscriptions"` flag is set to `"true"`. The level of `"detail"` should be set to `"minimal"` if minimal mode has been implemented.

`"handshake"` Syntax

```
{
   "type": "handshake",
   "slot": slotID,
   "payload" : {
      "trusted" : trusted flag,
      "password" : "device password",
      "force" : force connection flag,
      "build" : "major.minor.micro YYYY-MM-DD HH:MM",
      "detail" : "minimal" | "full",
      "subscriptions" : true | false
      }
}
```

`"handshake"` Example

```
{
   "type": "handshake",
   "slot": slotID,
   "payload" : {
      "trusted" : trusted flag,
      "password" : "1232khfsd6I9f3",
      "force" : force connection flag,
      "build" : "major.minor.micro 2022-01-07 07:10",
      "detail" : "minimal",
      "subscriptions" : true
      }
}
```

Field	Type	Required	Description
detail	String	*Yes - required for subscriptions.	This can be set to **minimal** | **full**, where: • **minimal** - Receives the minimal set. • **full** - Receives all. (This is the default for legacy panels)
subscriptions	Boolean	Yes	This can be set to **true** or **false**, where: • **true**- Indicates that this connection has support for subscriptions. • **false** - Indicates that this connection does NOT support subscriptions.

2. Collect subscription list from "device-request"

The DashBoard client collects the subscription list from all open panels and adds them to the subscription list in the `"device-request"` message. The `"device-request"` message includes the level of `"detail"` set to `"subscription"` and the `"subscription"` field provides a list of parameter OIDs that you wish to subscribe to:

"device-request" Message Syntax

```
{
   "payload":{
      "detail":"subscription",
      "subscription":[
         "oid_id1",
         "oid_id2",
      ]
   },
   "slot":0,
   "type":"device-request"
}
```

Field	Type	Required	Description
detail	String	For subscriptions, • **Yes** - it is required that you set it to **subscription.** If you are not supporting subscription, • **No** - The default is set to **full**.	This can be set to **subscription** | **minimal** | **full**, where: • **subscription** - Receives the minimal set and subscriptions list. *This is required for the subscriptions list! • **minimal** - Receives the minimal set. • **full** - Receives all. (This is the default for legacy panels)
subscription	Array of String	For subscriptions, • **Yes** - it is required that you include a list of parameters. If you are not supporting subscription, • **No** - The default is set to **full**.	List of parameter OIDs for subscription. • **Note**: You must set the **"detail"** field to **"subscription"** if you are including a subscription list.

"device-request" Message Example

```
{
   "payload":{
      "detail":"subscription",
      "subscription":[
         "db.touch.type",
         "db.serial*"
      ]
   },
   "slot":0,
   "type":"device-request"
}
```

Explanation:

In response to the `"device-request"` message, the device sends a `"device"` message response. This response is much smaller than before, because it only includes a subset of the device's parameters in the full description (the client subscription list and the device minimal set). When subscriptions is not supported, the message size is significantly larger, because it includes a full description of all the device's parameters.

3. Populate the device Message

In response to the DashBoard Client's "device-request" message, the device sends a "device" message response. Follow the procedures below to populate the "device" message. This includes adding the subscription flag to the device message, adding the descriptor of items in the minimal and subscription set, and flagging items in the minimal set.

a) Add the subscriptions flag to the device message

Update the **"device"** message, to include the **"subscriptions"** flag to `true` in the **"payload"** object, and set the level of **"detail"**. You must also include the list of **"subscription"** OIDs that were returned.

"device" Message Syntax

```
{
   "type": "device",
   "slot": slotID,
   "payload" : {
       "slot": slot id,
       "subscriptions": true,
       "detail": "minimal" | "full" | "subscription",
       "subscription": [
          "OID_1",
          "OID_2",
          "OID_3"
       ],
       "menu-groups": { menu-groups object },
       "params": { params object },
       "multi-set-enabled": multi-set-flag
   }
}
```

Explanation:

You must add the `"subscriptions": true` flag to indicate that your device supports subscriptions. Alternatively, if the flag is not present in the payload, this indicates that the device does not support subscriptions (and ensures that DashBoard panels that were created before subscriptions was released in DashBoard v9.4 and earlier will continue to function as previously).

b) Add the full descriptor of items from the minimal set and subscriptions set

In the params section of the payload of the "device" message, only include the descriptors of items from the minimal set and subscription list.

c) Flag every item in minimal set with "minimalset":true

Add the new "minimalset" flag to the param Object in device's minimal set of parameters.

param Object Syntax

```
"_d_OID" : {
   "oid": "oid",
   "parent": "parent oid and element index",
   "name": "parameter name",
   "type": "parameter type",
   "readonly": read-only flag,
   "widget": "parameter widget",
   "precision": parameter precision,
   "maxlength": maximum string length,
   "totallength": maximum total length of string array,
   "constraint": { constraint object },
   "config": { config object },
   "value": parameter value,
   "minimalset": Boolean
}
```

Field	Type	Required	Description
minimalset	Boolean	No	A flag to indicate a parameter is part of a device's minimal set. If present and **true**, the parameter is in the device's minimal set. Default is **false**.
parent	string	No. This attribute is required if the device is responding to a panel's subscription request for an element inside of a Struct, such as a **STRUCT_ARRAY**.	This attribute tells the DashBoard Client which oid this parameter is a child of. The string value is the oid of the parent oid. **Note**: For Struct types, such as a **STRUCT_ARRAY**, the string value must also include the **index** of the parent element in the struct. For more information, refer to the device's **basic-param-info** response, which now includes a new metadata attribute that includes a parent-in-payload child attribute.

Explanation:

Every parameter that is flagged as part of the **"minimalset"**, will always remain active and updates to it will always be required.

4. Provide support for the "update-subscriptions" message

a) Support DashBoard Client "update-subscriptions" message

After the device-request message is sent, every time a new device panel opens or closes, the DashBoard client collects the subscriptions and sends an `"update-subscriptions"` message to the device. This message allows the client to `"subscribe"` to or `"unsubscribe"` from the subscriptions list (of parameter OIDs).

Syntax

```
"type": "update-subscriptions",
"slot": slotID
"exe-id": "unique ID string",
"payload":
{
   "subscribe": [
      "ID_1",
      "ID_2*",
      …
   ],
   "unsubscribe": [
      "id3",
      "id4",
      …
   ]
}
```

Field	Type	Required	Description
type	String	Yes	Set to `update-subscription`.
slot	Number	Yes	Destination slot for this message.

