A first start developing atoms

Simulation Software / TUTORIAL

Papendorpseweg 77, 3528 BJ Utrecht, The Netherlands www.incontrolsim.com

A first start developing atoms

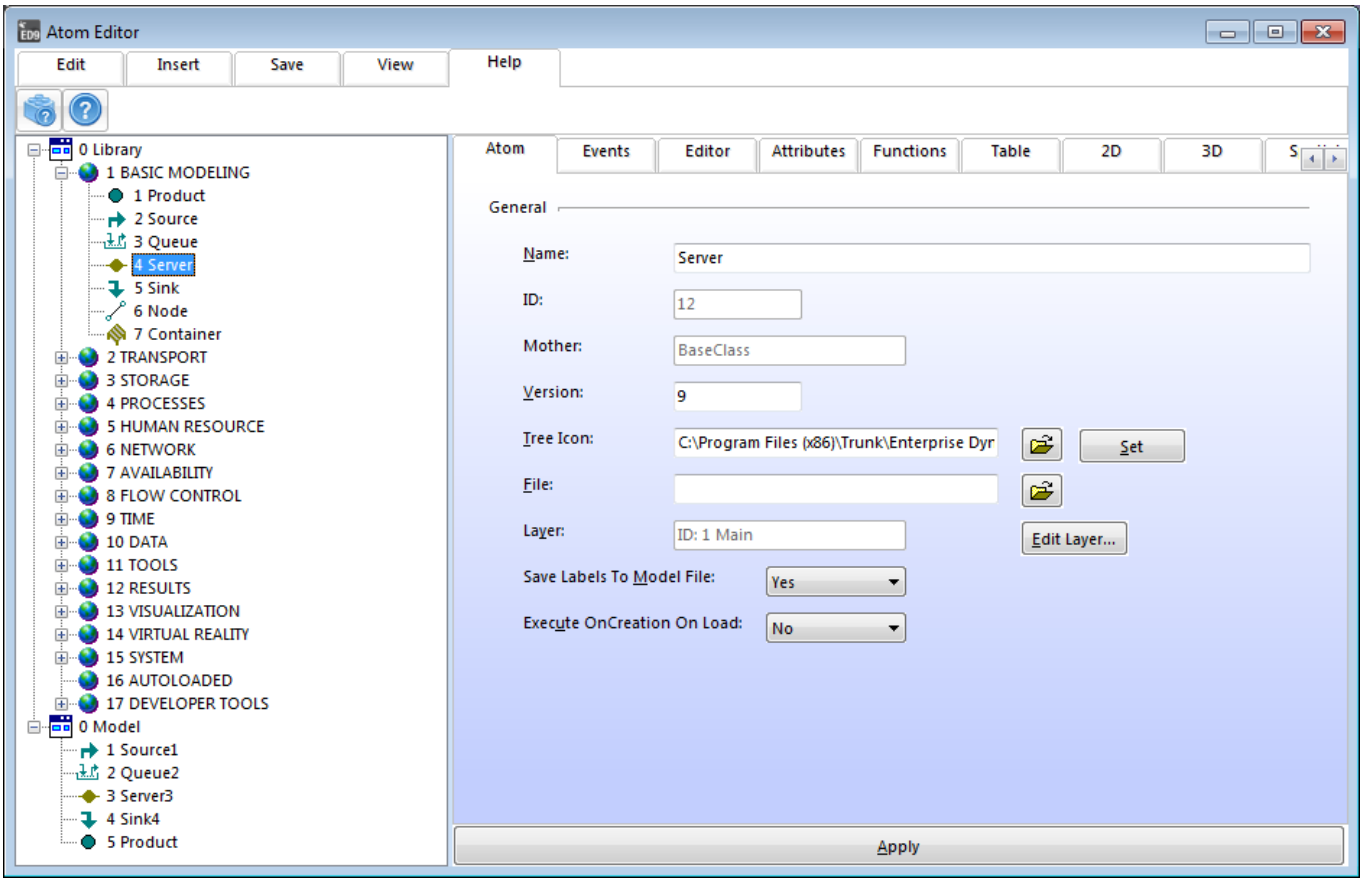
1 Table of Contents

2 The Atom Editor

2.1 The Atom Editor

The open structure of ED allows the advanced user to build and adapt atoms. Atoms are created or adapted with the Atom Editor.

