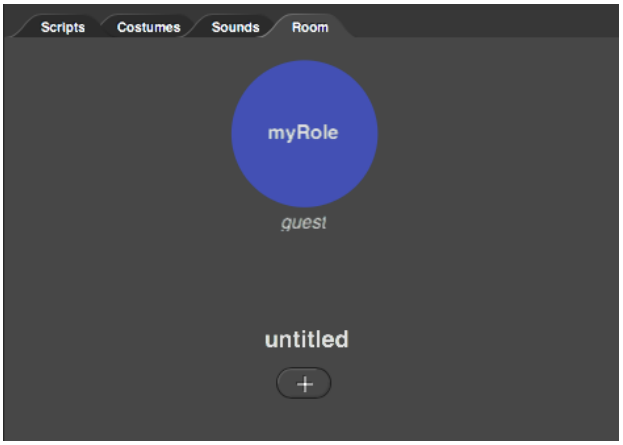
NetsBlox Lesson: Introduction to Messages
Chatroom

Remote Procedure Calls (RPC) provide one way to write a distributed program. A program using an RPC is distributed because part of it runs on your computer (inside your web browser), but the actual RPC runs on a different computer, the NetsBlox server. In fact, many of those RPCs will then request data from other sources, for example, Google provides mapping data. That means that even more computers are involved in a simple RPC call.

Today we'll look at another way of creating a distributed program using *message passing*, a technique widely used to share data between program running on different computers. Message passing enables you to send data to another NetsBlox program running on a different computer. In some sense, messages are similar to events we have seen before (remember the `broadcast` and the `When I receive blocks`?). There are two differences though: messages can contain data and they can be sent to NetsBlox programs running on different computers not just different sprites within the same program.

When you send an email to somebody, what is it that you must know? That's right, the email address. The address uniquely identifies your recipient: no two people can have the same address. Just like with phone numbers, postal mailing addresses or website url-s, you need a globally unique identifier. That works the same way with NetsBlox. If we want to send a message to another program, we have to make sure that we can uniquely identify our recipient. To understand how NetsBlox message addressing works, we need to introduce a new concept.

You might have seen the Room tab next to Scripts, Costumes and Sounds. If we select it, you'll see something like this:



Every NetsBlox project has a single Room that has one or more Roles. The Role is the concept NetsBlox uses to allow multiple subprojects within one project. A single NetsBlox project can have multiple Roles, that is, multiple subprojects, each of which can run on separate computers. For example, if we create a multi-play game like Tic Tac Toe, Chess or Battlefield,

we need two players. Each of these players is associated with a Role. For example, Tic Tac Toe would have two Roles named X and O, while chess might use the names black and white for its two Roles. A poker game would typically have more than two Roles, one for each of the players participating. Each of these Roles has its own sprites, costumes, scripts, etc. In other words, a Room is a way to group together NetsBlox projects that jointly implement a multi-player game or other distributed program.

Back to messages and the problem we raised: how to identify the recipient of a message? Or in computer networking terminology: how to address a message? In NetsBlox, the addressee of any message is a Role.

In fact, it is really easy to send messages to Roles within the same Room. Let's create our first messaging application! When you start learning a new computer language, the first program you write is typically one that prints "Hello World" on the screen. Our first messaging app is going to be a distributed version of Hello World. One Role will send the text "Hello World" to another Role, which, in turn, will print it on the stage.

Distributed Hello World Project

Let's create a new project, go to the Network tab, drag in the rectangular **`send msg`** block, and select the only option from the pull-down menu in the middle, called message. This is what you should see:

Messages have different types. The message type consists of a name and the list of fields, that is, the list of data items the message carries. NetsBlox comes with one built-in message type called **`message`** and that is what we picked from the menu. It has one field called **`msg`**. For this app, it will work just fine. Later we'll see how to create new message types.

