

# AnyLogic and Java

Nathaniel Osgood

# Advantages of AnyLogic

(as compared to other Agent-Based Modeling Software)

- Primarily declarative specification
- Less code
- Great flexibility
- Access to Java libraries
- Support for multiple modeling types
- Support for mixture of modeling types

# Painful Sides of AnyLogic Education/Advanced

- Export of model results: Lack of trajectory files
- Lack of a built-in debugger
- Need for bits of Java code
- Many pieces of system

# Internals of AnyLogic files: XML

```
C:\Usask\Classes\ABMCMCC2009\Models\HybridABMNetworkModeling1\HybridABMNetworkModeling1 Anylogic 6_2_2.alp - Notepad++
File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?
AnalyzeTBCaseContacts.R EraseFileInDirectory.pl FindMissingBrowseFiles.pl CINFilesToCSV2.pl CreateDataDictionaryFromSpreadsheet|ThinkEquations
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!--
3 *****
4 |           | AnyLogic Project File
5 |           | *****
6 -->
7 <AnyLogicWorkspace WorkspaceVersion="1.9" AnyLogicVersion="6.2.2.200806031102" AlpVersion="6.2.2">
8 <Model>
9   <Id>1257613518087</Id>
10  <Name><![CDATA[HybridABMNetworkModeling1 Anylogic 6_2_2]]></Name>
11  <ExcludeFromBuild>false</ExcludeFromBuild>
12  <EngineVersion>6</EngineVersion>
13  <JavaPackageName><![CDATA[hybridabmnetworkmodeling]]></JavaPackageName>
14  <ActiveObjectClasses>
15    <!-- ===== Active Object Class ===== -->
16    <ActiveObjectClass>
17      <Id>1257613518149</Id>
18      <Name><![CDATA[Main]]></Name>
19      <ExcludeFromBuild>false</ExcludeFromBuild>
20      <ClientAreaTopLeft><X>0</X><Y>0</Y></ClientAreaTopLeft>
21      <PresentationTopGroupPersistent>true</PresentationTopGroupPersistent>
22      <IconTopGroupPersistent>true</IconTopGroupPersistent>
23      <Generic>false</Generic>
24      <GenericParameters><![CDATA[T]]></GenericParameters>
25      <AgentProperties>
26        <SpaceType>CONTINUOUS</SpaceType>
27        <EnvironmentDefinesInitialLocation>true</EnvironmentDefinesInitialLocation>
28
29      </AgentProperties>
30
31      <DatasetsCreationProperties>
32        <AutoCreate>true</AutoCreate>
33        <SamplesToKeep>100</SamplesToKeep>
34        <FirstUpdateAtTime>true</FirstUpdateAtTime>
35        <FirstUpdateTime>0.0</FirstUpdateTime>
36        <FirstUpdateDate>1263556975211</FirstUpdateDate>
37
38      </DatasetsCreationProperties>
39    </ActiveObjectClass>
40  </ActiveObjectClasses>
41 </Model>
42 </AnyLogicWorkspace>

```

Normal text file 54746 chars 58036 bytes 1646 lines Ln:1 Col:1

# Java Code: When & How Much?

- “Java” is a popular cross-platform “object oriented” programming language introduced by Sun Microsystems
- Anylogic is written in Java and turns models into Java
- AnyLogic offers lots of ways to insert snippets (“hooks”) of Java code
- You will need these if you want to e.g.
  - Push AnyLogic outside the envelop of its typical support
    - e.g. Enabling a network with diverse Agent types
  - Exchange messages between Agents
  - Put into place particular initialization mechanisms
  - Collect custom statistics over the population

# Stages of the Anylogic Build

Modification Possible



Modification Not Possible

## Java Code

```
double initialPrevalenceOfInfection ) {
    if (initialPrevalenceOfInfection == this.initialPrevalenceOfInfection) {
        return;
    }
    this.initialPrevalenceOfInfection = initialPrevalenceOfInfection;
    onChange_initialPrevalenceOfInfection();
    onChange();
}

void onChange_initialPrevalenceOfInfection() {
    int index;
    index = 0;
    for ( Person object : Population ) {
        object.set_isInitiallyInfected((uniform() < initialPrevalenceOfInfection));
        index++;
    }
}
```