The Atom Editor can be opened from the main menu by selecting <Display | Atom Editor>. One can also use the shortcut key Shift+F5. After opening the atom editor a window will appear as in Picture 1.



Picture 1: Atom Editor window

On the left side of the Atom Editor the Library Tree and the Model Tree are displayed. Here you can select atoms from each of these libraries. On the right, you can view and edit the selected atom. Different aspects of the atom can be found on separate sheets. For example, the position of the atom on the Spatial sheet, the animation of the atom on the 2D and the 3D sheet and most importantly, the behavior of the atom on the Events sheet.

> ⚠ *Never change Library atoms! Either make copies or change Model atoms.*

The Atom editor also has a main menu which is subdivided in the following pages:

Edit	Cut, copy, paste atoms or reorder the hierarchy of the atoms in the tree.
Insert	Create or load an atom.
Save	Save the atom.
View	View the labels, the user interface (GUI) or table of the atom.
Help	Open the documentation.

2.2 Creating an atom from scratch

First decide where you want to create the new atom, in the model or directly in the library. It is advised to

create new atoms in the model and move it to the library when it is finished. Thus select the Model on the left side of the Atom Editor and click the left most button on the *Insert Sheet* at the top to insert a completely new blank atom.

Creating a new atom in the model allows you to save the changes to your atom by saving the model. If you have finished adding the functionality to your atom then you can save the atom as a .atm file. You can load such a file in your library and then use it to build a model with it. You can also create a packages that contains this atom and make sure that ED loads it automatically in the library when you start ED.

A new atom has no behavior defined on it and is animated as a 1 by 1 black square positioned at the origin. Changes made to the atom on the Sheets of the Atom Editor only become effective in the corresponding atom in the model when you click Apply at the bottom of the Atom Editor window. Vice versa the Atom Editor is not updated automatically if changes are made to the model. For example, if you resize the new atom with the mouse, these changes are not yet applied in the Atom Editor. Click F5 to refresh the Atom Editor, changes made in the model are now applied in the Atom Editor.

Questions and assignments

1. Open the Atom Editor and insert a new atom in the Model. Go to the Spatial Sheet and change the X and Y location of the new atom. Click Apply in the Atom Editor Window to update the atom in the Model. Next go to the 2D Sheet and change the *Layout Color* of the atom to blue. Click Apply to update the new atom in the Model. Look in the Model Layout window to find the new blue atom. Next change the size of the atom in the 2D Model Layout. Note that this change is not applied to the Spatial Sheet of the Atom Editor. Press F5 to update the Atom Editor with the changes you made in the Model.

We now know how to insert a new blank atom in the model, but we still need to know how to give the new atom its specific behavior, a better 2D and 3D appearance and a user interface. This is all explained in the next sections.

2.3 The events sheet

The Events Sheet in the Atom Editor defines the behavior of an atom. Here you define how an atom should respond to certain events that happen during a simulation run.

On the left of the Event Sheet all event handlers are listed which you can use to define the behavior of the atom. One can distinguish 3 types of event handlers:

- Dynamic event handlers
  These are event, entering, entered, exiting, exited, icready, ocready and message
- Control event handlers
  These are reset, user, creation, destruction and init
- Animation event handlers
  These are 2Ddraw and 3Ddraw

The important difference between these types of event handlers is when they are executed. The control event handlers are executed in response to a user action, for example the user clicks on the atom with the mouse, while the animation event handlers are executed 35 times per second. The dynamic event handlers are executed if a specific event occurs in the model. For example, if a product atom enters a Server atom the Entered event of the Server is executed.

If you select the field behind an event the code in the field will be displayed in the small editor at the bottom side of the Event sheet. If you switch to the Editor Sheet of the Atom Editor the 4DScript code of the selected event handler is displayed in a larger editor.