Field	Type	Required	Description
exe-id	String	Optional* **Note:** It is required if the DashBoard client is expecting a response.	The unique execution ID that DashBoard client adds to the header of the update-subscription message when it requests new subscriptions. When this ID is present in the request, the device adds the ID to the header of the params message response. This allows DashBoard to verify that it has received the response to a specific request.
payload	object	Yes	Provides the lists of parameter OIDs to subscribe or unsubscribe to.
subscribe	Array of String	Yes	Subscribe to the provided list of parameter OIDs.
unsubscribe	Array of String	Yes	Unsubscribe to the provided list of parameter OIDs.

b) Update the Device "params" Message

When an update subscription message has a `"subscribe"` section with content, the device is required to reply with a `"params"` message that includes the descriptors of the items that the DashBoard client is subscribed. It should also contain the same `"exe-id"` if the unique ID is present in the update subscription request.

Syntax

```
{
        "type": "params",
        "slot": slotID,
        "exe-id": "execution ID",
        "payload" : {
           "_d_OID1": { param object 1 },
           "_d_OID2": { param object 2 },
           "_d_OID3": { param object 3 }
        }
},
```

Field	Type	Required	Description
type	String	Yes	Set to params.
slot	Number	Yes	Destination slot for this message.
exe-id	String	Yes - Only required if an `exe-id` exists in the update subscription message request.	This field is only required if the `exe-id` exists in the update subscription message request. Then the device response should include the `exe-id` in the header of the params message. **Note**: The ID is an arbitrary string provided by the requesting client. It is not guaranteed to be globally unique.
payload	Object	Yes	The payload will include the parameter descriptor information for every requested parameter.
_d_OID	Param Object (descriptor)	No	Parameter descriptor for each parameter object.

Explanation:

When a device UI panel is opened in DashBoard, the DashBoard client will send the list of items in the panel's subscription list to the device and "`subscribe`" to the updates. When a panel is closed in DashBoard, the DashBoard client will send the list of items that are no longer in use to the device and "`unsubscribe`" from the updates. The device should use this information to update its local subscription list for this client and also to determine which updates are no longer required by the DashBoard client.

5. Populate the DashBoard Client's Basic Param Info Request

Follow the procedures below to populate the "**`basic-param-info-request`**" message. The basic param info request, or BPI request, is a message that is sent from the DashBoard client side.

a) Update the "basic-param-info-request"

The **`basic-param-info-request`** defines a request to get the name, type, oid, template, and length for the children of a particular OID. This is used to build parameter trees, anywhere you can see a list of all the devices and their parameters in the DashBoard CustomPanel. Whenever you open up this type of menu, a BPI request will be sent.

In the payload the **`recursive`** flag (**`true`** or **`false`**) indicates whether you want the response to include the entire sub-tree or just the immediate children of the specified parameter (**`oid`**). Where **`oid`** could be set to an asterisk (**`*`**) for all or a specific device parameter OID.

If a DashBoard user clicks through the parameter tree, DashBoard will send out the OIDs requested, and fill in the tree. Initially, DashBoard sends out **`oid:*`** and **`recursive=false`**, this indicates that it wants only the top level of OIDs returned. This implements a form of lazy-loading, where only the required OIDs are loaded as you need them. The BPI only includes the minimal amount of information required to fill in the tree and it only gets the required info when it's needed.

If a user attempts to use the **Search** to find a particular parameter or OID, DashBoard must send a request with **`recursive=true`** to get all of the information necessary to perform the search. When **`recursive`** is set to **`true`**, it ensures that you will get all the children and the children's children recursively.

Note: It is required that you support **`recursive=true`**, and recommended that you support **`recursive=false`**. If you choose not to support **`recursive=false`**, then you must still respond with a recursive list of BPI.

An example of a tree views in DashBoard includes the Visual Logic Editor for the Device & Parameters area (which appears only when you are editing ogScript code). For more details about the Visual Logic Editor, see the DashBoard User Guide.

The parameter list for **Devices & Parameters** is shown below:

Figure 1 *Visual Logic Editor - Devices & Parameters Tree View (Close-up)*

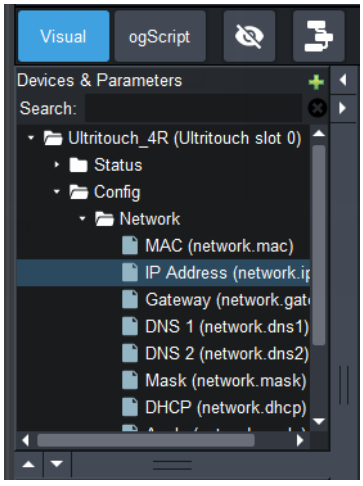
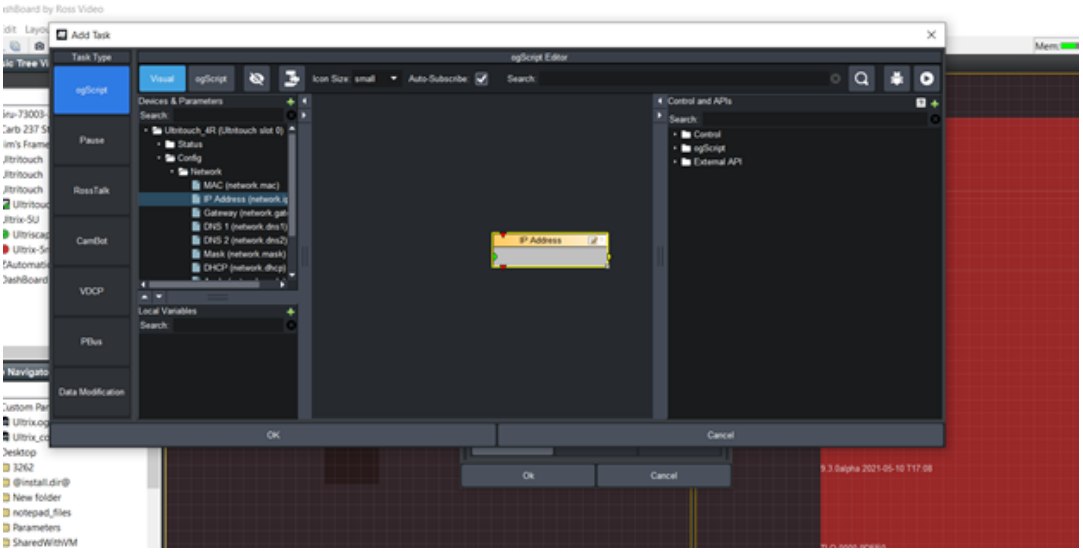


***Figure 2** Visual Logic Editor - Tree Views (Full View)*



"*basic-param-info-request*" *Syntax*

```
{
   "type":"basic-param-info-request",
   "slot": slot ID,
   "exe-id":"execution id",
   "payload":{
      "oid": oid*,
      "recursive": boolean
   }
}
```

Field	Type	Required	Description
type	String	Yes	Set to `basic-param-info-request`.
slot	Number	Yes	Destination slot for this message.