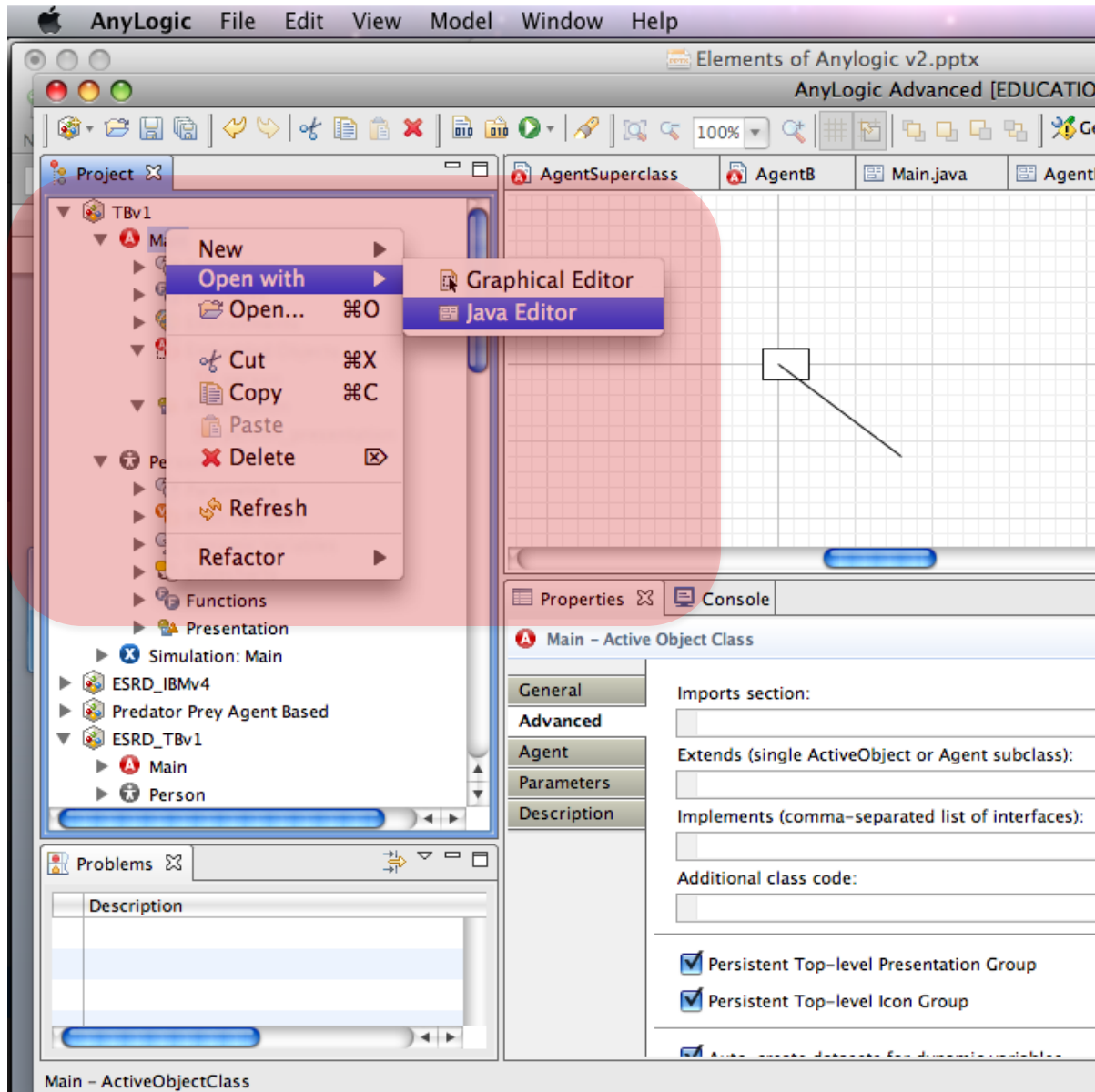
JVM  
Byte  
Code

Person.class

# Inspecting the Java code

- As a step towards creating an executable representation of the code, AnyLogic creates a Java representation
  - If you want to see the Java code for a model, you will need to do a “build”
- Sometimes it can be helpful to look at this Java code
  - To find errors about which AnyLogic may be complaining
  - Advanced: To see how things are being accomplished or “work”