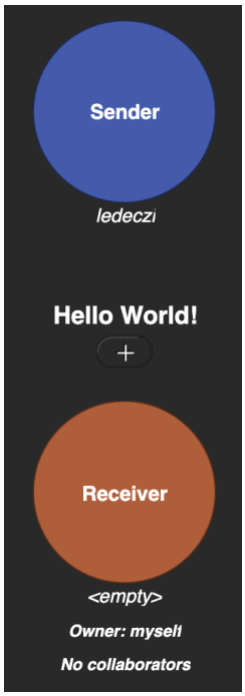
Back to our **`send msg`** block. The only data field this message type has is called **`msg`** and this is where we will type in our message: Hello World!

The rightmost pull-down menu is where we will put the address of the message. Since we have not created another Role, let's leave it alone for the time being.

Instead, let's create a second Role, the one that will receive our message. Click on the Room tab. We see the single Role called "myRole" and underneath it the name of the project called "untitled". Let's click on the Role name (make sure it is the name and not the blue background). Let's type the new name: "Sender". This Role will be used to send a message to the other Role.

We can also click the name untitled and type in a new name for our project. I'll use "Hello World!"

Now let's create that second Role, shall we? Click the plus sign in the middle and type in the name "Receiver". This is what you should see (except your username should show up, not mine, ledeczi).

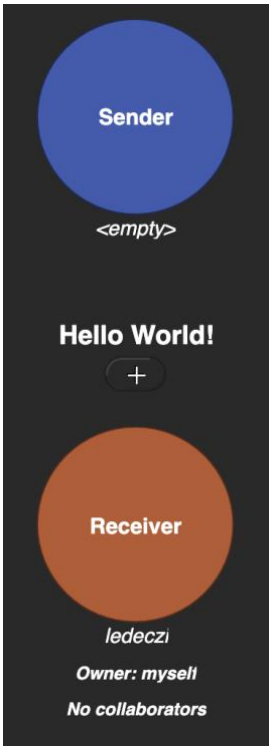


We have two Roles: Sender and Receiver. The name of the project is "Hello World!" We are occupying, that is, working with, the Role Sender and currently nobody is occupying the Role Receiver (indicated by the text <empty> under the brownish circle). Finally, we own this project and there are no collaborators.

NetsBlox allows multiple users to work on a project together, that is, to collaborate, very much like how Google Docs work. We won't go into that today.

Ready to write the code for the Receiver Role? Click the brownish background of the circle and select the "Move to" option. We are asking NetsBlox to switch Roles for us. When you switch Roles, any unsaved changes are lost, so NetsBlox asks you to Save your Role and you should click Yes.

You may or may not have noticed that only a small change occurred on the screen. The Sender Role is now showing <empty> and the Receiver Role has your username under it. Here is what I see:



Let's click on the Scripts tab. There is nothing there?! It should not be surprising though: we have created a brand-new Role and just like a new project, it starts out blank. A new Role is a new subproject. We have to create the sprites and write the scripts for them.

So, let's do just that! Go to the Network tab and drag in the `when I receive block` and select the only option in the pull-down menu, **`message`**.

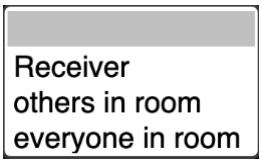
As you might have guessed, the pull-down menu lists all the available message types. Since we have not created any, it only shows the single built-in message type, called appropriately enough, **`message`**. And its only data field, called **`msg`**, shows up in the block as a variable!

All we have to do now is display it by using a `say` block (under the Looks tab). Simply drag the **`msg`** variable from the `when I receive block` and drop it in the `say` block. This is the entire script of the Receive Role:

Let's save our work. Notice that in the File menu, now it says Save Role as opposed to just Save. Once you have multiple Roles, you can only save the one you are working on.

Our Receive Role is ready, but we left one more thing to add in our Sender Role. So, let's go back. Click the Room tab and click the blue background of the Sender Role and pick the "Move to" option. If you have not saved the Receiver Role, click Yes when NetsBlox offers you to save the Role before switching.