Field	Type	Required	Description
exe-id	String	Optional *If you are not expecting a response.	The unique execution ID that DashBoard client adds to the header of the basic-param-info-request message when it requests new basic param info. This allows DashBoard to verify that it has received the response to a specific request.
payload	Object	Yes	The `"payload"` provides the `oid` and `recursive` field.
oid	String	Yes	The "oid" is the parent parameter you want the children for. This is a single oid (not a list). If the `"oid"` is `"*"`, it means you want the top-level parameters.
recursive	Boolean	Yes - • It is **required** to support `"recursive": true` • It is recommended to also support `"recursive": false`	In the payload the `recursive` flag (`true` or `false`) indicates whether you want the response to include the entire sub-tree or just the immediate children of the specified parameter.

Example Retrieving OIDs and Children of OIDs Recursively

To get an OID and the children of the specified OID, you must add the OID. You have the option to set `"recursive"` to `"true"` or to `"false"`,

```
{
   "type":"basic-param-info-request",
   "slot": 25,
   "exe-id":"1",
   "payload":{
      "oid":"auto.detect",
      "recursive":true
   }
}
```

Example Retrieving Only Top Level OIDs

To get only the top level of OIDs (or parent OIDs), set "recursive" to "false", and set the "oid" to an asterisk, "*".

```
{
   "type": "basic-param-info-request",
   "slot": 25,
   "exe-id": "1",
   "payload":{
      "oid": "*",
      "recursive": false
   }
}
```

b) Update the "basic-param-info" response from a device

Follow the procedures below to populate the "basic-param-info" response from a device.

"basic-param-info"

Defines a response message from a device. The basic param info response provides support for specific OIDs and the option to recursively retrieve all of the children of the specified OID.
Note: There is a new **metadata** attribute that provides support for subscription requests that include the parent in the payload Object, such as STRUCT_ARRAY types.

Syntax of Basic Param Info Response

```
{
    "metadata": {
      "index": value of Index,
      "parent-in-payload": boolean,
      "oid": "devices",
      "recursive": boolean
   },
    "exe-id": "executionID",
    "slot": slotID,
    "type": "basic-param-info",
    "payload": {
      "_d_OID1": "basic-param-info object 1",
      "_d_OID2": "basic-param-info object 2"
   }
}
```

Basic Param Info Response Attributes

Field	Type	Required	Description
metadata	String	Optional.	Used in scenarios where the device may want to respond to a basic-param-info" type request from the DashBoard Client with a more general (higher level) reply.
exe-id	String	Yes	This is the unique execution ID that DashBoard client added to the header of the **basic-param-info** message when it requests new basic param info. This allows DashBoard to verify that it has received the response to a specific request.
slot	Number	Yes	Destination slot for this message.
type	String	Yes	Set to `basic-param-info`.
payload	Object	Yes	The payload will include information for every child parameter requested.
OID	Param Object (descriptor)	No	Parameter descriptor for each parameter object.

Child Attributes for Metadata field

Field	Type	Required	Description
oid	object	Yes	Allows the device to communicate to DashBoard which parameter it's reporting the basic-param-info for. If you plan to change which **oid** you're reporting the **basic-param-info** for, make sure to populate this field (and the **index** - if applicable).
index	object	If the device is reporting a higher level parameter, like an array type, then this is required.	See description for "oid" above.
parent-in-payload	object	No - See description for when this field is required.	DashBoard usually expects only the children of the requested parameter in the **basic-param-info** message. If the device instead reports the parent in the message, then this attribute must be set to "true".
recursive	object	Yes	This indicates to DashBoard whether your message contains the full (recursive) tree of children for the requested oid.

Example Retrieving OIDs and Children of OIDs Recursively

This example shows the device response when you have set `"recursive"` to `true` in the `"basic-param-info-request"`.

```
{
   "exe-id":"1630693459714:2",
   "slot":0,
   "type":"basic-param-info",
   "payload":{
```

```
        "0xFF10":{
           "name":"DEVICE_IP_ADDRESS",
           "type":"STRING"
        },
        "deviceoptions.calibration":{
           "name":"Run Calibration",
           "type":"INT16"
        },
        "devices":{
           "name":"Devices",
           "type":"STRUCT_ARRAY",
           "length":4,
           "children":[
              {
                 "name":{
                    "name":"Device Name",
                    "type":"STRING"
                 },
                 "status":{
                    "name":"",
                    "type":"INT16"
                 },
                 "connection_settings":{
                    "name":"Connection Settings",
                    "type":"STRUCT_ARRAY",
                    "length":10,
                    "children":[
                       {
                          "key":{
                             "name":"Key",
                             "type":"STRING"
                          },
                          "value":{
                             "name":"Value",
                             "type":"STRING"
                          }
                       }
                    ]
                 }
              }
           ]
        }
     }
}
```

Example Retrieving Only Top Level OIDs

This example shows the device response when you have set `"recursive"` to `false` in the `"basic-param-info-request"`. Note that the `"devices"` OID, does not retrieve the child OIDs.

```
{
   "exe-id":"1630692185591:0",
   "slot":0,
   "type":"basic-param-info",
   "payload":{
      "0xFF10":{
         "name":"DEVICE_IP_ADDRESS",
         "type":"STRING"
      },
      "deviceoptions.calibration":{
         "name":"Run Calibration",
         "type":"INT16"
      },
      "devices":{
         "name":"Devices",
         "type":"STRUCT_ARRAY",
         "length":4
      }
   }
}
```