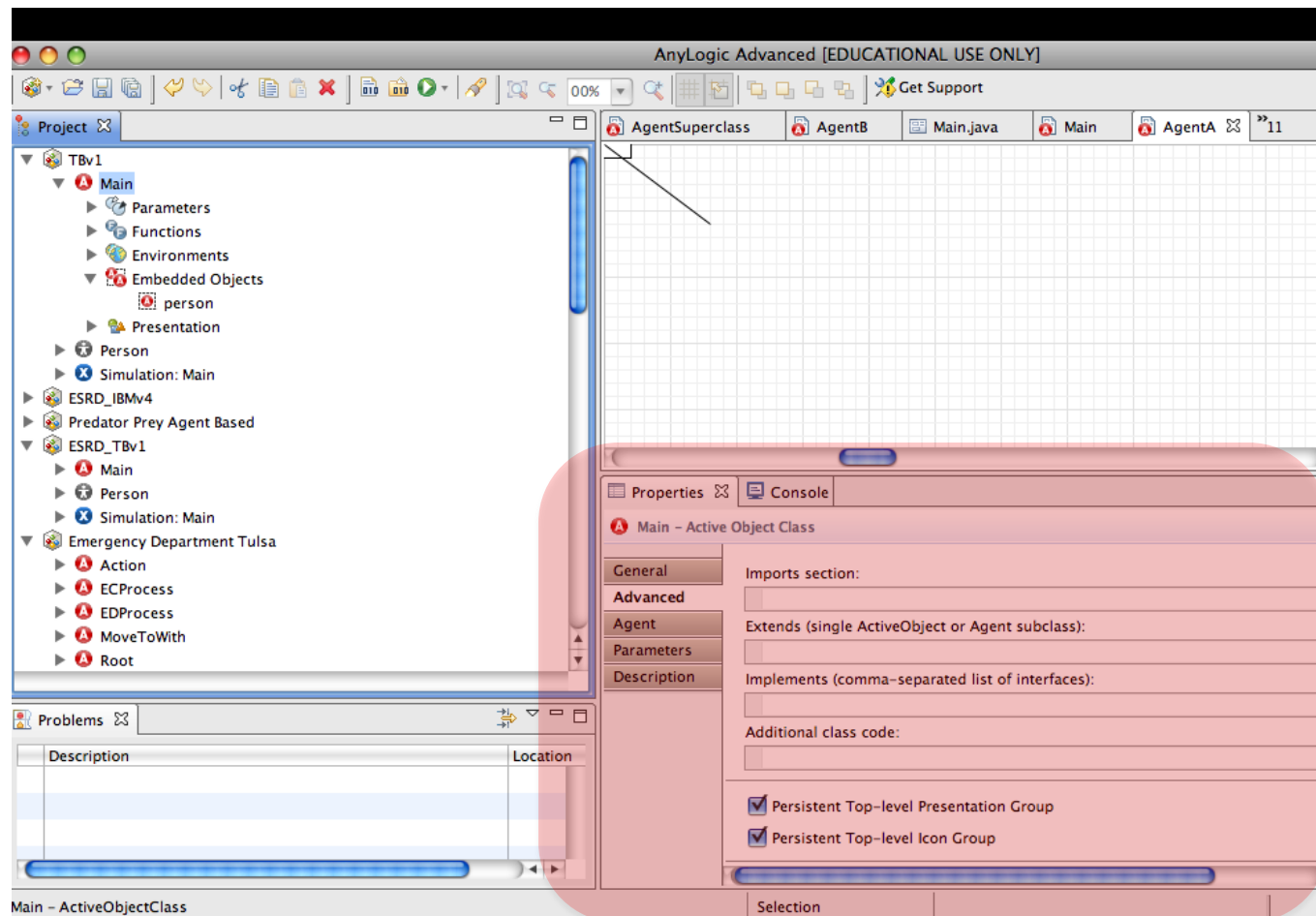
# Requesting Viewing of Java Code





# Examples of Where to Insert Code Object Properties

- “Advanced”



# Examples of Where to Insert Code

## Object Properties

- “General”

The screenshot displays the AnyLogic Advanced software interface. The top toolbar includes icons for file operations, navigation, and a 'Get Support' button. The main workspace shows a project tree on the left with the 'Person' object class selected. The right pane is divided into 'Properties' and 'Console' tabs. The 'Properties' tab is active, showing the 'Person - Active Object Class' configuration. The 'General' section is expanded, displaying the 'Name' field set to 'Person' and an 'Ignore' checkbox. The 'Agent' section shows the 'Agent' checkbox checked and the 'Generic' checkbox unchecked. The 'Description' section contains fields for 'Startup Code' and 'Destroy Code'. A red callout box highlights the 'Properties' and 'Console' tabs and the 'Person - Active Object Class' configuration area.

AnyLogic Advanced [EDUCATIONAL USE C

AgentSuperclass AgentB Main.java AgentFactory.java

Project TBv1

- Main
  - Parameters
  - Functions
  - Environments
  - Embedded Objects
    - person
  - Presentation
    - person\_presentation
- Person
  - Parameters
  - Plain Variables
  - Dynamic Variables
  - Statecharts
  - Functions
  - Presentation
- Simulation: Main
- ESRD\_IBMv4
- Predator Prey Agent Based
- ESRD\_TBv1
  - Main
  - Person