Click the Scripts tab and let's check out the `send msg` block we added previously. We left the address blank. Click the pull-down menu and this is what you should see:

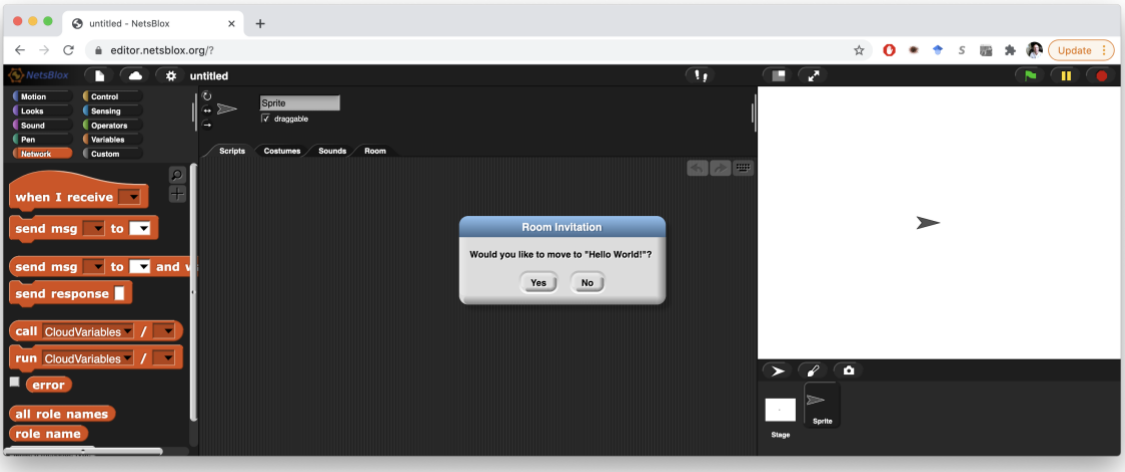


The first option is the only other Role in the project, Receiver. The "others in room" option means every other Role. Since there is only one other Role, this is equivalent to picking the Receiver Role in our case. The "everyone in room" option means every Role including the sender itself. Let's pick the Receiver option.

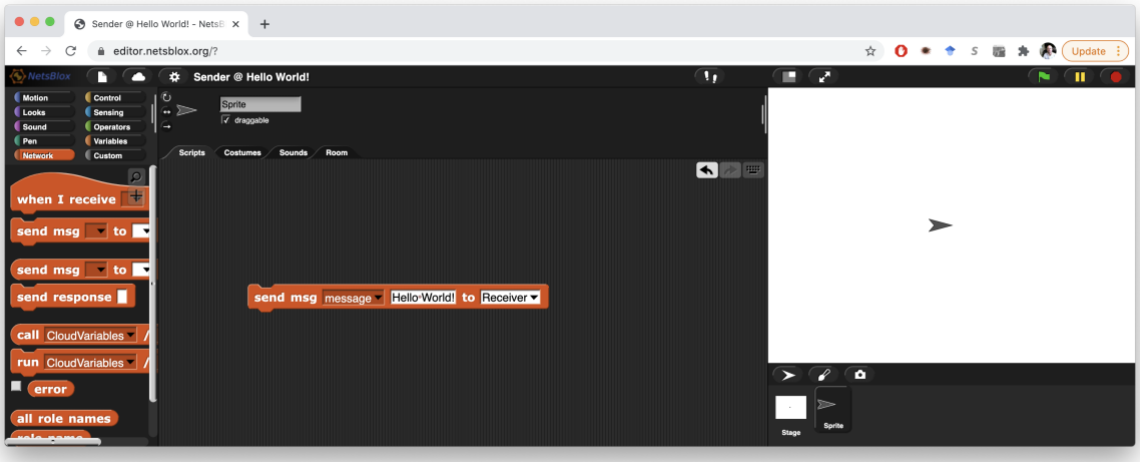
We are ready with our distributed Hello World app! It only took a grand total of three blocks of code! The question is: how can we test it? Don't we need two computers? It turns out that you can test your multi-role distributed NetsBlox programs in separate browser windows/tabs. Browser tabs are isolated from each other, so it is actually a perfectly valid test.