We will first explain how to build a simple Queue atom from scratch and use this example to explain the most important dynamic event handlers.

2.4 Building a simple queue from scratch

In this section we will explain how to define the behavior of a simple Queue atom with capacity 3

Questions and assignments

2. Follow the explanation in this section to build the simple queue atom.

We first open the Atom Editor and insert a new atom in the model. On the Atom Sheet we change the name of the new atom to SimpleQueue and press Apply. We switch to the Channels sheet, to give it 1 input and 1

output channel such that the atom can interact with other atoms in the model. Click Apply.

We would now like to define the specific behavior of our simple Queue atom. We first need to make sure that we are aware of all the logic that needs to be implemented. Our Simple Queue with capacity 3 needs the following behavior:

- To enforce the capacity, we need to close the input channel when there are 3 products in the queue.
- If a product is in the queue and the next server atom becomes available, the product needs to be moved to the server atom.
- If the content of the queue becomes less than 3, allow new atoms in the queue, i.e. open the input channels.

Note that for each of these behaviors an action has to be performed after an event that occurs in the model. For example, when a third atom enters the queue (the event), we need to make sure no more atoms can enter our queue by closing the input channels (the action necessary after the event) or if a product atom leaves the queue and there are thus less than three products remaining in the queue (event), we need to make sure a new product atom can enter our queue (action). This is typical for Event-Based programming. You might have already used this when you defined code on the triggers, but this is more complex.

Before we can build our simple queue we first need to know a bit more about some of the event handlers.

The **OnEntered** event handler of an atom is executed when another atom enters this atom. Thus in our example when a product enters our simple queue the OnEntered event of the Simple Queue will be executed.

For our Simple Queue this means that every time a product has entered, we check if 3 product atoms are in the queue. If there are indeed three product atoms we have to make sure that no more atoms can enter. We can do this by closing its input channels. Recall that in the Model Layout open channels are displayed green, closed channels red.

The **OnOcReady** event handler of an atom is executed when one of the connections to its output channels becomes ready. A channels connection is ready when both the output channel of the atom and the input channel to which the atom is connected are open. The channel is then ready to pass on atoms. In the Model Layout an open input or output channel is visualized by a green dot; a red colored input or output channel is closed. If both the input channel and the output channel of a connection is available then the channel is available (and colored green).

Let's try to understand this better by looking at our simple queue. Let's assume that our simple queue is connected to the input of a server and the server has just finished its service. The server will become available and will open its input channels to allow a new product to enter the server. If the output channel of our simple queue is now open, the connection becomes ready and the OcReady event of our simple queue will be executed. We can use this event to make sure that the product atom in the queue is moved to the next atom, in this case a server.

The **OnEvent** event handler of an atom is executed at a predetermined time, and is scheduled by the 4DScript function `CreateEvent`.

The **OnReset** event handler of an atom is executed when the model is reset. Typically this happens when the user presses the reset button of the Run Control. Before it is called, all in- and output channels are opened automatically by ED and ED makes sure that in this case the OcReady events are not executed. ED also makes sure that all products in the model have been destroyed.

For our simple queue we will use the Reset event to close the output channels. Initially when the model is reset our queue is empty. Since our simple queue has a capacity of 3 we should thus allow products to enter the queue, this is why we leave the input channels open. Initially the queue is still empty so we don't have products to send to the next atom. If an atom has no atoms contained in it that are ready to be sent, then we choose to model this by closing its output channel. For our simple queue this means that on OnReset we will close the output channels.

From the above explanation of some of the event handlers and of how this applies to our simple queue it is clear that controlling the input and output channels is very important. The 4DScript command `CloseAllOC` can be used to close all output channels of an atom. It has one parameter which should be a reference to the atom of which you would like to close the output channels. The 4DScript command `c` is an atom reference to the *current* atom. For example, if we write code in one of the event handlers of our simple queue, we can use the atom reference `c` the refer to itself. This means that the only thing we have to write in the OnReset of the simple queue is the command `CloseAllOC(c)`. The 4DScript command `OpenAllOC(c)` is used for opening all output channels of the current atom.