Properties Console

Person - Active Object Class

General Name: Person  Ignore

Advanced

Agent  Agent  Generic

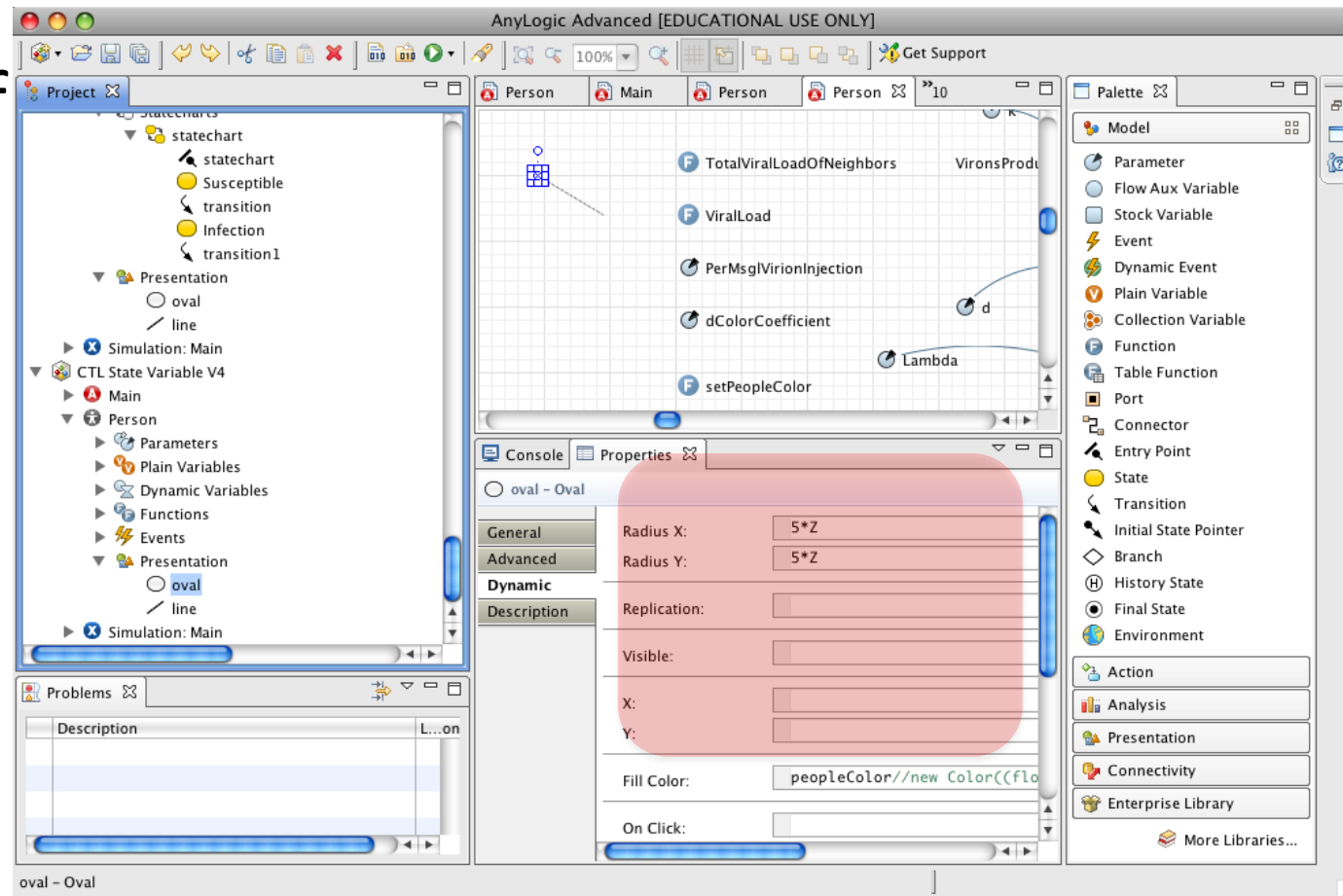
Parameters

Description Startup Code:

Destroy Code:

# Example of Where to Insert Code Presentations Properties

- “Dynamic” properties of presentation elements (especially of Agents)



# Tips to Bear in Mind While Writing Code

- Click on the “light bulb” next to fields to get contextual advice (e.g. on the variables that are available from context)
- While typing code, can hold down the Control key and press the “Space” key to request autocompletion
  - This can help know what parameters are required for a method, etc.
- Java is case sensitive!
- Can press “Control-J” to go to the point in Java code associated with the current code snippet
- Can press “build” button after writing snippet to increase confidence that code is understood

# Example of Contextual Information

The screenshot displays the AnyLogic software interface. The top menu bar includes 'AnyLogic', 'File', 'Edit', 'View', 'Model', 'Window', and 'Help'. The title bar reads 'AnyLogic Advanced [EDUCATIONAL USE ONLY]'. The main workspace shows a statechart with three states: 'people [...]', 'environment', and 'CountInfectious'. The 'CountInfectious' state is selected, and its properties are shown in the 'Properties' window. The 'Name' is 'CountInfectious', the 'Type' is 'Count', and the 'Expression' is 'Use: item: the embedded object'. The 'Condition' is 'item.statechart.isStateActive(Person)'. The 'Palette' on the right lists various model elements like 'Parameter', 'Flow Aux Variable', 'Stock Variable', etc. The 'Project' tree on the left shows the simulation structure, including 'Simulation: Root', 'Ophthalmology Department', 'MainPhase1-3', 'Simulation: MainPhase3', 'NetworkSEIR', 'Main', 'Parameters', 'Plain Variables', 'Functions', 'Environments', 'environment', 'Embedded Objects', 'Analysis Data', 'datasetAbsolutePrevalence', 'CountInfectious', 'Presentation', 'Person', 'Plain Variables', 'Statecharts', and 'statechart'. The 'Problems' window at the bottom left is empty.

AnyLogic Advanced [EDUCATIONAL USE ONLY]

Project

- Simulation: Root
- Ophthalmology Department
  - MainPhase1
  - MainPhase2
  - MainPhase3
  - Simulation: MainPhase3
- NetworkSEIR
  - Main
    - Parameters
    - Plain Variables
    - Functions
    - Environments
      - environment
    - Embedded Objects
    - Analysis Data
      - datasetAbsolutePrevalence
      - CountInfectious
    - Presentation
    - Person
      - Plain Variables
      - Statecharts
        - statechart

Person Main Person Person "9

Model

- Parameter
- Flow Aux Variable
- Stock Variable
- Event
- Dynamic Event
- Plain Variable
- Collection Variable
- Function
- Table Function
- Port
- Connector
- Entry Point
- State
- Transition
- Initial State Pointer
- Branch
- History State
- Final State
- Environment
- Action
- Analysis
- Presentation
- Connectivity
- Enterprise Library
- More Libraries...

Console Properties

people - Person

General

Name: CountInfectious

Parameters

Statistics

Type:  Count  Sum  Average  Min  Max

Description

Expression: Use: item: the embedded object

Condition: item.statechart.isStateActive(Person)

Add Statistics

Description	Location
-------------	----------

# Autocompletion Info (via Control-Space)

The screenshot displays the AnyLogic University interface. The main workspace shows a statechart diagram with a 'statechart' node connected to 'Susceptible' and 'Infected' states. The 'Person - Active Object Class' properties window is open, showing the 'On message received' field with the text 'statechart.rece' entered. An autocompletion tooltip is visible, providing details for the 'receiveMessage' method.

**Person - Active Object Class**

General: X: [ ], Y: [ ]

Advanced: [ ]

**Agent**

Preview: [ ]

Description: [ ]

Movement parameters:

Velocity: [ ]

Rotation: [ ]

On arrival: [ ]

On message received: statechart.rece

On before step: [ ]

On step: [ ]

**receiveMessage**

public boolean receiveMessage(int msg)

