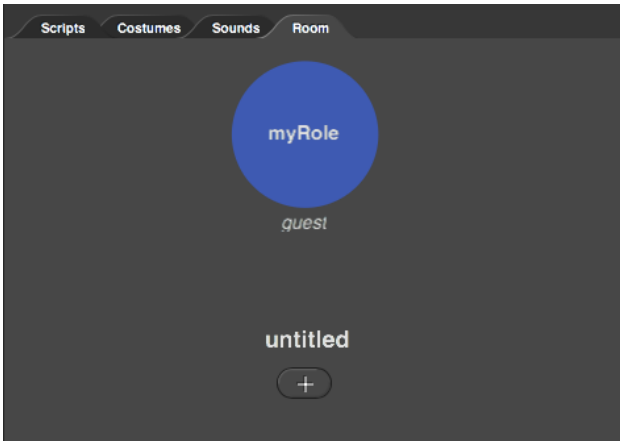
NetsBlox Lesson: Introduction to Messages
Chat Room

Remote Procedure Calls (RPC) was 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 RPC is executed 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 mens 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. Message passing enables you to send a message 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 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 target. To understand how NetsBlox message addressing works, we need to introduce a new concept.

You might have seen the, 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. A single NetsBlox project can have 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, you 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 together implement a multi-player game or other distributed program.

In this lesson, just like all previous lessons, we'll stick with a single Role. But Roles play an important role (no pun intended) in message addressing. A NetsBlox program can be uniquely identified by the user's name, 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 is defined as

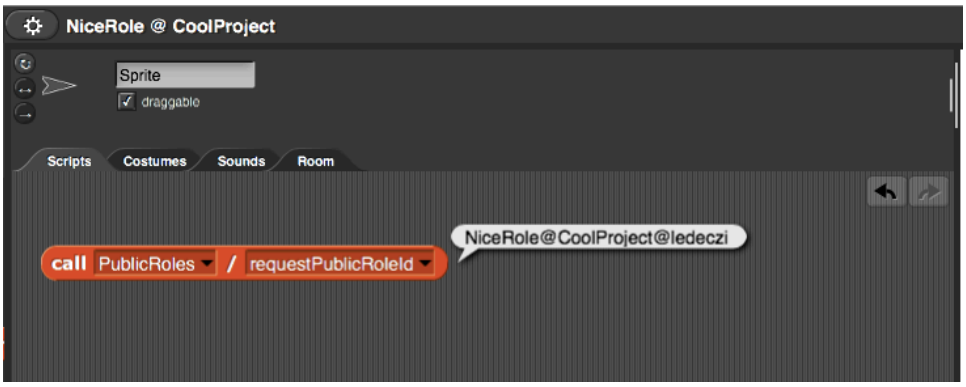
rolename@projectname@username

and it is guaranteed to be unique. For example, if I name the Room (which must have the same name as 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 Room and the Role by clicking on their current names. 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. 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. 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. Just like custom blocks and RPCs, you can specify what inputs a message can have. For that, however, you need to specify a message type. 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). Why is it a pull down menu? Because there is another way for addressing in NetsBlox: within the same Room you can use the Role names to send messages. But that is for a later lesson. For now, we'll stick with public role IDs.

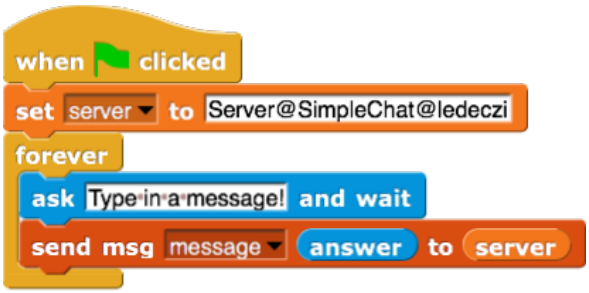
Chat Room project

After this long introduction, let's create our first program, a simple chat room.

We'll need to write two separate NetsBlox programs: one that manages the chat room (we'll call it the chat room server) and another that sends messages to the chat room (let's call it client). In class, I create the server and it runs on my computer connected to the projector, while each students creates their own client program.

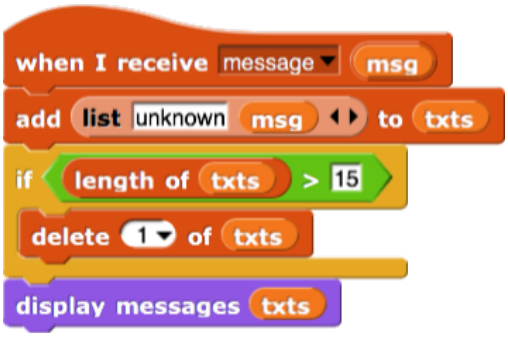
Let's start with the server. It is simply waiting for messages and displays them on the stage.