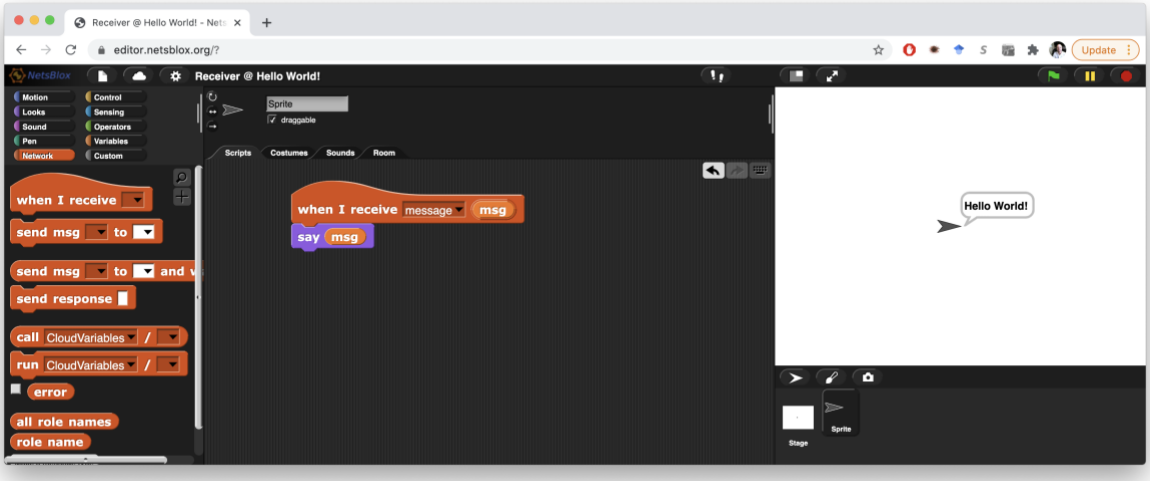
So, let's create a new browser tab and open NetsBlox in it as well. Position the two browser windows so that you can see them both on your screen. (If the browser tabs live in the same window, only the one that is visible is active. That means that a NetsBlox project will not run if another tab in the same window is on top.) In the original browser where we have the Hello World app open, go to the Room tab and click on the background of the receiver Role:

Select the Invite User option. The new window that pops up shows all currently active NetsBlox users, but the very first one in the list says: myself. Click that. In the second browser window, you should see an invitation to join the Hello World project:



Click Yes and you will find yourself in the receiver Role of the Hello World project once it is loaded. In the first browser tab, click the `send msg` block. You should see the sprite in the second browser window say Hello World like this:



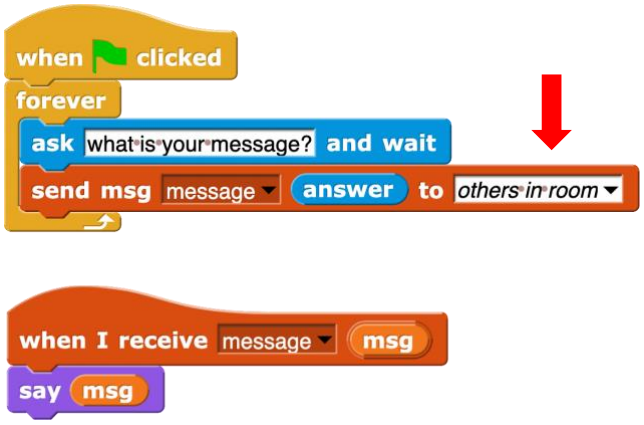


If you want to experiment a little bit, change the hello World text in the `send msg` block to say something else, like “NetsBlox is really cool!!!” Click the block again and you should see the text pop up on the other side.