Same as receiveMessage( Object msg ) but with an integer as message

**Parameters:**

msg - the integer posted to the statechart

Press 'Tab' from proposal table or click for focus

Selection X=246, Y=275

# Finding the Enclosing “Main” class from an Embedded Agent

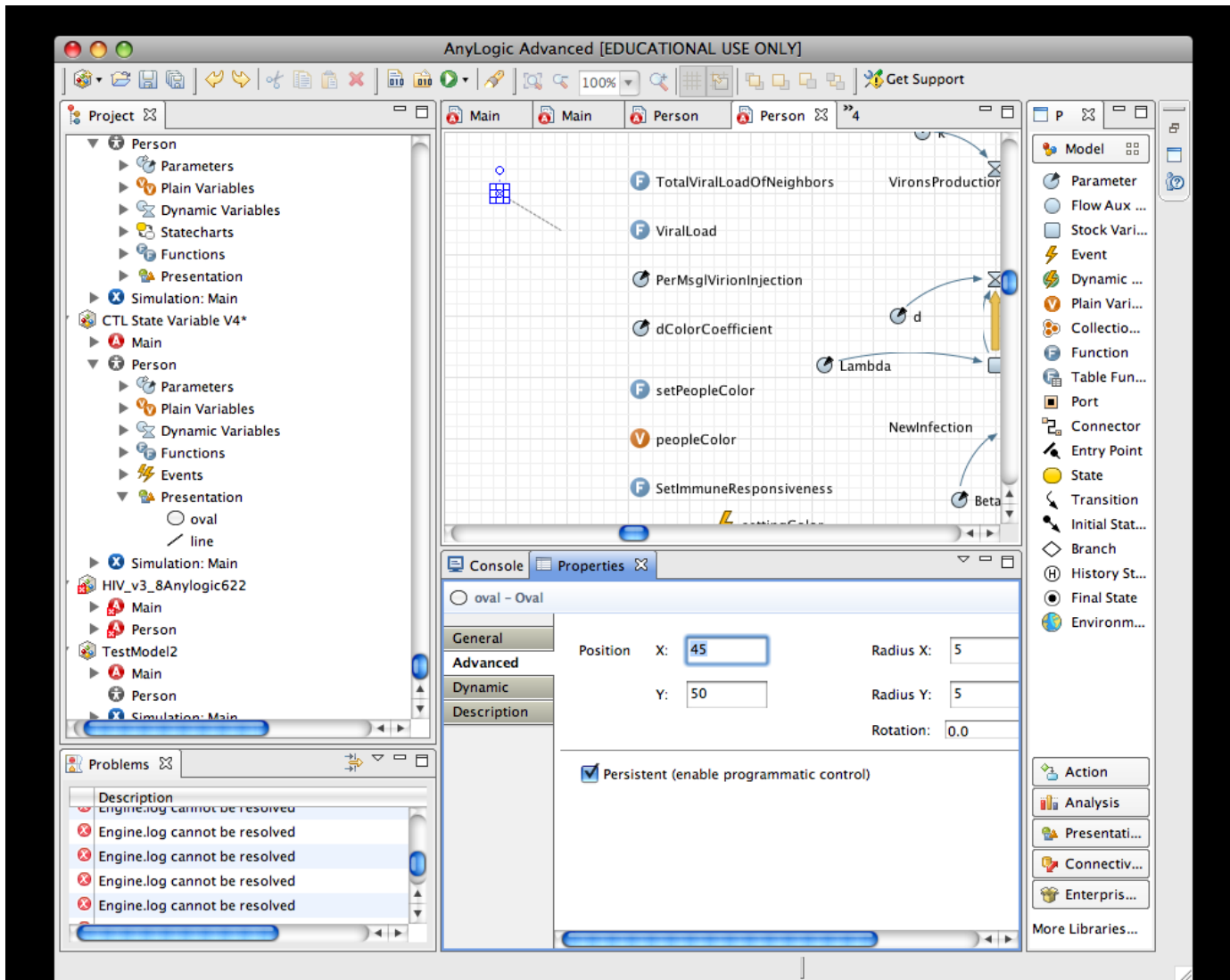
- From within an embedded Agent, one can find the enclosing “Main” class by calling `get_Main()`
  - This will give a reference to the single instance (object) of the Main class in which the agent is embedded
  - An alternative approach is to call `((Main) getOwner)`

# Presentation Properties

- Both key customizable classes (“Main”, various Agent classes) can be associated with “Presentation” elements
- These elements are assembled during execution into animations & presentations of the agents
- Many of these presentation elements have properties that can be set to Java expressions



# Enabling Programmatic Control



# Getting to the AnyLogic Help

- Choose “Help”/”Help Contents”
- AnyLogic help includes many components
  - Tutorials
  - User references
  - AnyLogic “library” information

# Getting Information on the Anylogic (Java) Libraries

Help - AnyLogic Advanced

http://127.0.0.1:63191/help/index.jsp

Wikipedia Save to Delicious My Delicious CMPT 858 CMPT 371 Env Canada Sask Weather Weather: Saskatoon Env Canada PA Weather The Pali Tex...h dictionary

Help - AnyLogic Advanced

Search:  **GO** Search scope: All topics

**Contents**




- Enterprise Library Tutorial
- Enterprise Library Reference Guide
- AnyLogic Help
- System Dynamics Tutorial
- Agent-Based Modeling Tutorial
- API Reference

## Using AnyLogic Help System

Browse topics in the **Contents** frame on the left. Click on a topic to have it displayed. Use the **Back** and **Forward** buttons to navigate within the history of viewed topics.

### Style conventions

To make things easy to follow, there are a number of formatting conventions and images used throughout the book:

- Bold** – Used for the names of UI elements such as menus, buttons, field labels, palettes, and view titles.
- Italic* – Used for emphasizing new terms.
- `Courier` – Used for code examples, references to class and function names.
-  – "How to" scenario.
-  – Reference to another help topic.
-  – The feature is available in **AnyLogic Professional edition** only.

### Printing multiple help topics

You can now [print multiple topics](#) in the help window with a single action. The new print drop-down button above the table of contents allows you to print a complete topic sub-tree at any level.

