Distributed Services Architecture (DSA)

DSA brings together entities like devices, services, and applications into one data model that updates in real time. By representing these entities as one system, DSA makes possible or simplifies various tasks, such as analytics, inter-device communication, distributed computing, and application development.

You can read more about DSA on the DSA home page.

Structure of DSA

DSA represents each of the entities in the system as one of the following types:

- **Broker**: Brokers manage security, links, subscriptions, and node permissions. Other entities in the system can operate only as permitted by a broker. A broker also saves configuration data to disk. A broker also routes data from a source to a destination.
- **DSLink**: DSLinks create, publish, and interact with data. DSLinks, which connect to brokers, can subscribe to data in the DSA system, to receive data whenever the data changes. You can download DSLinks from IOT-DSA on Github.
- **Node**: General term for the components connected by DSA. Node can be brokers, DSLinks, metrics, actions, attributes, or folders containing other nodes.
- **Metric**: A metric is a key/value pair. The value can be any of the data types listed in Supported Data Types, including an arbitrary value map. A metric can originate on any broker, DSLink, or other node.
- **Data Node**: Data nodes enable data to be stored on the broker's host server.
- **Action**: An action is an invocable command that can affect an entity. For example, an action might create a node or set a metric value. An action can exist on any broker, DSLink, other node, or metric.
- **Attribute**: An attribute is metadata for the selected entity, represented as a key/value pair. An attribute can exist on any broker, DSLink, other node, or metric.

Upstream and Downstream Connections

Upstream and *downstream* connections are important concepts for permissions configuration when you work with multiple servers or with multiple brokers on one server. A downstream entity requests permission, and an upstream entity either grants or refuses that permission. A broker is always upstream from its DSLinks. A broker can be either upstream or downstream from another broker.

Connecting Two DSA Installations (Brokers)

When you connect two DSA installations, one is an upstream broker and the other is a downstream broker.

For an example, see this video.

To connect two DSA installations:

1. From the downstream broker, right-click **sys** > **upstream**, and choose **Add Upstream Connection**.
2. For **Name**, specify the node name that you want to appear in the downstream tree to represent the upstream broker.
3. For **Url**, specify the port, followed by `/conn`, for example `http://localhost:8081/conn`.
4. For **Broker Name**, specify the name that you want to appear in the upstream tree to represent the downstream broker.
5. For **Group**, specify the permission group for the upstream broker.
6. Log in to the upstream broker.
7. To authorize the downstream broker to remove it from quarantine, from the upstream broker, choose **sys** > **quarantine** > **Deauthorize**, and specify the downstream broker.

Supported Data Types

DSA supports these data types:

- **String**—A sequence of characters or an empty string.
- **Number**—A number or a null value.
- **Bool**—A true or false value.
- **Array**—An array object or a null value. The values in the array are of the dynamic data type. An example array of number

values is `[2,3,5,7,11]`. An example array of map values is `[{"hello":"world","number":1},{"hello":"world"}]`.

- **Map**—A map object containing key/value pairs, or a null value. The key is always a string, and the value is the dynamic data type. An example value is `{"hello":"world","primes":[2,3,5,7,11]}`.
- **Binary**—A byte array expressed as a string, or a null value. The string begins with `\u001Bytes:` and ends with a byte array encoded in base 64.
- **Dynamic**—A value that can be any of the above types.