Examples showing when to use the Metadata field

- When the DashBoard Client requests BPI for devices.1.connection_settings.0:

```
{
   "payload":{
      "oid":"devices.1.connection_settings.0",
      "recursive":false
   },
   "exe-id":"1648571608632:32",
   "slot":1,
   "type":"basic-param-info-request"
}
```

- If a device decides that's too granular, it responds with the entirety of the "devices" parameter:

```
{
  "metadata":{
     "index":-1,
     "parent-in-payload":true,
     "oid":"devices",
     "recursive":true
  },
  "exe-id":"1648571608632:32",
  "slot":1,
  "type":"basic-param-info",
  "payload":{
     "name":"Devices",
     "type":"STRUCT_ARRAY",
     "length":2,
     "children":[
        {
           "name":{
              "name":"Device Name",
              "type":"STRING"
           },
           "status":{
              "name":"",
              "type":"INT16"
           },
           "connection_settings":{
              "name":"Connection Settings",
              "type":"STRUCT_ARRAY",
              "length":2,
              "children":[
                 {
                    "key":{
                       "name":"Key",
                       "type":"STRING"
                    },
                    "value":{
                       "name":"Value",
                       "type":"STRING"
                    }
                 },
                 {
                    "key":{
                       "name":"Key",
                       "type":"STRING"
```

```
            },
            "value":{
               "name":"Value",
               "type":"STRING"
            }
         }
      ]
   },
   "connection_state":{
      "name":"Current Connection State",
      "type":"INT16"
   }
},
{
   "name":{
      "name":"Device Name",
      "type":"STRING"
   },
   "status":{
      "name":"",
      "type":"INT16"
   },
   "connection_settings":{
      "name":"Connection Settings",
      "type":"STRUCT_ARRAY",
      "length":1,
      "children":[
         {
            "key":{
               "name":"Key",
               "type":"STRING"
            },
            "value":{
               "name":"Value",
               "type":"STRING"
            }
         }
      ]
   },
   "connection_state":{
      "name":"Current Connection State",
      "type":"INT16"
   }
}
```

```
        ]
    }
}
```

Example to show the effect of the parent-in-payload flag:

```
{
    "payload":{
        "oid":"struct",
        "recursive":false
    },
    "exe-id":"1648575366928:3",
    "slot":0,
    "type":"basic-param-info-request"
}
```

Example with no flag (reporting only the children of the struct):

```
{
    "slot":0,
    "type":"basic-param-info",
    "exe-id":"1648575366928:3",
    "payload":{
        "name":{
            "name":"Name",
            "type":"STRING"
        },
        "integer":{
            "name":"Integer",
            "type":"INT16"
        },
        "substruct":{
            "name":"Child Struct",
            "type":"STRUCT"
        }
    }
}
```

Example with flag (reporting parent and children)

```
{
    "metadata":{
        "parent-in-payload":true
    },
    "exe-id":"1648575366928:3",
```

```
    "slot":0,
    "type":"basic-param-info",
    "payload":{
       "name":"My Struct Param",
       "type":"STRUCT",
       "children":[
          {
             "name":{
                "name":"Name",
                "type":"STRING"
             },
             "integer":{
                "name":"Integer",
                "type":"INT16"
             },
             "substruct":{
                "name":"Child Struct",
                "type":"STRUCT"
             }
          }
       ]
    }
}
```

Related to

- basic-param-info-request — (on page 4–17)

"basic-param-info" object

See the entry for **basic-param-info-request** and **basic-param-info** for more information.

Note: It is required that you include the oid, name, type (and length of the parameter for arrays only) in the basic-param-info object.

Syntax

```
"OID":{
   "oid": "oid",
   "name": "parameter name",
   "type": "parameter type",
   "length": "integer"
}
```

Member of

- basic-param-info — (on page 4–20)

Members

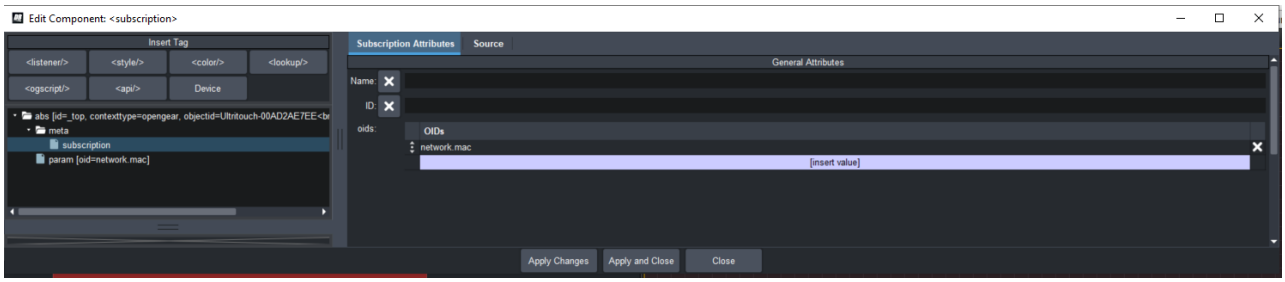
Member	Type	Required	Description
oid	String	Yes	The Object Identifier (OID) for this parameter. The OID must be globally unique within the scope of the device.
name	String	Yes	Parameter Name
type	String	Yes	The data type for the parameter. See "**Data Types**" on page 3–33 for a list of valid data types.
length	Integer	Yes - It is required for array data types.	The length of the parameter.

6. Add the subscription flag to the DashBoard CustomPanels

a. In DashBoard, to add the subscription flag to the custom panel, go to the **Component Editor > Source** tab and within the top level container of the `opengear` device context, include a list of all of the OIDs used in that custom panel page.

Tip: You can navigate to the editor by double-clicking on a blank area in your DashBoard CustomPanel while in PanelBuilder **Edit** Mode, and the **Component Editor** will open with the header selected in the side navigation. You can see an example with an Ultritouch panel below:

b. Add the `<subscription oids=""/>` tag. Once you have added the tag, in the tree view on the left side, select **subscription**. Select the **Subscription Attributes** tab, and under **OIDs**, you can add the list of device parameter OIDs you wish to include.