### Searching

To quickly locate topics on a particular subject in the documentation, enter a query in the **Search** field. Use the **Search** frame to display the **Search** view. After you run a search and find a topic you were looking for, click **Show in Table of Contents** button to match the navigation tree with the current topic.

# The Notion of a Code “Library”

- A “library” lets third parties (e.g. xjtek) share compiled code they have developed with others
- The classes built into our AnyLogic projects (e.g. Agent, ActiveObject, NetworkResourcePool, etc.) are contained in the library
- The available libraries that come with AnyLogic & Java have many additional components that can offer tremendous additional functionality
  - By tapping into this functionality, we can avoid having to write code ourselves
- To use a library, you need to know what is in it!

# Finding out Information

## Interfaces for Library Elements 1

The screenshot shows a web browser window with the URL `http://127.0.0.1:63191/help/index.jsp`. The browser's address bar contains a search box with the text "GO" and "Search scope: All topics". The browser's tabs show "Help - AnyLogic Advanced". The browser's bookmarks bar includes "Wikipedia", "Save to Delicious", "My Delicious", "CMPT 858", "CMPT 371", "Env Canada Sask Weather", "Weather: Saskatoon", "Env Canada PA Weather", and "The Pali Tex...h dictionary".

The browser's content area displays the "API Reference" page for the `com.xj.anylogic.engine` package. The page title is "API Reference > com.xj.anylogic.engine". The page content includes a navigation bar with links for "Overview", "Package", "Class", "Use Tree", "Deprecated", "Index", and "Help". The "Class" link is selected. Below the navigation bar, there are links for "PREV CLASS", "NEXT CLASS", "SUMMARY: NESTED", "FIELD", "CONSTR", and "METHOD". There are also links for "FRAMES", "NO FRAMES", and "DETAIL: FIELD", "CONSTR", "METHOD".

The main content area shows the class `com.xj.anylogic.engine.Agent`. The class is a subclass of `java.lang.Object`. The class hierarchy is shown as follows:

```
java.lang.Object
├── com.xj.anylogic.engine.Presentable
│   └── com.xj.anylogic.engine.Utilities
│       └── com.xj.anylogic.engine.ActiveObject
│           └── com.xj.anylogic.engine.Agent
```

The page also lists "All Implemented Interfaces:" `com.xj.anylogic.engine.internal.Child`, `java.io.Serializable`.

The class definition is shown as follows:

```
public class Agent
extends ActiveObject
```

The page provides a description of the class: "A subclass of `ActiveObject` designed to support agent based modeling, in particular:"

- time (continuous or discrete)
- space (continuous or discrete) and spacial animation
- connections between agents, networks (e.g. social) and their visualization
- communication - message passing and broadcasting

A user-defined agent class should be a subclass of `Agent` in order to use those features. If your model is agent based, but none of the above features are required, it is recommended to use regular `ActiveObject` as a base class for your agents, and not this class: `Agent` requires 36+ bytes more memory than `ActiveObject`.

# Finding out Information Interfaces for Library Elements 2

Help - AnyLogic Advanced

http://127.0.0.1:63191/help/index.jsp

GO Search scope: All topics

Contents

- AnyLogic Help
- System Dynamics Tutorial
- Agent-Based Modeling Tutorial
- API Reference
  - com.xj.anylogic.engine
    - AbstractShapeGISMap
    - ActiveObject
    - ActiveObjectArrayList
    - ActiveObjectCollection
    - ActiveObjectIntegrationManager
    - ActiveObjectList
    - Agent**
    - CustomDistribution
    - Dimension
    - DynamicEvent
    - Engine
    - Environment
    - Environment.AgentCollection
    - Event
    - EventCondition
    - EventOriginator
    - EventRate
    - EventTimeout
    - Experiment
    - ExperimentCompareRuns
    - ExperimentOptimization
    - ExperimentParamVariation
    - ExperimentSimulation

**Fields inherited from class com.xj.anylogic.engine.Presentable**

[ALIGNMENT\\_CENTER](#), [ALIGNMENT\\_LEFT](#), [ALIGNMENT\\_RIGHT](#), [LINE\\_STYLE\\_DASHED](#), [LINE\\_STYLE\\_DOTTED](#), [LINE\\_STYLE\\_SOLID](#), [SHAPE\\_ARC](#), [SHAPE\\_BUTTON](#), [SHAPE\\_CAD](#), [SHAPE\\_CHART\\_BAR](#), [SHAPE\\_CHART\\_HISTOGRAM](#), [SHAPE\\_CHART\\_HISTOGRAM2D](#), [SHAPE\\_CHART\\_PIE](#), [SHAPE\\_CHART\\_PLOT](#), [SHAPE\\_CHART\\_STACK](#), [SHAPE\\_CHART\\_TIME\\_COLOR](#), [SHAPE\\_CHART\\_TIME\\_PLOT](#), [SHAPE\\_CHART\\_TIME\\_STACK](#), [SHAPE\\_CHECKBOX](#), [SHAPE\\_COMBOBOX](#), [SHAPE\\_CURVE](#), [SHAPE\\_EMBEDDED\\_OBJECT](#), [SHAPE\\_FILECHOOSER](#), [SHAPE\\_GROUP](#), [SHAPE\\_IMAGE](#), [SHAPE\\_LINE](#), [SHAPE\\_LISTBOX](#), [SHAPE\\_OVAL](#), [SHAPE\\_PIXEL](#), [SHAPE\\_POLYLINE](#), [SHAPE\\_PROGRESSBAR](#), [SHAPE\\_RADIOBUTTONS](#), [SHAPE\\_RECTANGLE](#), [SHAPE\\_ROUNDED\\_RECTANGLE](#), [SHAPE\\_SLIDER](#), [SHAPE\\_TEXT](#), [SHAPE\\_TEXTFIELD](#)