Installing DSA

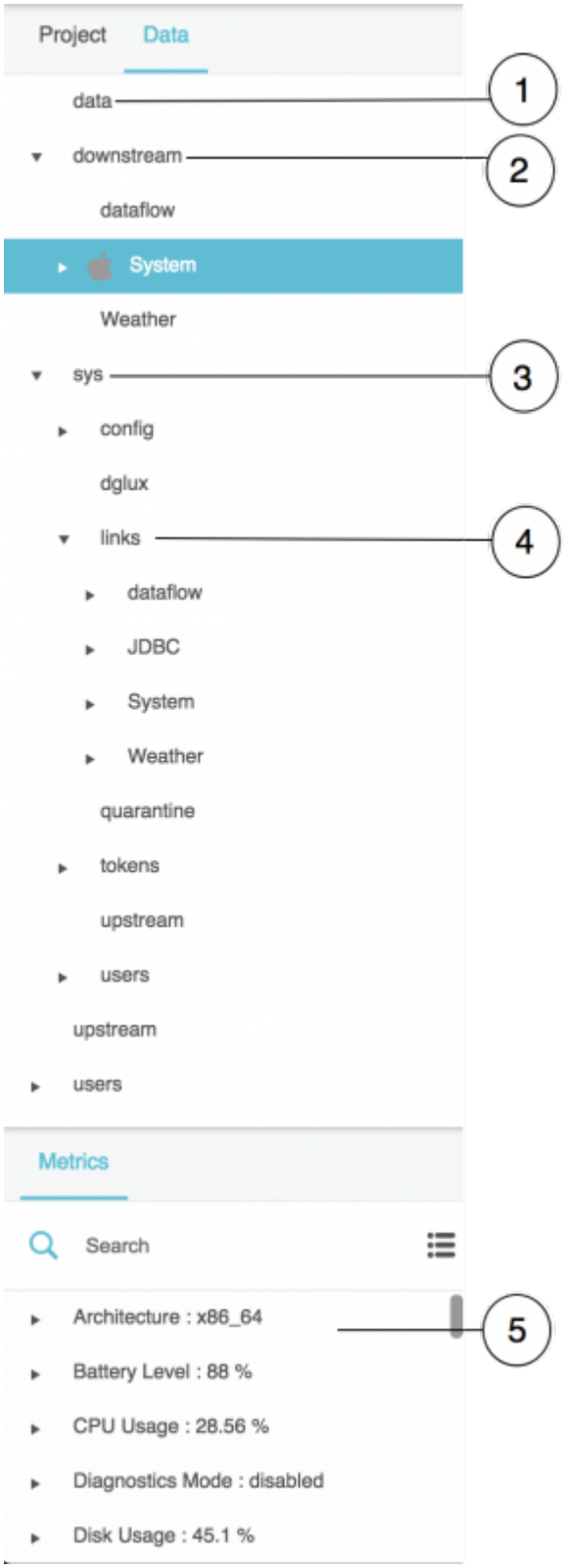
To install DSA on Mac OS X, Microsoft Windows, Linux, or other platforms, follow the steps in the relevant video on this page, depending on your platform.

If you are installing DSA on a Raspberry Pi, BeagleBone Black, or DGBox, you can also follow the steps in this blog post and video.

If you install DSA with DGLux5, and do not have a DGLux5 license, follow the steps to request a license when you are prompted to do so.

Orientation to the DGLux5 Data and Metrics Panels with DSA

The following image shows the Data panel and Metrics panel in DGLux5 when DSA is used.



1	data node	2	downstream node	3	sys node
4	sys > links node	5	metrics at the currently selected node (downstream > System)		

Storing Data on the DSA Server

To store data on the DSA Server:

1. In the Data panel, right-click the data node.
2. Choose **Add** > **Node** or **Add** > **Value**.

   The **Value** option lets you specify a data type.

3. Without moving the mouse pointer away from the pop-up menu, enter a node name, and optionally choose a data type.

   A top-level data node is created.

4. In the Metrics panel, right-click the new data node that you created.
5. Choose **Add** > **Node** or **Add** > **Value** to create a child of this node.
6. Continue adding nodes until all nodes are represented.

To set a value:

1. In the Data panel, select the data node.
2. In the Metrics panel, right-click the metric whose value you want to set.
3. Choose **@set**, enter the value, and click **Invoke**.

To invoke an action on a metric:

1. In the Data panel, select the data node.
2. In the Metrics panel, right-click the metric on which you want to invoke an action.

   A pop-up menu of available data actions is displayed.

3. Choose the action, enter any required information, and click **Invoke**.

Installing and Updating a DSLink

To install a DSLink:

1. In the Data panel, right-click the **sys** > **links** node.
2. Choose **Install Link**, and choose an installation method:
   - If you have a link to a ZIP file, choose **from Url**
   - If you have a ZIP file to upload, choose **from Zip**.
   - If the link can be found in IOT-DSA on Github, choose **from Repository**.
3. Specify a name for the new DSLink.

4. Without moving the mouse pointer away from the pop-up window, click **Invoke**.

   When the link has been successfully installed, a "Success" message appears.

5. To start the link, right-click the link under **sys** > **links** > **[link name]**, and choose **Start Link**.

   When the link has started successfully, it appears under **downstream** > **[link name]**.

