Hello World! Your first Solace app

Hello and welcome to your first Solace app tutorial. Appropriately, we’ll be looking at the sample called “HelloWorld”. At first glance, this application might seem longer than the first Java/Python/etc. program you might have ever written. E.g.:

public static void main(String... args) {

System.out.println(“Hello world!”);

}

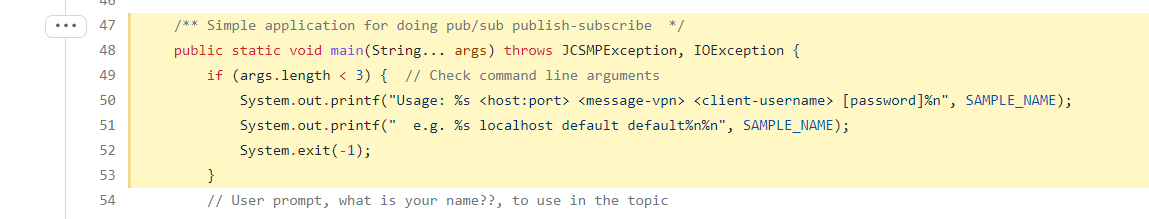
However, as you can tell from this [Wikipedia article](https://en.wikipedia.org/wiki/%22Hello,_World!%22_program), there are many different types of “Hello World” programs. Rather than trying to do the bare minimum to produce some visual output, this Solace Hello World will demonstrate some very fundamental and basic features of Solace APIs and pub/sub messaging:

* **Publish *and* Subscribe**: most Solace applications do both the “I” and “O” in I/O
* **Dynamic topics:** topics are hierarchical and descriptive, not static
* **Wildcard subscriptions:** attract multiple topics via a single subscription
* **Asynchronous message receipt:** use of callback handlers
* **Connection lifecycle management:** connect once, and stay online for the lifetime of the app

This app will connect to a Solace event broker, and publish/subscribe in a loop. Let’s begin!

# Command line arguments

The first couple lines of the program simply read in connection parameters from the terminal:

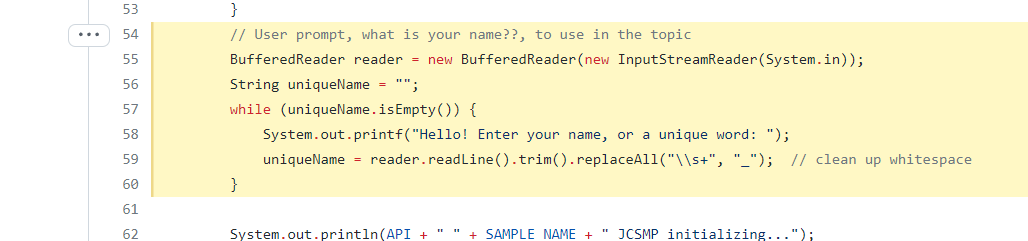
[](https://github.com/SolaceSamples/solace-samples-java-jcsmp/blob/275739cb858cacea5140c5c7c8310cfb50868695/src/main/java/com/solace/samples/jcsmp/HelloWorld.java#L47-L53)

Specifically, for Solace Native (SMF) APIs, we need to know

* Broker / host IP address or hostname
  + E.g. “localhost”, “192.168.42.35”, “mr-abc123.messaging.solace.cloud”
* Message VPN: a virtual partition of a single broker, how Solace support multitenancy
  + E.g. “default”, “lab-vpn”, “cloud-demo-singapore”
* Username
  + E.g. “default”, “testuser”, “solace-cloud-client”
* Password (optional)
  + This is for basic authentication, different if using certificates or single-sign-on