So, our first distributed application that uses message passing is complete! You may think that is fairly limited. Well, then let’s turn it into a real texting app, shall we?

Texting Project

The idea is to keep asking the user to type in a message and then sending it to every other Role in the project. Let’s create a new blank project. Let’s add the sending code like before, but also the `when I receive msg` block so that every Role will be both a sender and a receiver just like in a normal texting program. This is how the script should look like:



Notice how we used the “others in room” address in the `send msg` block this time. This way, every Role can have the exact same script since we are not using specific Role names!

Now let’s create one or more additional Roles. Go to the Room tab, rename the current Role to something like “Person 1” by clicking on the original name “myRole.” You can also rename the project from untitled to something more meaningful. Here comes the trick: click on the blue background of the Role and select Duplicate. This will make an identical copy of the Role. Feel free to rename the new Role to something like “Person 2.” If you switch to this Role, you’ll see that the scripts are all the same as in the original Role.

Let’s test the project the same way we did the Hello World one. Create an additional browser window, open NetsBlox and invite “myself” to the currently empty Role (Person 1 or Person 2). Click the green flag in both browser windows and you can send message back and forth between the two NetsBlox windows.

If you have two computers handy, you can log in on both, invite yourself from one and you can use the app to send messages across the computers.

You can also invite your friends to use the app. They need to open NetsBlox on their computer, log in, wait for your invite and then accept it. You can create additional Roles by duplicating one and inviting multiple friends. In that case, the current downside of our implementation is that you do not know who sent which message. While we do not have time to fix that problem now, you can do it on your own. (Hint: you need a new message type and everybody needs to send their name and the text message in a single message). We'll do something similar in the next project, the Chatroom.

If you did not quite finish your project, here is mine:
https://editor.netsblox.org/?action=present&Username=ledeczi&ProjectName=VDN-Texting&

Chatroom Project

There are a few shortcomings of our previous approach. We can keep adding multiple Roles, but it will always be a fixed number. Even if we add 10 Roles, we cannot accommodate 11 people. Also, we need to invite our friends one by one. It would be much better if any number of users could join the group on their own.

This is the goal in this Chatroom project. For that, we will need to create two programs: a Chatroom server and a Chatroom client. There will be single instance of the server program running on one computer. Its role is to 1) accept connections form clients, 2) receive their text messages and 3) resend them to every other client. Each user who wants to participate in the Chatroom needs to run a client that 1) connects to the server, 2) sends text messages to it, 3) accepts messages form the server, and 4) displays them on its screen.

For this to work, we need to learn another way of message addressing in NetsBlox. It turns out that messages can be sent to any role of any project of any user by using a "global address." You can think of what we did in the previous project as "local addressing:" we were able to send messages to any Role in our own projects. Here we'll see how we can send messages to any running NetsBlox program of any user. We need a global address that is unique: just like phone numbers that consist of country code, area code and then the local number making them globally unique, we need a similar approach here.

A Role of a NetsBlox program can be uniquely identified by the username, the project name and the role name trio. User names are unique. Each user must have uniquely named projects. And each project must have unique names for its Roles. Hence, a fully qualified address in NetsBlox can be defined as