To update a DSLink:

1. In the Data panel, right-click the link under **sys** > **links**.
2. Choose **Update** and choose an update type, and then click **Invoke**.

Remote Dataflow

DSA installations feature a dataflow DSLink, also referred to as *remote dataflow*. This tool is similar to DGLux5 dataflow, described here, also known as *local dataflow*.

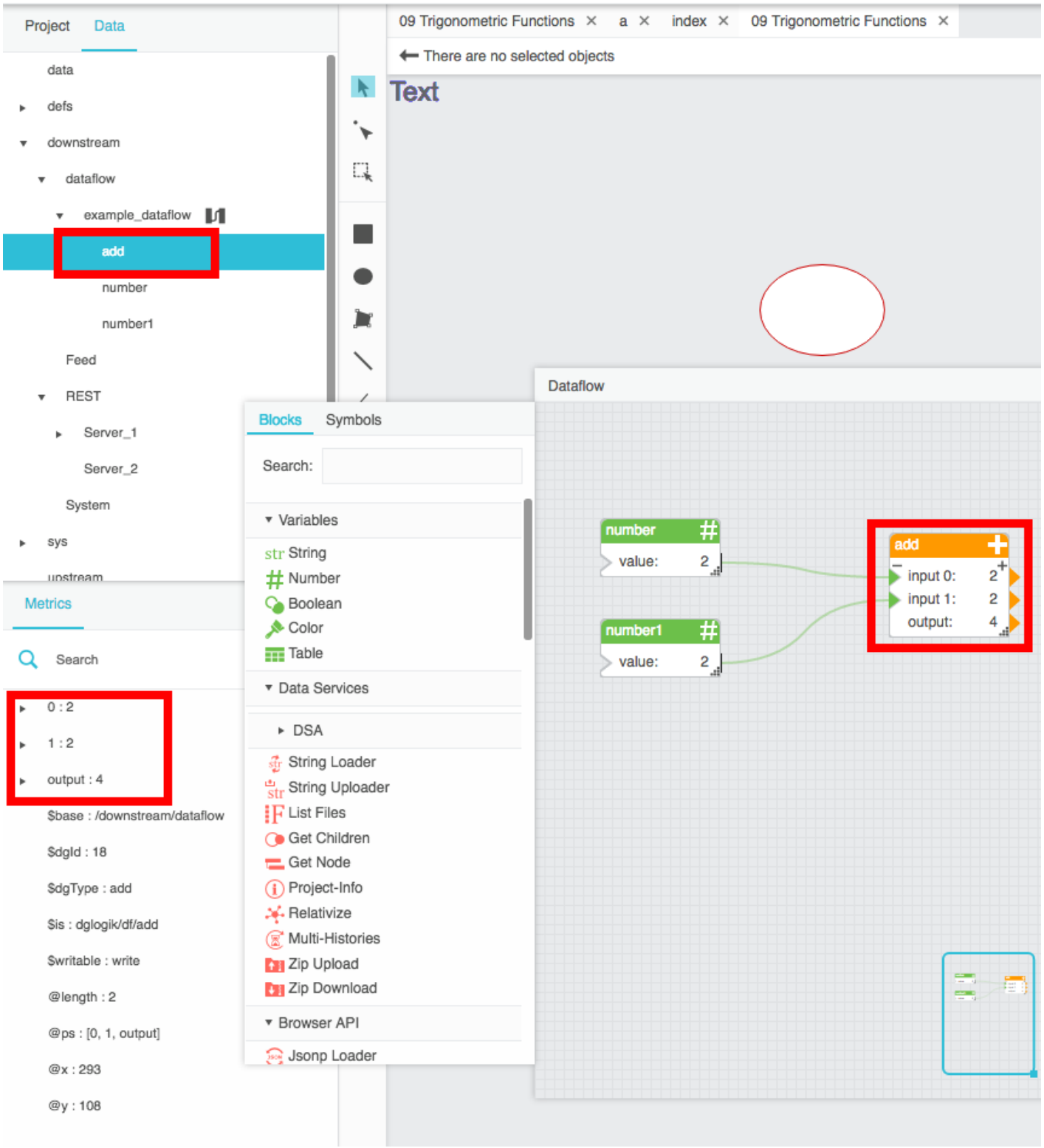
Differences between the two are:

- Remote dataflow is stored on the DSA server with other DSLinks, not in the project files. Therefore, remote dataflow can run even when the project is not open in a browser.
- Certain blocks, such as Browser API blocks, are not available in remote dataflow.