Syntax

```
<abs contexttype="opengear" id="_top" keepalive="false" objectid="My_Device"
objecttype="MyDevice" subscriptions="true">
   <meta>
      <subscription oids="list of comma-separated oids here"/>
   </meta>
</abs>
```

Ultrix Example

```
<abs contexttype="opengear" id="_top" keepalive="false" subscriptions="true"
objectid="MyUltrix" objecttype="Ultrix">
   <meta>
      <subscription oids="devices.*,types.audiomixer*,params.framesrc.ports.0.port"/>
   </meta>
...
```

Note: You can use wildcard asterisks to include multiple OIDs simultaneously that have the same starting prefix in the name. The wildcard should be added after this prefix. These wildcards are useful when you don't want to type out a whole list of similar OIDs manually. Instead you can add a subset of OIDs by including a wildcard. If wildcards are used, your list of subscriptions are optimized by DashBoard to use the wildcard that includes the most items. For more details see, "**About Wildcards**".

About Wildcards

Adding a wildcard asterisk to a list of parameter OIDs in a DashBoard device panel, will allow you to quickly add multiple sets of parameter OIDs that start with the same prefix. You can only add an asterisk to the end of an oid prefix name. The asterisk means that you will subscribe to all parameters that start with the prefix you entered.

For example, if you wanted to add three OIDs, **`types.audiomixer`**, **`types.audiomixerpartition`** and **`types.audiosound`**, you could use the following wildcards: **`ty*`**, **`types.audio*`**, or **`types.au*`**. If you use more than one wildcard that applies to the same parameters, DashBoard will choose the most efficient wildcard to optimize. In the example above, **`ty*`** would be used. You cannot add a wildcard before the prefix or have text after the wildcard. For example, **`*ypes.`** and **`ty*p`** are not valid.

7. Add the subscription list to the DashBoard device panels

You must add support to subscribe and/or unsubscribe from parameter updates in the device panel's OGLML structure. You can also use the template that is provided in the DashBoard PanelBuilder Script Palette.

For More Information on...

- Templates, see: "**Using the DashBoard Script Palette's Command Templates**"

"params.subscribe" Syntax

```
//SUBSCRIBE
var subList = new Array();
   subList.push("oid1");
   subList.push("oid2");
var subscriptionOwnerObject = params.subscribe(subList, callback);
```

"params.subscribe" Example

```
 <task tasktype="ogscript">
          var subList= new Array();
             subList.push("deviceoptions.speakerlevel");
             subList.push("db.touch.version.*");
          var subscriptionOwnerObject = params.subscribe(subList, callback);
   ogscript.putObject('my-subscription-owner-object', subscriptionOwnerObject);
</task>
```

Function	Parameters	Returns	Description
subscribe	[Array of strings, callback]	Returns `subscriptionOwnerObject` for later use to unsubscribe.	Subscribes to parameters with the provided OIDs.

Explanation

In this example, the **`ogscript.putObject`** is used to retain the result of the **`params.subscribe`** function, which is later used to unsubscribe.

"params.unsubscribe" Syntax

```
//UNSUBSCRIBE
params.unsubscribe(subscriptionOwnerObject);
```

Function	Parameters	Returns	Description
unsubscribe	[subscriptionOwnerObject]	N/A	Unsubscribes from the OIDs provided by the subscriptionOwnerObject.

"params.subscribe" Example

```
 <task tasktype="ogscript">
          var subscriptionOwnerObject = ogscript.getObject('my-subscription-owner-object');
          params.unsubscribe(subscriptionOwnerObject);
</task>
```

Explanation

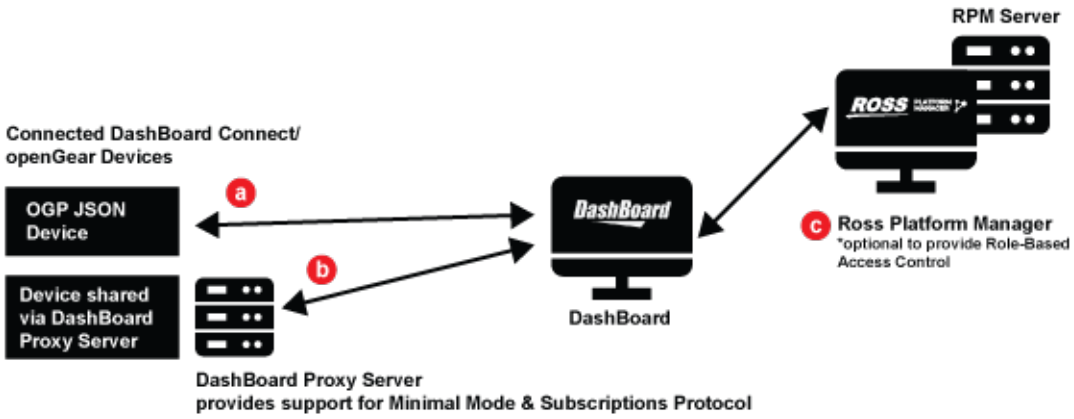
In the subscribe example above, the **`ogscript.putObject`** is used to retain the result of the **`params.subscribe`** function and **`ogscript.getObject`** fetches it when we want to unsubscribe (**`params.unsubscribe)`**. You can see that the subscribe response object is used to unsubscribe.

Now that you have successfully implemented subscriptions support, make sure that you leverage the built-in automations within DashBoard to support subscriptions. See, "**Using the DashBoard Script Palette's Command Templates**".

DashBoard PanelBuilder Features

A DashBoard panel builder may wish to connect to OGP devices with subscriptions support, or share multiple devices through the DashBoard Proxy Server (which provides support for Minimal Mode and Subscriptions protocol). This workflow greatly improves the user experience for any users who need DashBoard to run smoothly with numerous connected devices, but especially if those devices send the DashBoard Client a high volume of parameter updates. Subscriptions improves the DashBoard platform's performance by eliminating any unnecessary communication with connected devices that are not in use, and that otherwise would send a high volume of parameters even while idle. You can see an example of a typical workflow below:

Figure 3 *DashBoard Panel Workflow with Devices that Support Subscriptions*



a. **OGP JSON Device** — Any OGP JSON device that supports subscription. This includes Ross devices in the DashBoard Connect ecosystem that support subscriptions, such as Ultritouch.

b. **Device Shared via DashBoard Proxy Server** — The DashBoard Proxy Server will provide support for Minimal Mode and Subscriptions Protocol.

c. **The Ross Platform Manager (RPM)** — This is an optional component that can be used to provide Role-Based Access Control for connected devices.