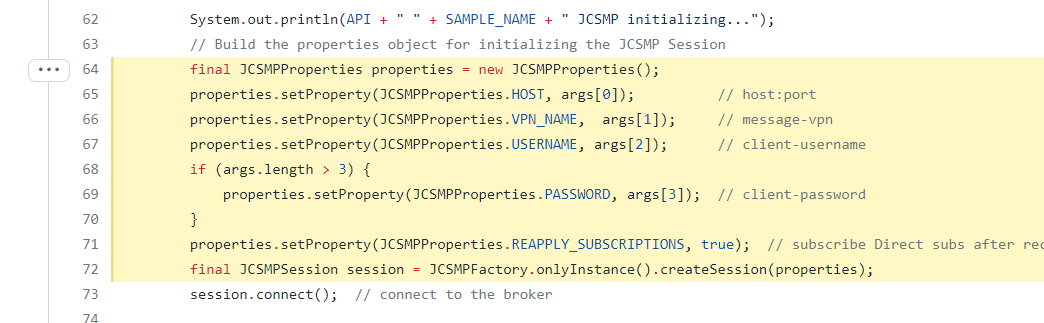
# Enter your name

This part is certainly not required in production applications, but allows Hello World allows to easily build a dynamic topic based on some input. It is also useful if running multiple instances of this application, to see them “talking” to each other, as we’ll see later.

[](https://github.com/SolaceSamples/solace-samples-java-jcsmp/blob/275739cb858cacea5140c5c7c8310cfb50868695/src/main/java/com/solace/samples/jcsmp/HelloWorld.java#L54-L60)

# Connection Properties

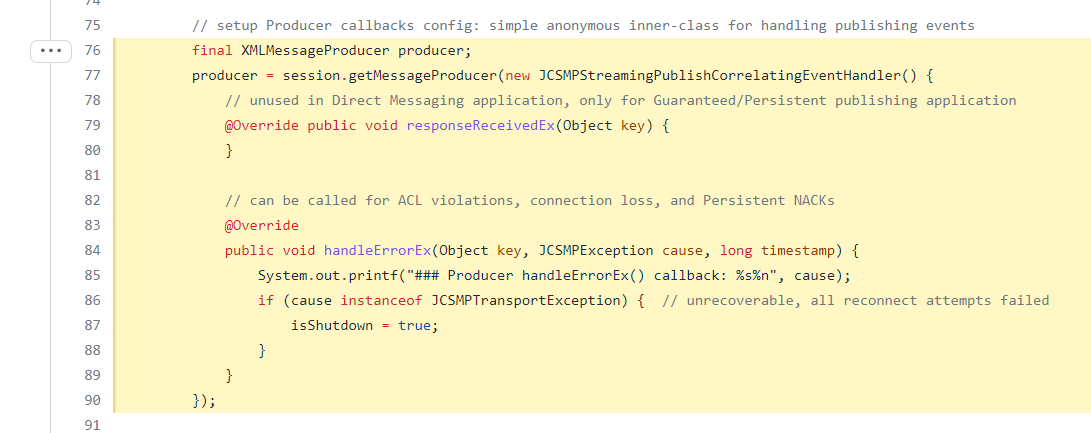
The next few lines of Hello World initialize the connection parameters, as well as a few other properties that might be useful:

[](https://github.com/SolaceSamples/solace-samples-java-jcsmp/blob/275739cb858cacea5140c5c7c8310cfb50868695/src/main/java/com/solace/samples/jcsmp/HelloWorld.java#L64-L72)

The additional property set “reapply subscriptions” is very useful for applications using Direct messaging (e.g. at-most-once delivery): it tells the API that following a reconnection (either due to network flap or broker failover), the API should automatically add any subscriptions back; by default, this set to false and the application would be responsible.