To create a remote dataflow model:

1. In the Data panel, right-click **downstream** > **dataflow**.
2. Choose **Create Dataflow**.
3. Specify a name for the new dataflow model, and click **Invoke**.
4. Choose the new dataflow model at **downstream** > **dataflow** > **[dataflow model name]**.

Block property values in a remote dataflow model appear as metrics in the Metrics panel. They can be used like any other metric. The following image demonstrates an example of remote dataflow metrics.



Security and Permissions

See also: Tokens.

Quarantined Brokers and Links

When *quarantine* is enabled on a broker, any downstream broker or link without an authorized token is held in quarantine. The system can read, subscribe to, and command nodes that are in quarantine, but a node that is in quarantine cannot access other nodes in the system.

Whether quarantine is enabled is specified in `dsa/dglux-server/server.json`.

To remove a node from quarantine, you can *authorize* or *deauthorize* the node. An authorized node is granted access as the permission group that you specify. A deauthorized node is refused access and removed from the system. You can also deauthorize any downstream broker or node that has previously been authorized.

When quarantine is disabled on a broker, any link can connect to the broker without approval.

To authorize a broker or link:

1. Right-click **sys** > **quarantine** > **Authorize**.

   A list of quarantined nodes appears.

2. For **DsId**, choose the node that you want to authorize.
3. For **Group**, choose the permission group that you want to assign.
4. Specify a name, and click **Invoke**.

To deauthorize a broker or link:

1. Right-click **sys** > **quarantine** > **Deauthorize**.

   A list of quarantined nodes appears.

2. For **Name**, choose the node that you want to authorize.
3. Click **Invoke**.

Permission Groups

In DSA, you assign permission groups to users. Permission groups are defined within `dsa/dglux-server/server.json`. Information about permission groups can be found here. Any user with "superuser" enabled has the maximum permission level and does not need a permission group defined.

More information about DSA permissions can be found here.

Users

To create a new user:

1. Right-click **sys** > **users**, and choose **Create User**.
2. Without moving the mouse pointer away from the pop-up window, enter a user name and password, and specify whether the user has superuser access.
3. Click **Invoke**.

To edit a username:

1. Right-click **sys** > **users** > **[username]**, and choose **Rename User**.

To edit a user password:

1. Right-click **sys** > **users** > **[username]**, and choose **Change Password**.
2. Enter the new password, and click **Invoke**.

To edit a user's permission group, landing page, or other user properties:

1. Right-click **sys** > **users** > **[username]**, and choose **Edit User**.
2. Edit the properties, and click **Invoke**.