We now continue implementing the behavior of our queue. Each time a product enters our queue the

OnEntered event handlers of our queue is executed. Thus we can check here if our queue is full. If the queue has indeed reached the maximum capacity of 3 we then need to make sure that no more product atoms can enter our queue so we need to close the input channels. Closing the input channels of an atom can be done with the 4DScript command `CloseAllIC`, this command has again one parameter which is a reference to the atom of which you want to close the input channels. There is a second behavior of the queue that can be implemented on the OnEntered event handler. If a product is in our queue it is immediately ready to be send to an atom that is connected to one of the output channels of our queue. Therefore we want to open the output channels of our queue. This should happen each time a product enters. Summarizing we get the following two commands on the OnEntered event of our queue:

```
if(Content(c) >= 3, CloseAllIC(c))
```
and
```
OpenAllOC(c)
```

We will not open the output channels on the OnEntered event handler directly. We can not explain this in detail here, but is has to do with the fact that opening the output channels can lead to a channel connection becoming available. If this is the case ED will immediately execute the OnOcReady event. Later we will show what problems can arise from this. For now we advise you not to open an input or output channel from an OnEntered or OnExited event handler, instead schedule an event to open the channels, as we will explain below.

Events that need to occur in an atom at a specific time can be scheduled with the 4DScript command `CreateEvent(e1, .., e5)` which has up to 5 parameters of which only the first one is obligatory. The first parameter is the time in seconds after which the OnEvent event handler will be triggered. Parameter e2 is a reference to the atom of which the OnEvent handler will be executed. The other parameters will be explained later. A typical example for which you would use the OnEvent handler is to schedule an event for the moment a product is ready to leave a server. Let's assume we have a server which models a process that takes exactly 9 seconds. The moment a product enters a server we have to make sure that after 9 seconds all actions are taken for the product to leave the server. Therefore we will schedule an event that after 9 seconds will open the output channels of the server. On the OnEntered of the server we need to write the command `CreateEvent(9, c)` and on the OnEvent handler of the server we write `OpenAllOC(c)`. On the OnEntered event of the simple Queue we would like to open the output channels immediately. This means that we will create an event to occur after 0 seconds. We we enter the code

```
CreateEvent(0, c)
```

At some point in time an atom connected to an output channel of our simple Queue will open its input channel. If our simple Queue contains a product we have opened our output channel. Thus the opening of the input channels of the atom that is connected to our simple Queue will result in the channel becoming ready. At that moment the OnOCready event of our simple Queue will be executed. Another situation in which the OCReady Event can be triggered is when the connecting atom is already available to receive a new product but our queue is still empty. The moment a new atom arrives at our simple Queue we open the output channels, which again results in the channel becoming ready. In both cases we need to send the product to the available atom connected to our simple Queue.

The best way to move the product atom from our simple Queue to the atom connected to it is to use the 4DScript command `MoveRequest(e1, e2)`. This function has two parameters. The first parameter e1 is a reference to an atom that you would like to move. The second parameter e2 is a number of an output channel through which you request to move the atom. Note that this is only a request to send the atom. For our simple Queue assume a standard First in First out (FIFO) strategy. Thus we would like to send the first atom in our Queue and since we have only 1 output channel we will send it through channel 1. This means we have to write the following code on the OCReady event:

```
MoveRequest(First(c), 1)
```

When the product is moved to the atom connected to an output channel of our Queue, the OnExited event of our Queue will be executed. If our Queue is now empty we have to close the output channels. Also if our simple Queue was full, i.e., it contained three products, but due to the product that is leaving it becomes ready to accept new products, we have to open the input channels to allow new products. Again if we open channels we do not do this on the Exited event directly, but we will create an Event that will trigger immediately to open the input channels. Thus

```
if(Content(c) = 0, CloseAllOC(c))
```
and
```
CreateEvent(0, c)
```

We now have two different events that can be scheduled for the event handler OnEntered we schedule events to open the output channels and on Exited we schedule events to open the input channels. To distinguish these to different events we will use the third parameter of the `CreateEvent` command.