# Setup Producer

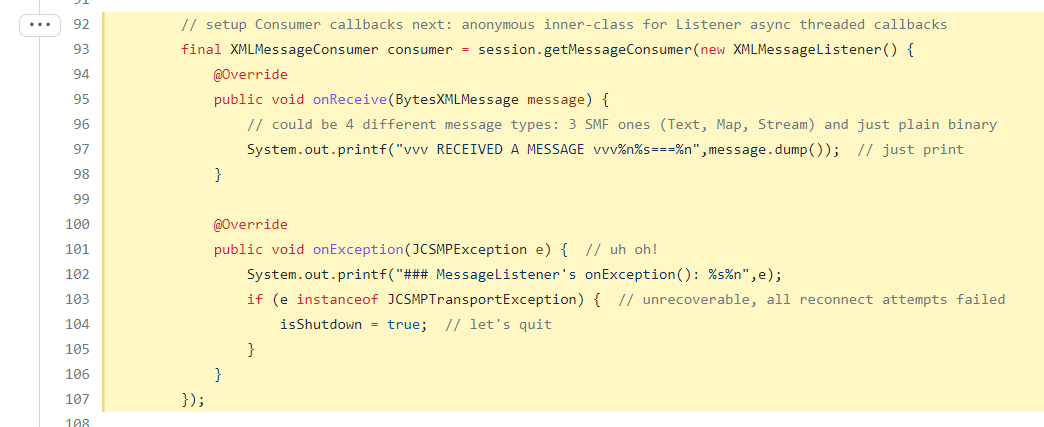
The “producer” or publisher in Solace APIs is the component that sends message to the Solace broker. The producer can be configured to send both Direct and Guaranteed messages on the same session, as well as transactions and other qualities-of-service.

[](https://github.com/SolaceSamples/solace-samples-java-jcsmp/blob/275739cb858cacea5140c5c7c8310cfb50868695/src/main/java/com/solace/samples/jcsmp/HelloWorld.java#L76-L90)

The producer configuration options vary from API to API. In JCSMP, you specify two callback handlers. These are *mostly* used for Guaranteed messaging applications, to receive “ACKs” (successful acknowledgements) and “NACKs” (negative acknowledgements) from the broker. As our Hello Word app uses only Direct messaging, these are not as useful, but still need to be configured regardless. In Python or the PubSub+ Messaging API for Java, Direct publishers do not have to configure this.

# Setup Consumer

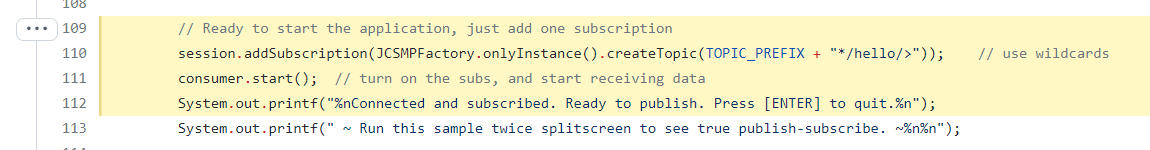
The next part of the sample sets up the ability to receive messages from the broker asynchronously – that is: the application does not have to *poll* the broker for the next message.

[](https://github.com/SolaceSamples/solace-samples-java-jcsmp/blob/275739cb858cacea5140c5c7c8310cfb50868695/src/main/java/com/solace/samples/jcsmp/HelloWorld.java#L92-L107)

As you can see, the “on Receive” callback does not do very much in this simple application, it simply outputs the message to the screen, and continues.

# Add Direct message subscriptions

To receive messages from the broker, you have to tell it what you want to receive. To receive messages via Direct messaging, you add a (topic) subscription:

[](https://github.com/SolaceSamples/solace-samples-java-jcsmp/blob/275739cb858cacea5140c5c7c8310cfb50868695/src/main/java/com/solace/samples/jcsmp/HelloWorld.java#L109-L112)

Notice a few things:

* The topic subscription is hierarchical: there are “/” characters between levels
* The use of “\*” and “>” wildcard characters in the subscription
* Direct subscription (not using Guaranteed delivery yet)