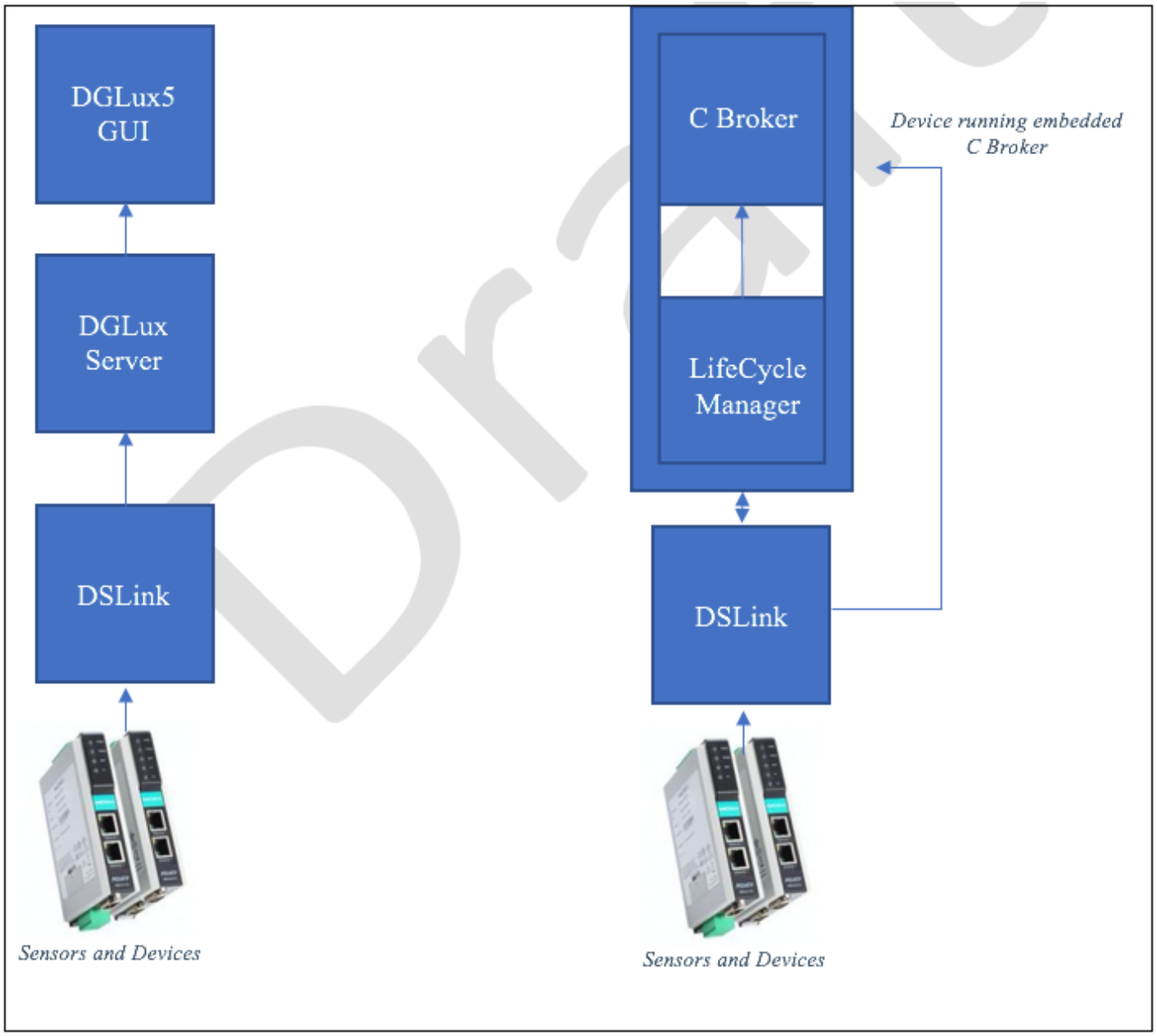
To delete a user:

- Right-click **sys** > **users** > **[username]**, and choose **Remove User**.

The C Broker

The C Broker is a small-footprint version of the DGLux Server (to be precise, the DART Broker). The C Broker is intended for embedding in devices where economy of scale is important. This document tells you how to build, configure and launch the C Broker. This document assumes you are familiar with DSA and DGLux5.

The C Broker receives data from devices by way of DSLinks, in the same way as the DGLux Server. To use the DGLux5 GUI to manage DSLinks that connect to an instance of the C Broker, install and run the LifeCycle Manager as described below. The following figure shows a standard topology on the left and an embedded topology on the right.



Differences between the DSA Server and the C Broker are as follows:

Feature	DGLux Server/DART Broker	C Broker
Built-in web server	Yes	No
Hosts DGLux5 GUI	Yes	No
Manages users and filesystem access	Yes	No

Note: Because and the C Broker support a subset of the features of the DGLux Server, the JSON configuration files for the C Broker are not interchangeable with those of the DGLux Server.

Building the C Broker

To download the source code from GitHub, you must have `git` installed. To build the C Broker, you must have `cmake` 3.0.0 installed These instructions assume you are building on Ubuntu 14.04.

To download the C Broker source code from github, issue the following command:

```
git clone https://github.com/IOT-DSA/sdk-dslink-c
```

To build the C Broker, issue the following commands in the directory where you cloned the source code:

```
mkdir build
sh tools/build.sh
```

Configuring the C Broker

To configure settings for the C Broker, edit the broker.json file, which is created when you build the C Broker. This file must reside in same directory as the executable. The default broker.json file contains the following settings:

```
{
  "http": {
    "enabled": true,
    "host": "0.0.0.0",
    "port": 8080
  },
  "log_level": "info",
  "allowAllLinks": true,
  "maxQueue": 1024,
  "defaultPermission": null,
  "storage": {
    "path": "."
  }
}
```

The following table lists valid C Broker settings.