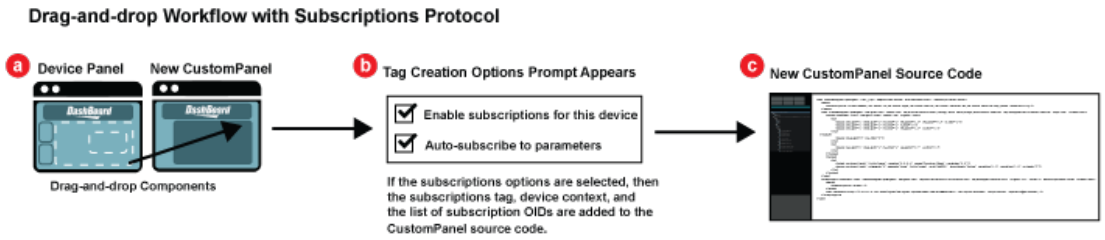
Overview

DashBoard panel builders can use the following PanelBuilder automations to create device panels for OGP devices/ or devices shared through the DashBoard Proxy Server:

- Drag-and-Drop Panel Components with Subscriptions Support - Reuse panel components from an existing device panel that supports subscription, and use the provided checkboxes to choose whether newly dragged in components will have subscriptions turned on or off, and whether to add the parameters to the subscription list for the device.
- Adding Subscriptions Support for a Device Context - When adding a device to DashBoard context, panel builders can choose to enable subscriptions for that device using the new "**Enable subscriptions for this device**" checkbox in the Device Context Dialog.
- Using the DashBoard Script Palette's Command Templates - Use the ogScript Editor's Script Palette to add provided code snippets for **`param.subscribe`** and **`param.unsubscribe`**.

Drag-and-Drop Panel Components with Subscriptions Support

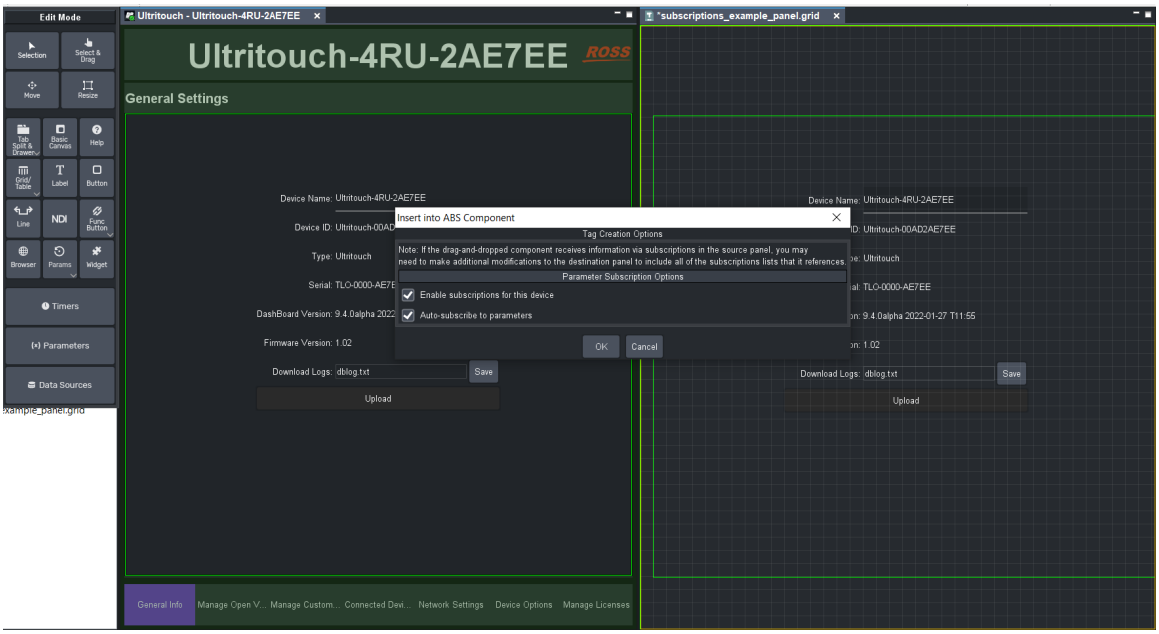
DashBoard provides the ability to drag-and-drop components from one device panel for reuse in a secondary CustomPanel, and the any data-backed components will receive any updates from the original source panel. DashBoard provides automatic support for subscriptions using the following workflow below:



a. When you want to bring components from an existing openGear device frame that supports subscriptions into a new CustomPanel, you can drag and drop parameters from the existing device panel into your new panel.

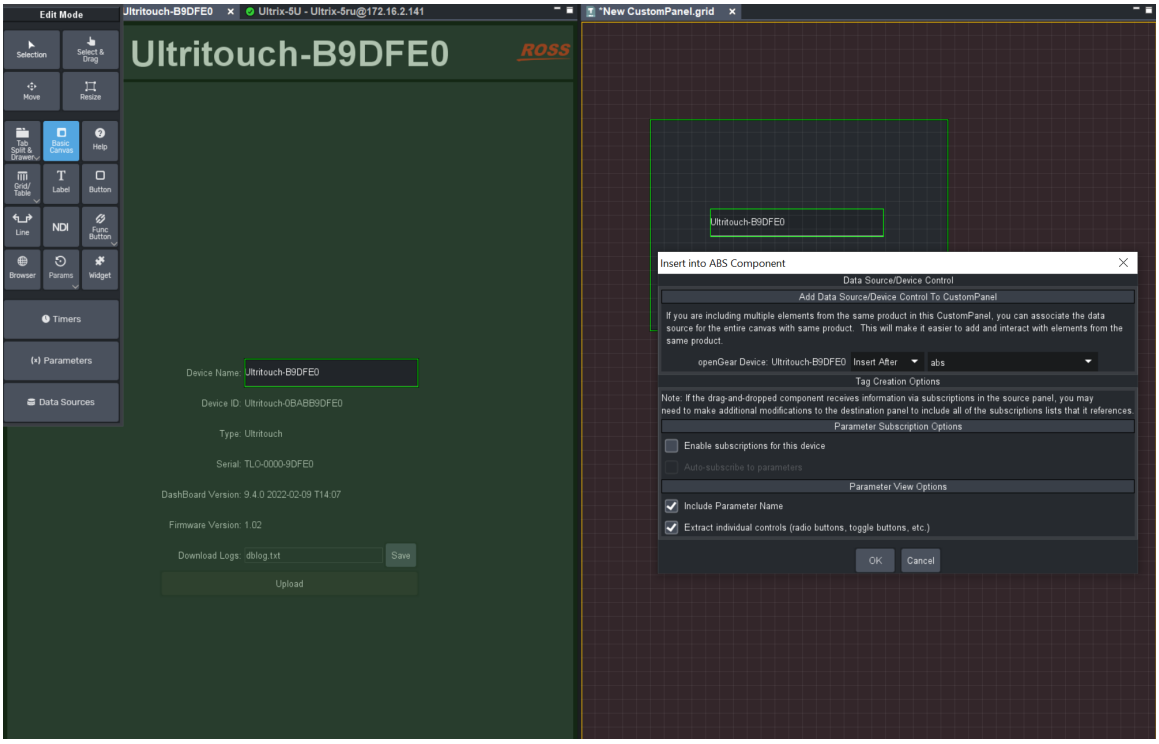
b. When you drag a parameter in, a Insert into ABS Component dialog appears. This dialog now has two new options: "**Enable Subscriptions for this device**" and "**Auto-subscribe to parameters**", as shown in the screen-shot below. Checking both of these boxes will automatically add the parameter to the panel's subscription list.