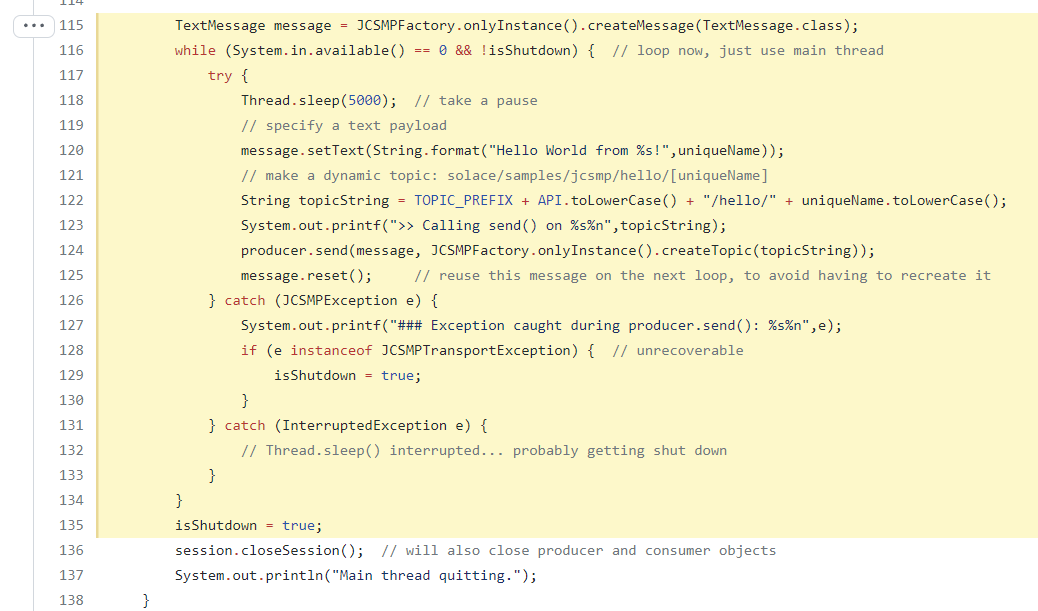
These are called a single-level and multi-level wildcard respectively. The “\*” will match anything up to the next level, including empty-string; for the multi-level, as long as the message’s first part of the topic matches the subscription to that point, the “>” wildcard will match any remaining (one-or-more) levels. See here for more details [topics](https://docs.solace.com/Basics/Understanding-Topics.htm) and on [wildcards](https://docs.solace.com/Basics/Wildcard-Charaters-Topic-Subs.htm).

So, our HelloWorld app is adding a subscription: solace/samples/\*/hello/>

After adding the only one subscription (you can add as many as you’d like, within the limits of the broker), start the Consumer object which tells the broker to start to receive messages.

# Publish and receive messages

Now we are ready to send some messages, and the subscription will allow us to receive those messages back. So in a loop, wait for some user input, and publish a message every 5 seconds:

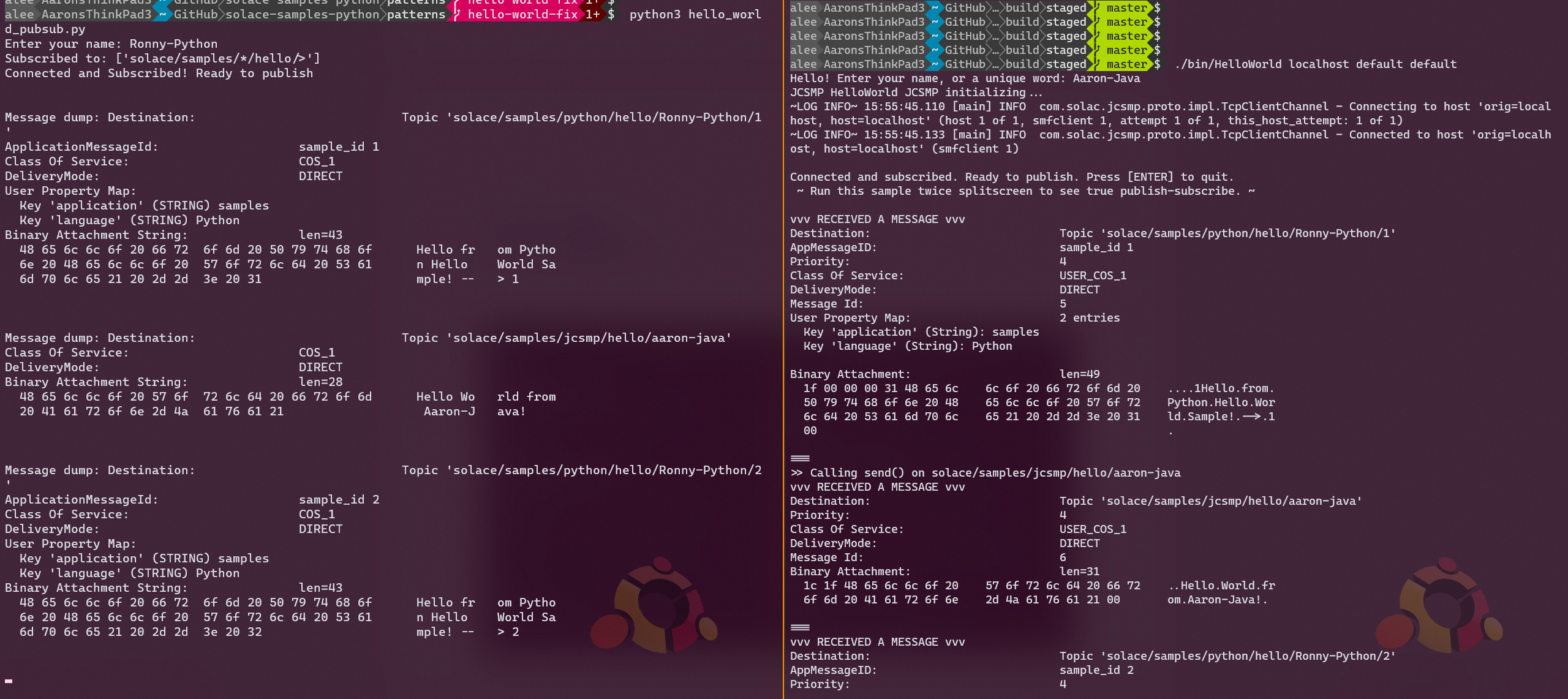
[](https://github.com/SolaceSamples/solace-samples-java-jcsmp/blob/275739cb858cacea5140c5c7c8310cfb50868695/src/main/java/com/solace/samples/jcsmp/HelloWorld.java#L115-L135)

Note that we specify the payload each loop (could be a text String, or a binary byte[] payload, etc.), as well as define what the message’s published topic is. Recall: topics are not configured in the Solace broker, they are metadata of the message, and a *pattern matching* (via subscriptions) is done on each received message.

# Run it again!

Running this application once is ok to ensure you have broker connectivity, but event-driven technologies like Solace are all about decoupled architectures: you need multiple applications to communicate asynchronously. So split your terminal, or get a second screen, and try running it again.

Note that you could run a different language for your 2nd instance, or even another protocol that Solace supports (e.g. REST, MQTT, AMQP 1.0). Just ensure your topics match the subscriptions.



Here is the Python HelloWorld app talking to the Java JCSMP HelloWorld app. Both are subscribed using wildcards, one subscription each, and they will match topics published by other API flavours: note the published topic for each is different (more for demonstration purposes than anything).

# What’s Next?

Got a handle on how a basic Solace native app sends and receives messages? Wondering what the next step is?

* Higher performance publish-subscribe using Topics
* Message payload transformation using *Processor* pattern
* Request-Reply using blocking call (not everything needs to be asynchronous)
* Guaranteed messaging publish-subscribe

­­­­­­

Direct Publisher Subscriber

This tutorial assumes you’ve already reviewed the basic concepts found in the Hello World tutorial, and will go into a bit more detail on the

Note that these samples are using *Direct Messaging*, Solace’s “at-most-once” or “best-effort” quality of service. This is in contrast with *Guaranteed Messaging*, or persistent messaging, which offers a high-quality service in terms of reliability and guarantees against message loss if used correctly.

Using Direct Messaging is great for:

* Introductory applications like this one
* High performance applications where every millisecond (or even *micro*second) count!
* Applications that can tolerate some message loss in failure scenarios:
  + Sequence numbers to detect gaps
  + Local caches available to resend data
  + Internet of Things (IoT) applications where data is constantly being published by, say, a temperature sensor or a connected car
  + Etc.

In the Hello World application, it both published and subscribed to messages.