HTTP Settings	
enabled	To configure the broker to accept HTTP connections, set to true. Default: true
host	Specify the host name of IP address where the broker runs. Default is 0.0.0.0, which means that if the host has multiple IP addresses, the C Broker accepts connections on all of them.
port	Specify the port on which the broker accepts HTTP connections. Default is 8100.
HTTPS Settings	
enabled	To configure the broker to accept HTTPS connections, set to true. Default: true

HTTP Settings	
host	Specify the host name of IP address where the broker runs. Default is 0.0.0.0, which means that if the host has multiple IP addresses, the C Broker accepts connections on all of them.
port	Specify the port on which the broker accepts HTTPS connections. Default is 8463.
certName	Certificate file name (.pem file). Put certificate files in a subdirectory named “certs,” under the directory where the C Broker is running.
certKeyName	Key file name (.pem file). Put key files in a subdirectory named “certs,” under the directory where the C Broker is running.
Other settings	
log_level	Specifies the logging verbosity. Valid values are `fatal`, `error`, `warn`, `info` and `debug`. Default is `info`.
allowAllLinks	Specifies whether the C Broker accepts connections from all DSLinks or only previously-registered ones.
maxQueue	The maximum number of messages from DSLinks that remain queued when the requestor has specified the qos 1 or qos 3 options. The qos (“quality of service”) settings control how the broker caches incoming data to handle disparities between the responsiveness of requestors and that of responders.)
defaultPermission	Specifies the permissions to be assigned to DSLinks that do not have any permissions configured for their associated user. Valid permissions, from least to most restrictive, are `config`, `write`, `read`, `list`, and `none`. (More information...)
path	The directory that the C Broker uses for working storage for qos queue data, incoming data values, and other data that it must store locally.

Specify the defaultPermission setting using JSON notation; for example:

```
  "defaultPermission": [
      [":config","config"],
      [":write","write"],
      [":read","read"],
      [":user","write"],
      ["default","read"]
   ]
```

If you are using tokens to authenticate connections from DSLinks, token files must reside in the “tokens” subdirectory under the directory where the C Broker is running. The name of the token file must be the first 16 characters of the full token string. Specify tokens in JSON format, as follows, one token per file, as follows:

```
{"$$timeRange": null, "$$token":
"47S7ybkxV6ka7Ha6zIHsFl16wiYwu4i8EGYlqdzK7NQw5Rus", "$$managed": false,
"$$group": ":write"}
```

- $$timeRange: A date and time range that specifies the start and end of the period when this token is valid. Example: 2017-03-10T00:00:00/2017-03-11T12:00:00
- $$token: 48-character token
- $$managed: To disconnect and remove all associated DSLinks when the token expires, set to true.
- $$group: The permission group to be assigned to the DSLink that connects using this token.

Deploying the C Broker

To run the C Broker on a machine other than the machine where you built it, copy the broker executable and the broker.json file to the desired location.

To start the broker, invoke its executable on the command line. For example:

```
./broker &
```

To verify that the broker is accepting connections, issue the following command (assuming you have configured the broker to listen on port 8080):

```
telnet localhost 8080
```