Summarizing we have the following code to model the behavior of our simple Queue.

OnReset event handler

```
CloseAllOC(c)
```

OnEntered event handler

```
do(
 if(Content(c) >= 3, CloseAllIC(c)),
 CreateEvent(0, c, 1)
)
```

OnOCReady event handler

```
MoveRequest(First(c), 1)
```

OnExited event handler

```
do(
 if(Content(c) = 0, CloseAllOC(c)),
 CreateEvent(0, c, 2)
)
```

OnEvent event handler

```
Case(
 EventCode,
 OpenAllOC(c),
 OpenAllIC(c)
)
```

2.5 Storing data

In the previous section we have developed a queue with capacity 3. The code for a queue with a different capacity will be the same except for the value of the capacity itself. Such data can be stored in an attribute. The attributes of an atom can be viewed in the Attribute sheet of the Atom Editor. Attributes can contain Values, Text or 4DScript Expressions.

ED uses '=' to indicate that it is a 4DScript expression. You use the value of the attribute on that atom by calling the attribute name. Example, if we give our SimpleQueue an attribute named *capacity* we can query the value on the events of the SimpleQueue atom by simply calling the name of the attribute.

Questions and assignments

3. Open the Atom Editor and go to the attribute sheet of the SimpleQueue. Set the number of attributes to 1. In the name field type 'capacity' and in the attribute field type 3. On the Events sheet change the event codes by replacing the value 3 with the name of the attribute. If you are finished change the value of the attribute to 10 to test if your changes worked.

2.6 Saving and loading atoms

It is good practice to develop new atoms in the model. Changes made to the atom can then be easily stored in model files by just saving the model. If you have finished your atom you can save it as an .atm file. All the

functionality of your atom is saved in this file. You can save an atom as described in the following steps.

Save an atom

1. Select the atom in the Atom Editor.
2. Click the *Save* tab and then click *Save Treeatom atom as*.
3. A selector window appears. Make sure your atom is selected and click OK.
4. In the File name box, enter a name for you atom and click Save.

After you have saved an atom you can load it in the library and use it to build a model with it.