Figure 4 *An Ultritouch Device that supports Subscriptions (left) and a New Panel (right)*



✶ **Tip**: You can also drag-and-drop a component into an existing area, such as a basic canvas, and the dialog prompt will have additional options that allow you to choose where you want the data source to be inserted.

***Figure 5** An Ultritouch Device component that has been dropped into an existing basic canvas*

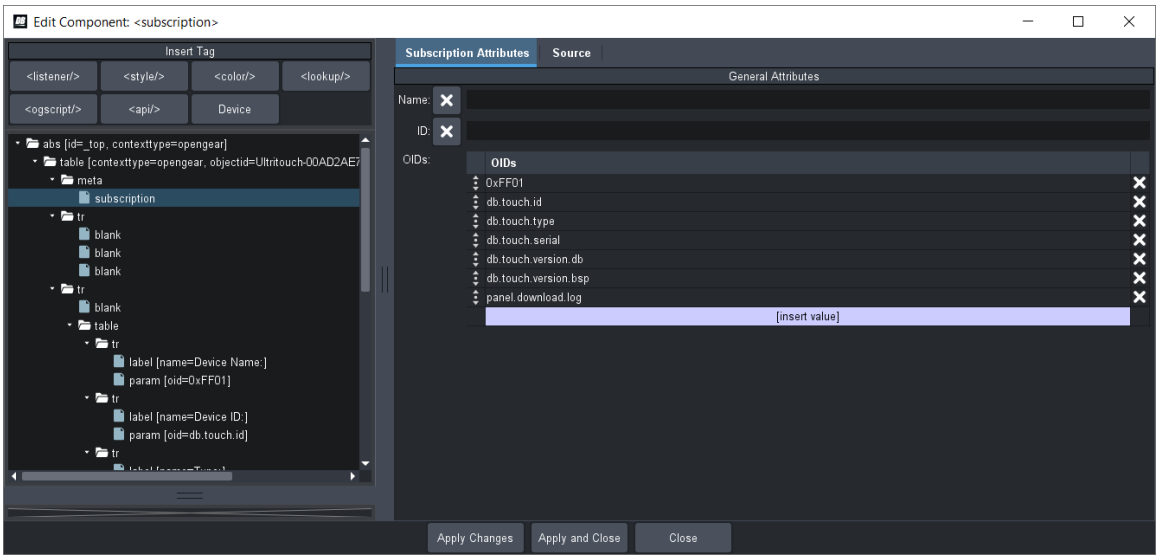


c. You can verify that it's been added, by double-clicking on the displayed parameter to open the **Component Editor** dialog and navigating to the **Source** code tab to view the updated subscription list. It will include the parameter's oid in the subscription list:

```
<meta>
   <subscription oids="0x907"/>
<meta>
```

You can also view the OIDs, in the **Component Editor** dialog's tree view by selecting **Subscriptions** in the tree view. The Subscriptions Attributes tab on the right includes the full list of OIDs, as shown below:

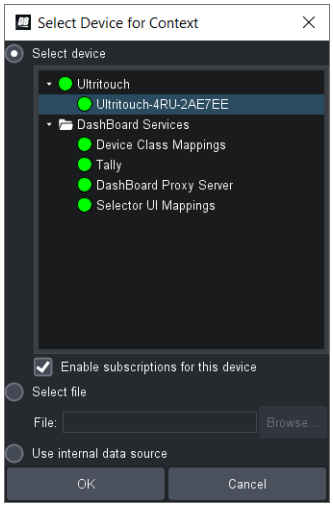
Figure 6 *Subscriptions OIDs displayed in the Subscriptions Attributes tab*



Adding Subscriptions Support for a Device Context

When adding a device to DashBoard panel's context, you can choose to enable subscriptions for that device using the new "**Enable subscriptions for this device**" checkbox in the Device Context dialog, as shown below:

Figure 7 *Enabling Subscriptions Support in the Device Context Dialog*



✶ **Note:** The **Enable subscriptions for this device** checkbox only appears if the selected device supports subscriptions, if not it will be disabled.

Selecting the **Enable subscriptions for this device** checkbox, and clicking **OK**, will automatically add a **`subscriptions="true"`** tag to the device context, as shown in the example below:

```
<abs contexttype="opengear" id="_top" keepalive="false" objectid="MyUltritouch"
objecttype="Ultritouch Device" subscriptions="true">
```