**Constructor Summary**

[Agent](#)([Engine](#) engine, [ActiveObject](#) owner, [ActiveObjectCollection](#)<?> collection)

**Method Summary**

java.lang.String	<a href="#">agentInfo</a> ()
void	<a href="#">connectTo</a> ( <a href="#">Agent</a> a) Creates a bi-directional connection between this agent and a given other agent.
void	<a href="#">deliver</a> (java.lang.Object msg, <a href="#">Agent</a> dest) Delivers a message to a given agent immediately during this method call.
void	<a href="#">deliver</a> (java.lang.Object msg, int mode) Delivers a message to an agent or a group of agents, as specified by the mode parameter immediately during this method call.
boolean	<a href="#">disconnectFrom</a> ( <a href="#">Agent</a> a)

# Using Libraries

- There are two major libraries that are “built in” and can be used without additional reference: Java libraries & AnyLogic libraries
- To use an object in the Java libraries, you will use an “import” statement

# Using External Libraries

- There are tremendous numbers of 3<sup>rd</sup> party libraries available for Java
- The functionality associated with these libraries is incredibly diverse
- Many of these libraries are available for free; others are sold
- It is very easy to make use of the functionality of 3<sup>rd</sup> party libraries from AnyLogic
  - In order to do this, AnyLogic needs to “know about” the external library.



# Adding External Libraries 1

The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a diagram of a 'Person' entity with a black dot representing a state. A yellow arrow labeled 'Aging' points to the 'Age' variable. Other variables shown include 'Weight', 'Sex', and 'Ethnicity'. The 'Properties' panel at the bottom lists 'color', 'CirclePerimeterColorFromState', and 'CirclePerimeterWidthFromState'.

The 'Dependencies' panel is open, showing the following table:

AnyLogic libraries required to build the model:		
Name	Version	Location

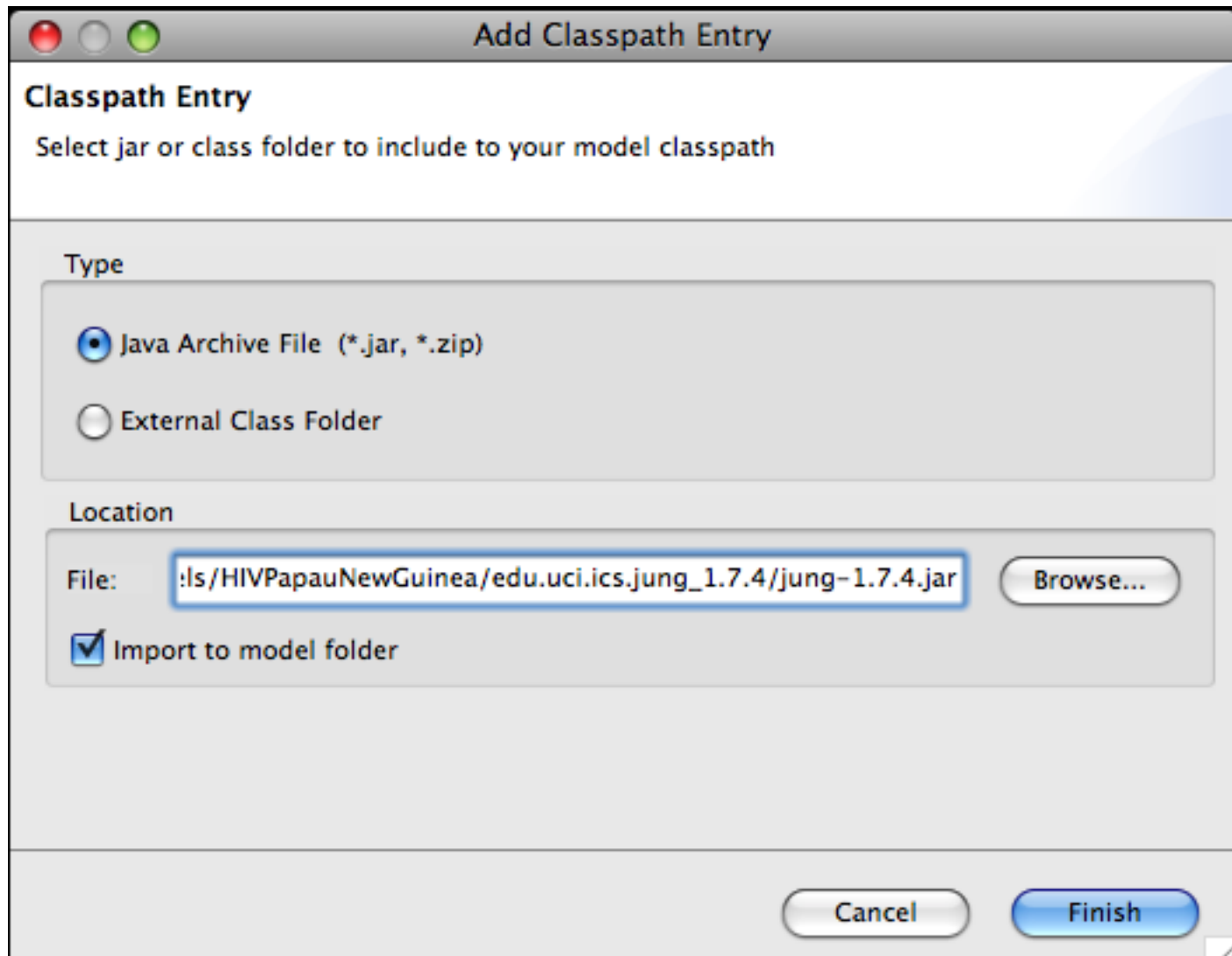
Below this table, the 'Jar files and class folders required to build the model:' section contains a table with one entry:

Location
jung-1.7.4.jar

The 'Problems' panel at the bottom left shows four error messages: 'The import edu cannot be resolved'.