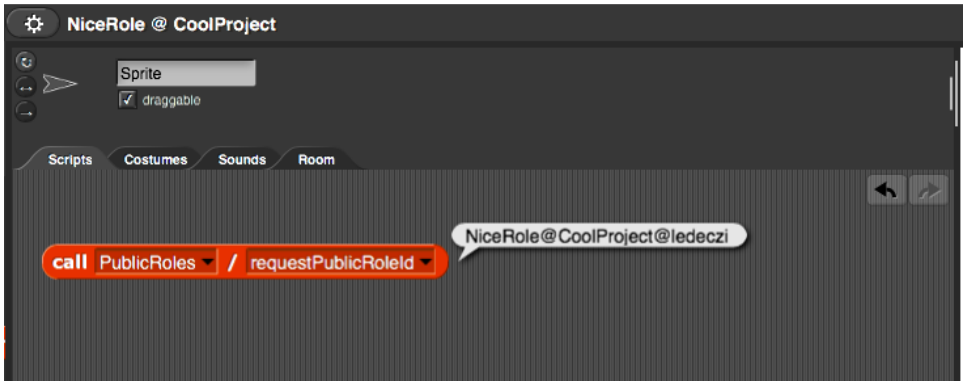
rolename@projectname@username

and it is guaranteed to be globally unique. For example, if I name the project CoolProject and we name the Role NiceRole then the address other programs can use to send me a message would be

NiceRole@CoolProject@ledeczi

You can change the name of the project and the Role by clicking on their current names under the Room tab. Since we are using Role names for addressing, a NetsBlox address like this is called “public role id” because it identifies a Role and if someone knows it, they can send a message to you.

To help you out a bit, there is a Service called “PublicRoles” with a single RPC called “requestPublicRoleId” that will return the public Role ID of the current Role of your own project, that is, your own address. See below:



Notice how the Role and Room names are shown in the top bar of the NetsBlox user interface.

If I wanted to send a message to this Role from any other program, this is what I would need to do:

We need a variable to store the address because you cannot type directly in the **`send msg`** block.

Let’s talk about message types now. The **`send message`** block, before we add anything to it, looks like this:

The first pull down menu allows you to select the message type. What makes a new message type? Just like custom blocks and RPCs, messages can have various inputs. NetsBlox comes with a built-in message type called **`message`** which has a single data input (or payload) called **`msg`**. When you select that message type, the block reconfigures and shows the required inputs:

That is exactly the message type we used above by typing in “Hello there!” and then dragging in the **`server`** variable into the address input placeholder (the last pull-down menu).

Back to the Chatroom. Now that we know how to send messages to any other project, we understand how the clients will send their messages to the server. There is only one server and all clients will have to know its address. Just like a phone number or a url, you need to know that information to contact the person or site identified by them.

But how does the server know the address, that is, the public role id of the clients? Well, they have to let the server know. It means that first a client needs to send a message to the server with its own public role id. The server will store the address of every client in a list. Then when a client sends a regular chat message, the server will send it to every item of the list, that is, every connected client. Notice that we have just designed our first *communication protocol*:

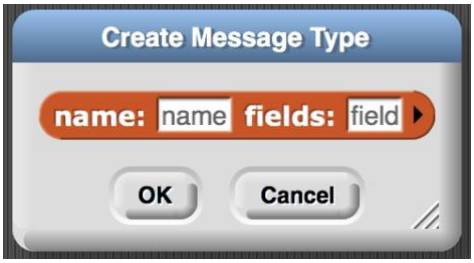
1. Client sends its own address to server
2. Client sends chat message to server
3. Server forwards said message to every client
4. Go to step #2

The content and order of these messages are important. For example, we should not send a chat message before we send our address to the server because we will not get any chat messages from anybody else until we let the server know our own address.

Since the content of the first message and the subsequent messages are different (first we send our address then we send chat messages), we need to define separate message types for them. It is the message type that tells the server what kind of message it is according to the protocol and what data payload it has.

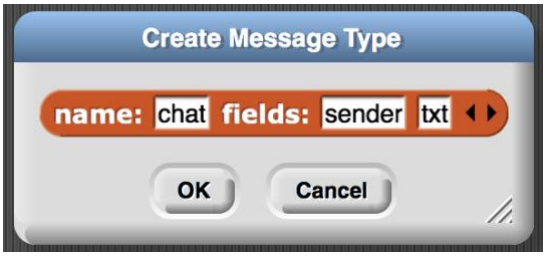
For this application, we define a **`connect`** message with an **`address`** field and a **`chat`** message with a **`sender`** and **`txt`** fields. The sender will contain our name so that the recipients will know who sent the given message.