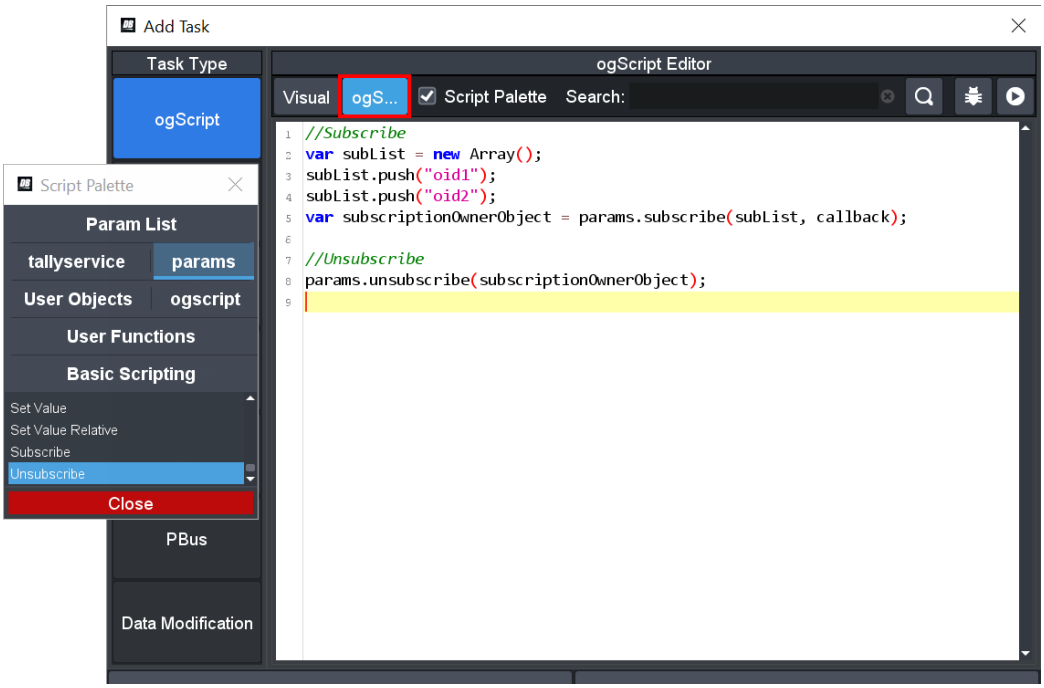
If the checkbox is not selected, then DashBoard automatically adds a **`subscriptions="false"`** tag to the device context. For more details on how to add a device context, see the procedure below.

Adding a device context

1. In PanelBuilder **Edit Mode**, double-click anywhere on the blank canvas to open the Component Editor, and scroll down to the **Data Source/ Device Control** area.
2. For the openGear or XPression Datalinq field, select the checkbox and click **Configure**. The **Select Device for Context** dialog appears, with the option to select a device, and enable subscriptions (only if that device supports subscriptions).

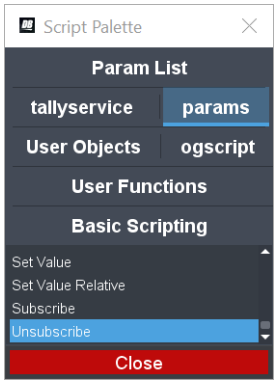
Using the DashBoard Script Palette's Command Templates

You can use the **Subscribe** and **Unsubscribe** command templates that are built into the **Script Palette**. Follow the steps in the procedure below to use the templates, as shown below:



To add the template for the Subscribe or Unsubscribe command as code snippets

1. Ensure that you have a button with a task added.
2. In the manual **ogScript Editor,** select the **Script Palette** checkbox, and click the **params** tab.
   Note: The Script Palette is currently only available in the manual **ogScript** text editor, not the **Visual** editor.
3. Move the cursor to the area that you would like to add the code snippet, and then double-click on the "**Subscribe**" or "**Unsubscribe**" templates to add the code snippet there.



4. Close the Script Palette when you are finished, and save your changes.

Glossary

This section provides a list of terms used in this document, and defines them for clarity.

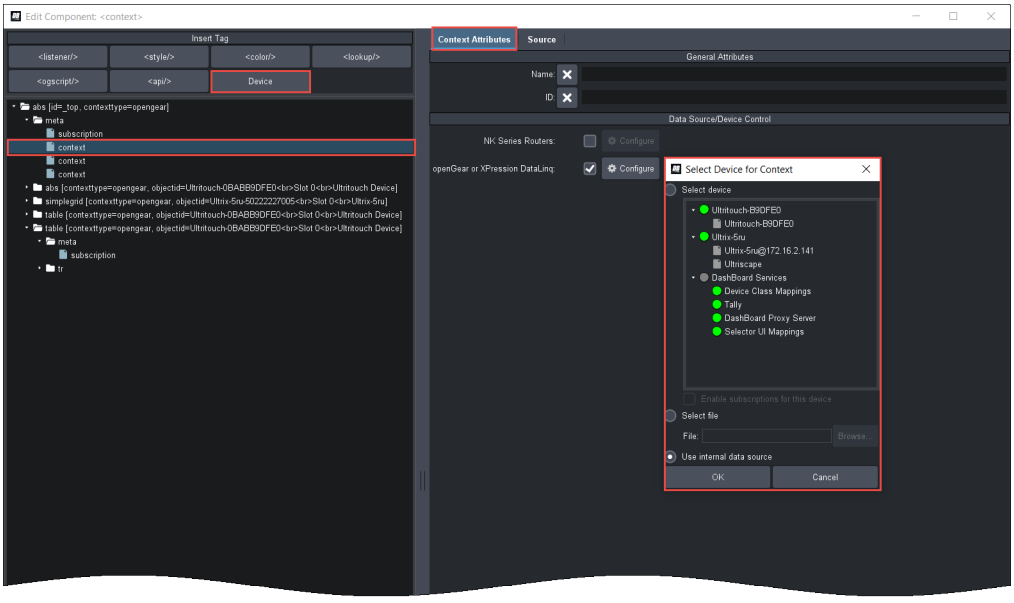
DashBoard Application/ DashBoard platform — Refers to Ross Video's DashBoard platform, which is a free application designed for remote control and monitoring of the openGear architecture and openGear™ ecosystem.

DashBoard Client side — Refers to the messages sent from the DashBoard Client side (the OGP JSON Client).

DashBoard Connect™ Ecosystem — The ecosystem of openGear devices manufactured by our openGear partners that can be controlled using the DashBoard Facility Control System.

Device context — Subscriptions will be implemented within the device's context, which defines the scope within the OGLML document. Each default panel is created with a default context named `"opengear"`, but if multiple devices are linked to the OGLML document, each will have its own separate context. A device context can be a `<context/>` tag or it can be defined in any container tag by setting the `contexttype` attribute to `"opengear"` and specifying the device's ID with the `objectid` attribute.

The diagram below shows a **Device** button that allows you to insert a `<context/>` tag, which appears highlighted in the tree view on the left. When the context is still highlighted, on the right side under the **Context Attributes** tab, you can configure a device for that context.



Device side — Refers to the device messages sent from the device side (the OGP JSON Server).

openGear Protocol (OGP) — OGP is the protocol for parameter reporting, control and alarm reporting.