The right sidebar contains a 'Model' palette with various components like Parameter, Flow Aux, Stock Vari, Event, Dynamic, Plain Vari, Collectio, Function, Table Fun, Port, Connector, Entry Point, State, Transition, Initial Stat, Branch, History St, Final State, and Environm. At the bottom of the sidebar are buttons for Action, Analysis, Presentati, Connectiv, and Enterpris, along with a 'More Libraries...' link.

# Adding External Libraries 2



# Common Contextual Variables that are Used by Code Snippets

- In statistics: “item” indicates current agent
- In “On Message Received” handler for agent: “msg” indicates received message
- In Dynamic properties of an Agent’s replicated line property: “index” indicates current person’s index
- In “Parameters” properties of Agent populations (used to set properties of agents within population): “index” indicates the index of the current agent in the population

# Example code to Export Dataset

```
FileOutputStream fos = new  
    FileOutputStream("Filename");  
PrintStream p = new PrintStream(fos);  
p.println(datasetName.toString()); // outputs  
    comma delimited values
```

# Useful Bits of Java Code

- `get_Main()` gets reference to Main object
- `ActiveObject.trace(str)` outputs string to log
- `Engine.getTime()` gets the current time
- `agents.size()` gets number of objects in collection  
`agents`
- `agents.item(i)` gets item i from agent collection
- `uniform()` generates a random number from 0..1

# Useful Bits of Java Code : General Expressions

- `ActiveObject.traceIn(Stringstr)` outputs string to log
- `time()` gets the current internal model time (different from the time in the external world)
- Members of `com.xj.anylogic.engine`.[Utilities](#)
  - `uniform()` generates a random number from 0..1
  - `uniform(x)` gen. a random number in range 0 to x
  - `lognormal(double meanNormal, double sigmaStdDevNormal, double minNormal)` draws from a lognormal distribution
  - `normal(double meanNormal, double sigmaStdDevNormal)` draws from a normal distribution
  - Many other probability distributions

# Methods on Populations of Agents (in Main class)

- `population.size()` gets number of objects in collection *population*
- `population.statName()` retrieves the current value of the population statistic `statName`, as computed for *population*.
- `population.item(int i)` gets item `i` from *population* collection
- `add_populationname()` Adds agent to that *population*
- `remove_populationname()` Removes agent from that *population*

# Useful Java Code: Methods to Call on (or from within, using “this”) an Agent

- `a.getConnectionsNumber()` returns number of connections between this agent and others
- `get_Main()` gets reference to Main object
- `toString()` gets string rendition of agent
- `a.getConnections()` gets a collection (linked) list of agents to which this agent is connected (& over which we can iterate)
- `a.connectTo(Agent b)` connects a to b
- `a.disconnectFrom(Agent b)` disconnects b from a
- `a.disconnectFromAll()` disconnects all agents from a
- `a.getConnectedAgent(int i)` gets the ith agent connected to a
- `a.isConnectedTo(Agent b)` indicates if a is connected to b



# Methods on Statecharts

(Called from within Agent code)

- `isStateActive(int statename)` indicates whether agent is in a given state (composite or simple)
- `getActiveSimpleState()` Get number of simple state. Can then compare to different state names, e.g. in switch statement.

# Methods on Process Flow Diagrams

- *source.inject(int count)* injects a count of entities into the *source* object (i.e. into an object of type Source)

# Gotchas

- Changing rates for leaving a state do not get updated until leave & reenter state (including by a self-transition)

# Example Use of getActiveSimpleState

```
switch (TBProgressionStatechart.getActiveSimpleState())
{
    case LTBI:
        return Color.YELLOW;
    case UnDiagnosedActiveTB:
        return Color.RED;
    case DiagnosedActiveTB:
        return Color.ORANGE;
    case TBSusceptible:
    default:
        return Color.BLACK;
}
```

# Useful Snippets: Handling Messages

- Sending
  - `sender.deliver(msg, receiver)` immediately deliver a message from sender to receiver
  - `sender.send(msg, receiver)` deliver a message from sender to receiver
  - `environment.deliverToRandom(msg)` [within Main] immediately deliver a message to a random agent in the environment
  - `send("Infection", RANDOM_CONNECTED)` [within an agent] send a message to a randomly selected agent connected to this one (where those agents are selected w/uniform prob)
- Receive message
  - `TBProgressionStatechart.receiveMessage(msg)` to forward message received by agent to statechart

# Useful Snippets

- Fields of dynamic properties of line object for Agent Presentation (Under “*Dynamic*” tab of line’s properties)
  - Replication: `getConnectionsNumber()`
  - dX: `getConnectedAgent(index).getX() - getX()`
  - dY: `getConnectedAgent(index).getY() - getY()`
  - These basically allow for appropriate initiation of visual properties of the inter-agent connections
- In Agent’s “On Message Received” Handler (Under “Agent” tab of Person)
  - `statechartname.receiveMessage( msg )`
  - This forwards a message received by this agent to statechart; note that if there are different messages, destined for different statecharts, they can be dispatched here to different targets