Connecting to the C Broker

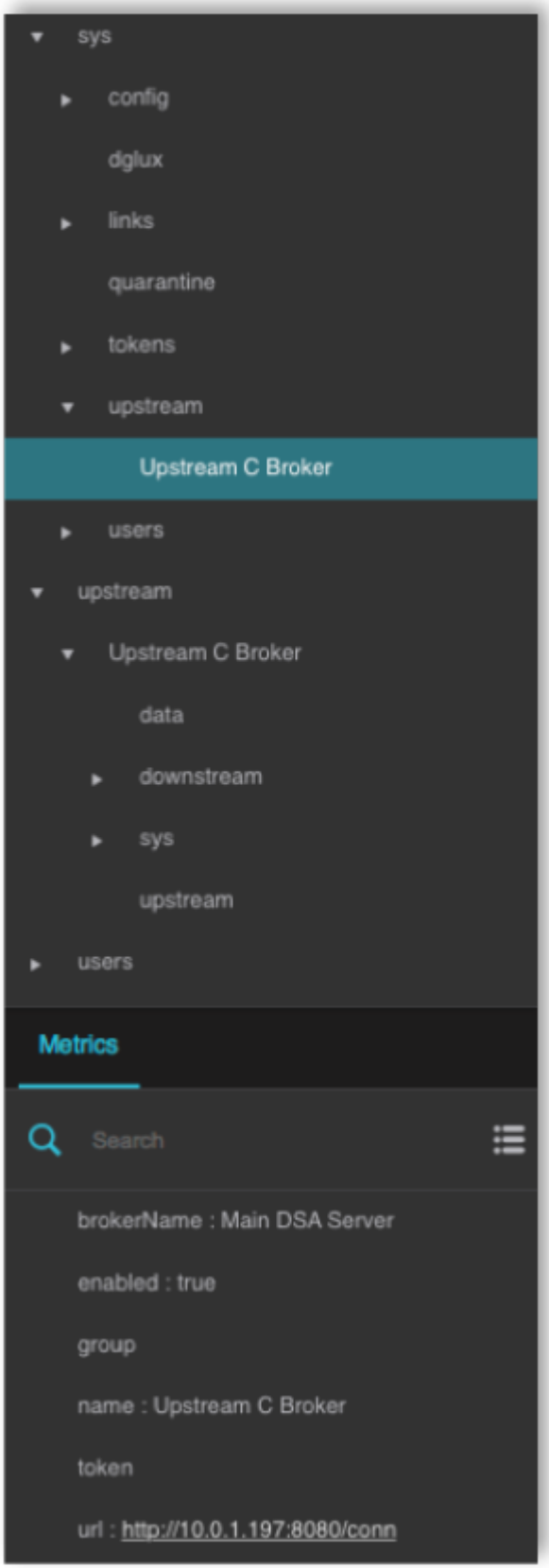
To use DGLux5 to get access to the data that the C Broker receives from its DSLinks, configure a connection as follows:

1. In the **Data** panel, expand the **sys** node and right-click upstream.
2. Choose **Add Upstream Connection** and specify settings,
3. Click **Invoke**.

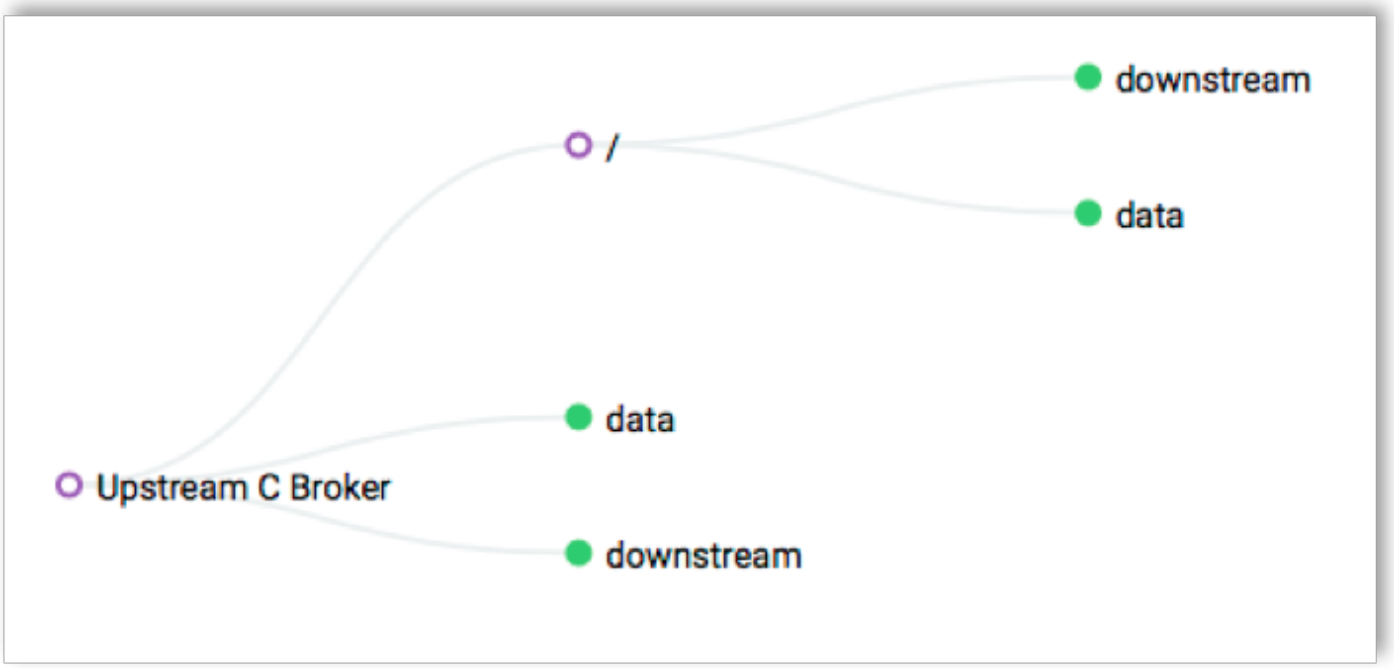
Settings:

- Name: The name to be displayed in DGLux5
- Url: The URL required to connect to the DSA Server. For example: http://101.0.1.197:8080/conn
- Broker Name: The name to be displayed by the upstream broker.
- Token: If required, the name of the token file containing the token needed to authenticate.
- Group: The group defining the permissions to be assigned to the connection.

When the C Broker accepts the connection, it displays a success message. To view the C Broker in DGLux5, expand the **sys > upstream** node and verify that your broker instance is listed.



The figure below shows the topology.



You can configure the C Broker connection to the DSA Server by manually editing configuration files that reside on the machine where the C Broker runs, as follows:

1. In the directory where the C Broker runs, create a subdirectory named “upstream.”
2. In the “upstream” directory, create one JSON file for each upstream connection that you want to define, using whatever file naming convention is meaningful to you. In these files, specify the connection settings in JSON, formatted as follows:

```
{"name": "rnd", "brokerName": "mybroker", "url": "http://rnd.iot-dsa.org",
"token": "<token string>", "enabled": true, "group": ":config"}
```

Settings:

- name: Name of the upstream broker, must be same as the file name in the upstream folder
- brokerName: Name of the current broker to be displayed under the upstream broker
- url: The URL required to connect to the C Broker. For example: http://101.0.1.197:8080/conn
- token: (Optional) Token to be used when the C Broker connects.
- group: The permission group for the C Broker.

The LifeCycle Manager

The C Broker lacks the DSLink-management features of the DGLux Server. The LifeCycle Manager enables you to use DGLux5 to manage the DSLinks that connect to the C Broker.

To build the LifeCycle Manager, your system must have the following software installed. The versions listed below are known to work. The build process and the software itself has not been tested with other versions of these packages.

- g++-5 14.04
- cmake 3.0.0
- libssh2 1.4.3
- zlib 1.2.8
- boost 1.54
- libcurl 7.35
- http-parser 2.1

Building the LifeCycle Manager

Note: The github repository for the Lifecycle Manager is private. If you attempt to clone the source and are denied permission, contact DGLogik to request access to the repository.

To download the LifeCycle source code from github, issue the following command:

```
git clone https://github.com/dglogik/dslink-manager
```

To build the LifeCycle manager, issue the following commands in the directory where you cloned the source code:

```
mkdir build
cd build
git submodule init
git submodule update
cmake -DCMAKE_CXX_COMPILER=g++-5 -DCMAKE_BUILD_TYPE=Release  ..
make
```

To verify that the build succeeded, check that the build directory now contains an executable file named “dsmanager.”

Deploying the LifeCycle Manager

To run the LifeCycle Manager on a machine other than the machine on which it was built, copy the following files from the build machine to the desired production directory:

- dsmanager (executable)
- libsdk_dslink_c.so
- dslink.json (To configure the display name for LifeCycle Manager, edit this file.)

Before starting the LifeCycle Manager, verify that the C Broker is running. To start the LifeCycle manager, invoke its executable on the command line. For example, to run the LifeCycle Manager (assuming the broker is configured to listen on port 8080) in background mode, issue the following command:

```
./dsmanager --broker http://localhost:8080/conn
```

If the LifeCycle Manager started successfully, it displays “Successfully connected to the broker.” To verify that the LifeCycle Manager is accessible using DGLux5, expand the Data pane upstream node. Locate the C Broker, expand its downstream node, and verify that the LifeCycle Manager is listed and is not grayed out. To manage the DSLinks that are connected to the C Broker, right-click the LifeCycle Manager node.

Troubleshooting

Both the C Broker and LifeCycle Manager log all their messages to standard output and do not create any physical logs.