Pretty simple, huh? The client is not too complicated either:



The program keeps asking for input from the user and sends it to the address: Server@SimpleChat@ledeczi which corresponds to the chat room server program's Role and Room names.

We can make the server use some fancy display as opposed to just using the say block so that we can see multiple messages on the screen. Do not worry about the **`display messages`** custom block though (it is complicated because it breaks up long messages so that they fit on the screen):



The **`txts`** variable stores the last 15 messages along with the sender (for now, it is unknown, but the next version of the program will make good use of it). The **`txts`** variable is a list and each of its items is a list also: it stores the sender as its first item and the message as its second.

When we receive a message, we create a new list with the word 'unknown" and the **`msg`** variable, the received text as its two elements and we add it to the end of the list stored in the **`txts`** variable. If the length of **`txts`** goes beyond 15, we delete its first element to make sure that we fit on the screen.

Here is the server:
https://editor.netsblox.org/?action=present&Username=ledeczi&ProjectName=NiceChat

and the client:
https://editor.netsblox.org/?action=present&Username=ledeczi&ProjectName=NiceClient

Note that if you try out the server, its public role id will be different for you since your user name is not ledeczi. You will need to change the client accordingly.

Advanced Chat Room

Our first chat room program was fairly simplistic: you needed to see the screen of the server program to see the messages. Not much of a chat room. It would be much better if every member of the chat room would see each other's messages on their own screen. How can we do that? One approach would be to send your messages to everybody else. But then you would need to know the public role id of each and every client. That is would be pretty problematic. Instead, what if the chat room server program forwarded every message it receives to everybody else? Let's do that.

First, we have to answer an important question: how does the server know the address, that is, the public role id of the clients? Well, we have to tell it. It means that first we have to send a message to the server with our own public role id. The server will store the address of every client in a list. Then when we send a regular chat message, it will send it to every item of the list. Notice that we have just designed our first communication protocol:

1. Client send its own address to server
2. Client send chat message to server
3. Server forwards said message to every client

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.

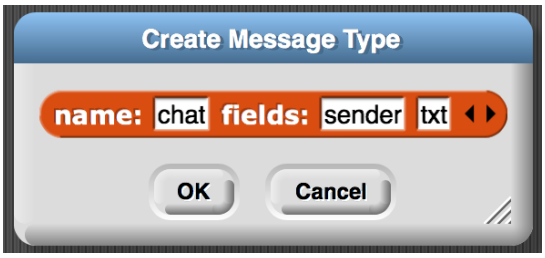
Since the content of the first message and the subsequent messages are different (first we send our address then we send chat messages), we need to define new message types. It is the message type that tells the server what kind of message it is according to the protocol and what data payload it has.

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 when everybody else display the message they will know who sent it.

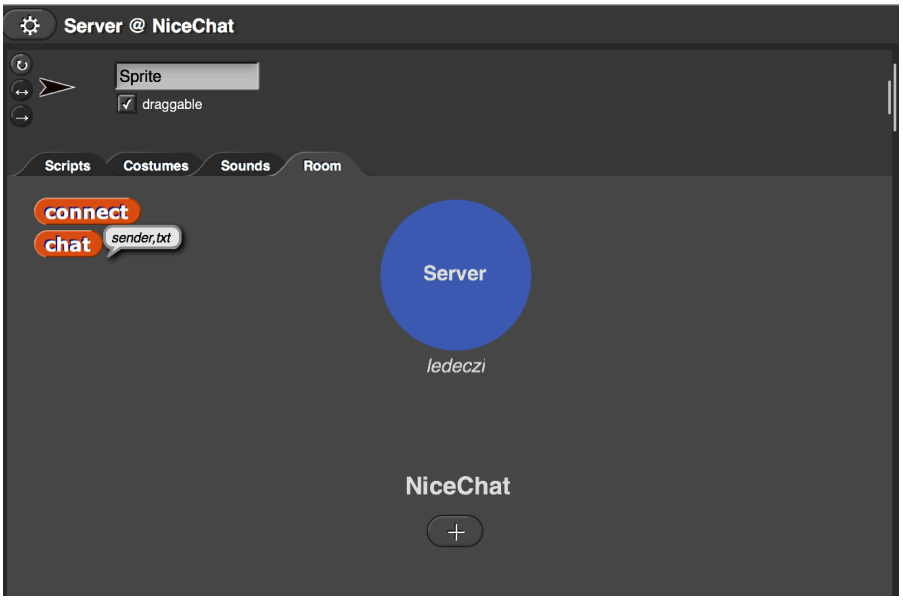
How do we define new messages in NetsBlox? In the Network tab there is a gray button called “Make a message type. “ When you click it, this window pops up:

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

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

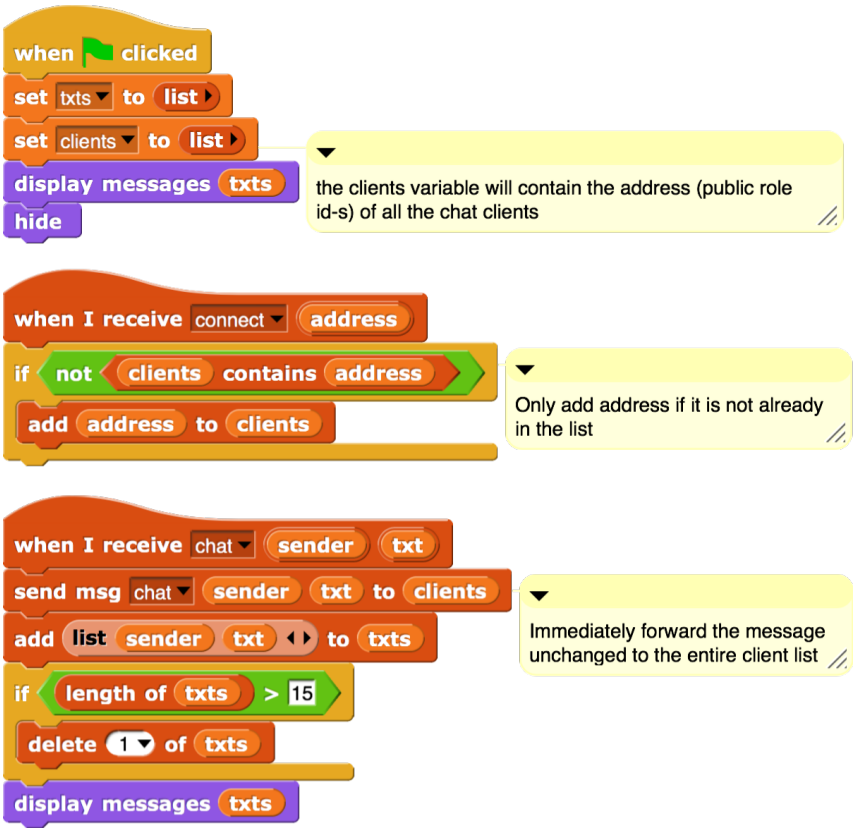


Once we defined these messages, we can actually see them under the Room tab. Check out the orange blocks, connect and chat, near the top left corner. If you click on them, they show their fields too:



It is very important that both NetsBlox programs, the sender and the receiver, have the exact same message definitions. If a single character is different, the receiver will never get the message.

We are ready to create our programs now. Here is the server:

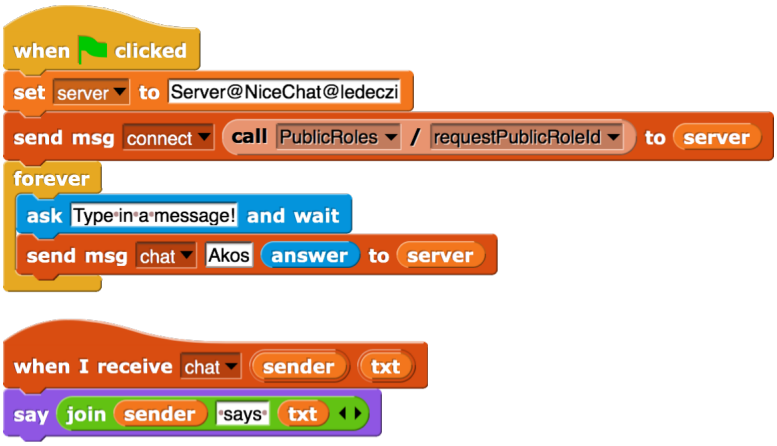


When the green flag is clicked, we initialize two variables: **`txts`** stores the last 15 chat messages along with their senders, while **`clients`** stores the addresses (public role id-s) of all connected clients. Both are initialized to empty lists. The initial call to the **`display messages`** custom block just clear the screen and displays the message: “Waiting for messages…” at the bottom of the stage.

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 address, it sends the a message to each of the items of the list. The rest of the code is very similar to the previous simple chat server: we add the sender and the message text to the list of messages and display them. It also prevents the txts list to become longer than 15.

Other than a little extra housekeeping and forwarding the messages, the server program is not more complicated than the first version. What about the client?



The first script is almost the same, but it starts by sending a connect message containing our own address that we obtain by calling the RPC requestPublicRoleId. Instead we can simply type it in since we know our own username, project name and Role name, but it is simpler and less error prone by using the RPC. Another change is that when we send actual chat message, we include our name, Akos in my case.

And the final change is that we have to receive chat messages too! To make life simple, we simply use the say block to display the received chat 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 too.