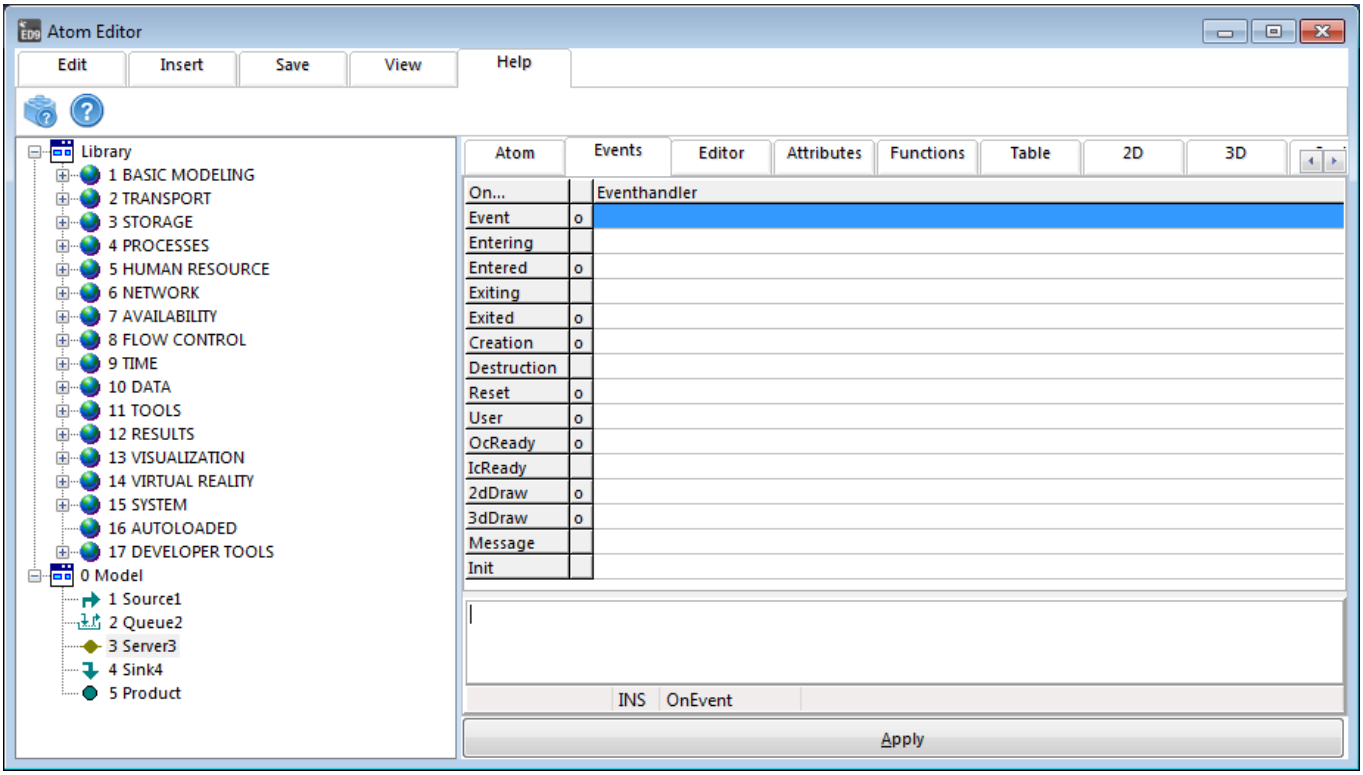
Load an atom in the library

1. Open the Atom Editor.
2. Select the Group atom in which you want to place the atom.
3. Click the Insert tab and then click *Load and add an atom to Library*.
4. In the left pane of the Open dialog box, click the folder that contains the atom.
5. Click the atom and then click Open.

You can now build a new model, dragging your own atom from the library to the layout window.

2.7 Inheritance

If you select in the Atom Editor an existing atom from the Model and select the Event Sheet it seems that there is no behavior defined on it. If we select a Server atom from the model the fields behind the events are empty, see Picture 2.



Picture 2: The Atom Editor Event sheet of a Server in the model

If we take a second look at Picture 2 we see an “o” symbol in the second column of Events Sheet. The “o” indicates that the Event of that row is inherited from the mother atom of this atom. On the Atom sheet you will see that the mother of this atom is the Server atom which can be found in the Library tree. The code can be copied from the mother to the daughter by double-clicking the event. You can then edit the code. Changes to the code will only effect the daughter in this case Server3 in the model. Note that if the event handler of daughter is empty and it has a “o” symbol behind it, inherited code from the mother is used, but if you write code in the field behind it the field is no longer inherited from the mother.

3 Project approach

3.1 Project approach

If you start building new atoms you should treat this like any other software development project as any other kind of software that needs to be written. Start by writing down the user requirements and specifications for the atom.

The second step is to make a technical design. How will the atom mimic reality? What is the internal logic of the atom?

The third step is to break down your logic into functions which make the atom do what you want it to do. Write down these functions in normal language so anyone can understand what your functions need to do.

You should now have a small document that anybody can still read and check. Two know more than one.

It is now time for 2 more steps. Write your functions in pseudo code, and the last step (before you start Enterprise Dynamics in the first place) is to write down each event handler in pseudo code.

By now you have already created your atoms logic, but not in Enterprise Dynamics yet. However this approach will let you think about the concept before you start programming. Many faults in logic have been detected in this way before time was spend on the actual programming to find out you made a mistake.

It will also make it much easier to actually program the atom because you have a validated script that you have to implement.

3.2 Tips for building atoms

1. Try to make each atom independent. Don't assume things because what might be true in one model might not be applicable in another model.
2. If you refer to attributes do it by their attribute names. Using the `Att` function is also possible but in this way you assume that the index of the attribute will never change. If you have to change the index for any reason you have to rewrite large parts of your code.
3. Do not save the results of calculations in your On2DDraw and On3DDraw event handlers. If you do this it will lead to different results of your simulation run depending on the visibility of the atom.