How do we define new message types in NetsBlox? In the Network tab there is a gray button called **`Make a message type`**. When you click it, this window pops up:

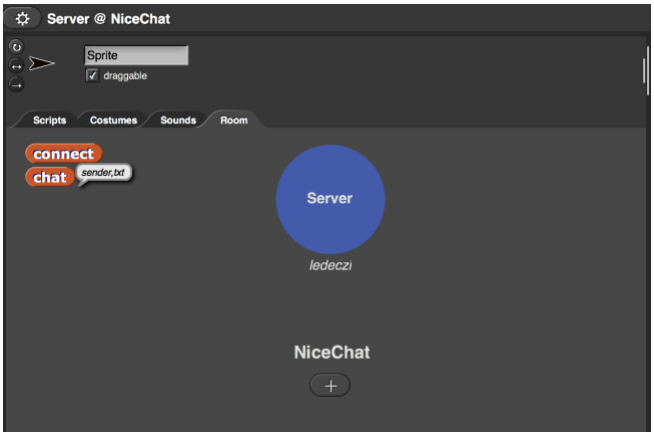


We type in “connect” for the name and “address” for the field. Note that capitalization matters, so *Connect* and *connect* are different names!

For the second message type, we click the arrow to get a second field and type in the names like so:

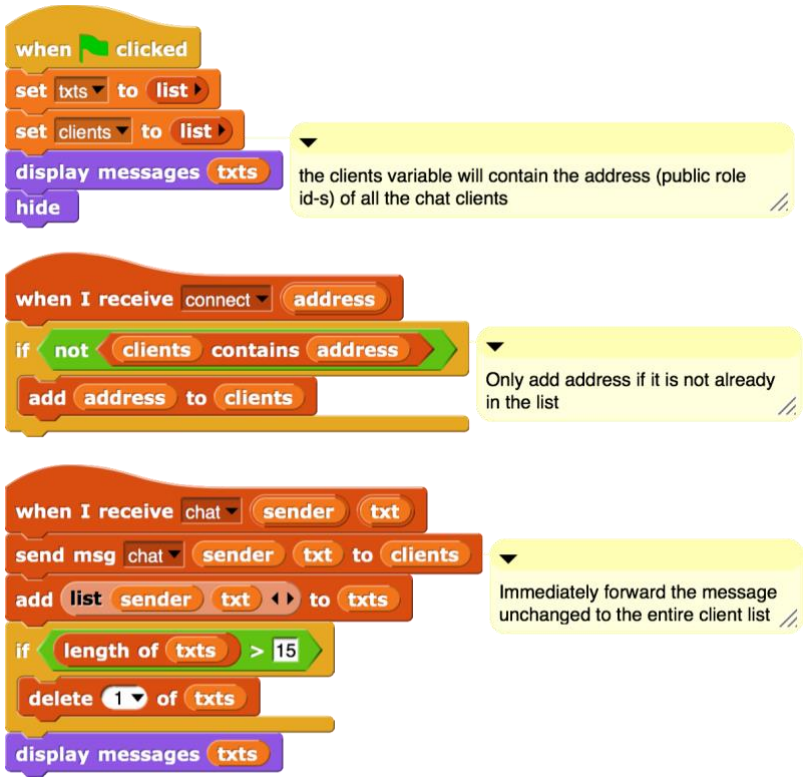


Once we defined these messages, we can actually see them under the Room tab. Check out the orange blocks, `connect` and `chat`, near the top left corner. If you click on them, they show their fields too:



When trying to communicate between NetsBlox programs, it is very important that both the sender and the receiver have the exact same message type definitions. If a single character is different, the receiver will never get the message.

We are ready to create our programs now. I have already created the server. Since only I will run it, you do not need to bother making it. Let’s just take a quick look at how it works:



When the green flag is clicked, we initialize two variables: `txts` stores the last 15 chat messages along with their senders, while `clients` stores the addresses (public role id-s) of all connected clients. Both are initialized to empty lists. The initial call to the `display messages` custom block just clears the screen and displays the message: "Waiting for messages…" at the bottom of the stage. The custom block displays the last few messages breaking up longer text into multiple lines. Feel free to look at the implementation (right click, edit), but it is not necessary to understand how the server works.

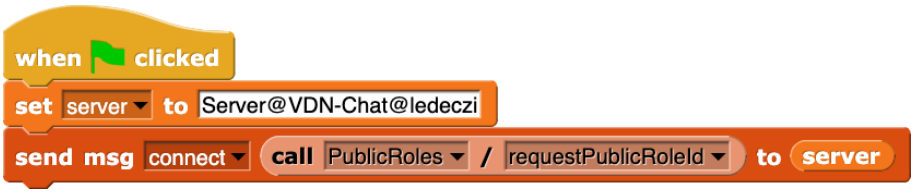
The `when I receive connect` script checks whether the just received address (public role id) is already in the list and if not, it adds it.

The `when I receive chat message` immediately forwards the message to the client list. The NetsBlox `send msg` block is smart enough so that if it gets a list of addresses, it sends the message to each of the items of the list. The rest of the code is pretty simple: we add the sender and the message text to the list of messages and display them. The code also prevents the `txts` list to become longer than 15.

As you can see, the server program is not terribly complicated. What about the client? Let's create it now!

If you have not defined the new messages types yet, create a new project and follow steps described in the previous page. Make sure you have the names and fields exactly as specified because they have to match the ones the server is using!

The first thing the client needs to do is register its own address (public role id) with the server. That is, it needs to send a `connect` message to the server:



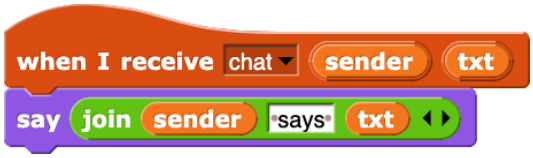
The server's address is "Server@VDN-Chat@ledeczi" because the Role name is "Server", the project name is "VDN-Chat" and the user name of the person running the server is "ledeczi." You need to create a variable to store the address and drag it in the `send msg` block because you are not allowed to type directly in that slot. The `address` data field of the `connect` message type needs to be the client's own address, that is, public role id. But you do not need to manually type that in. Simply use the `call` block (Network tab), select the `PublicRoles` service and the `requestPublicRoleId` RPC to have NetsBlox provide it for you.

That's it. If you click the green flag, you will register with the server (if it is already running).

Let's do the chatting part now, shall we? That is going to be very familiar from the first part of this lesson:

In a forever loop, we wait for the user to type in a message and send a chat message with our name (Akos in my case) as the sender and the provided answer as the message itself.

The only thing missing is handling the chat messages that the server will send us. That is pretty simple as well:



We use the `join` block (Operators tab) to include the sender of the message. Notice that we do not need to display our own messages separately; we will receive them from the server just like everybody else's messages since we are on the client list too! That is, the server will send us our own messages as well.

Here are the links to these two projects:

Server: https://editor.netsblox.org/?action=present&Username=ledeczi&ProjectName=VDN-Chat

Client: https://editor.netsblox.org/?action=present&Username=ledeczi&ProjectName=VDN-Client

If you decide to run the server yourself, do not forget to change the client so that it uses the new address for the server. The username portion will no longer be "ledeczi," but your own username.

An optional extension is to export the `display messages` block from the server program, import it in your client and use it to display the last 15 or so messages like the server does as opposed to "saying" just the very last one. We leave this as an exercise.