Routing Test Simulation

Set up simulation

```
open('RoutingTest.slx');
```

The Start Function of the simulation model sets all of the global properties in the base workspace (when the simulation is started)

- Service time = DIM(machineType, processType, productType) -- The serviceTime generation has the potential to not be a complete matrix (controlled by [rand(i,j,k) < 1]), denoting that some parts or processes cannot be executed on particular machines. Currently, it's complete to simplify assignment.
- Instantiates a Routing Controller (typed by Controller)
- Instantiates a Simple Routing strategy class and gives it to the controller for routing decision support
- For each resource/machine in the system, creates a corresponding workplace object and sets up event listeners

```
get_param('RoutingTest', 'StartFcn')
```

```
ans =
    'coder.extrinsic('evalin');
     coder.extrinsic('strcat');

     productInstances = [];
     productIDs = [];

     evalin('base', 'productTypeIDSet = {''ProductW001'', ''ProductW002'',''ProductX001'', ''ProductY001'', ''Produ
     evalin('base', 'productTypeIDSet = string(productTypeIDSet);');

     numProd = evalin('base', 'length(productTypeIDSet)');
     evalin('base', 'serviceTime = [rand(5,5,numProd) < 1].*[1+0.3*randn(5,5,numProd)];');

     evalin('base', strcat('RoutingController = Controller; RoutingController.Routing = SimpleRouting;'));

     evalin('base', 'machine1 = Machine(1); RoutingController.createResourceListener(machine1);');
     evalin('base', 'machine2 = Machine(2); RoutingController.createResourceListener(machine2);');
     evalin('base', 'machine3 = Machine(3); RoutingController.createResourceListener(machine3);');
     evalin('base', 'machine4 = Machine(4); RoutingController.createResourceListener(machine4);');'
```

Arrival Process

Basic attributes about the Task Generation block. Attribute Names indicate the attributes of Task that can be tracked during the simulation and carried by the Task

```
get_param('RoutingTest/TaskGenerator', 'Period')
```

```
ans =
'2.5'
```

```
get_param('RoutingTest/TaskGenerator', 'EntityTypeName')
```

```
ans =
'Task'
```

```
get_param('RoutingTest/TaskGenerator', 'AttributeName')
```

```
ans =
'productTypeID|serialNumber|nextNode|ServiceTime|condition|enterTime'
```

The Generate Action

- Selects a product type ID at random (from 6 product types)
- Assigns it a serial number (and tracks serial numbers)
- Gets the current simulation time and sets it to the entity

```
get_param('RoutingTest/TaskGenerator', 'GenerateAction')
```

```
ans =
    'coder.extrinsic('randi');
     coder.extrinsic('evalin');
     persistent idx;
     persistent rngInit;
     if isempty(idx)
         idx = 1;
     end
     if isempty(rngInit)
         seed = 12345;
         rng(seed);
         rngInit = true;
     end

     % Pattern: Arbitrary discrete distribution
     % V: Value vector
     % P: Probability vector
     entity.productTypeID = randi(8);

     entity.serialNumber = idx;
     idx = idx + 1;

     entity.enterTime = evalin('base', 'get_param(''RoutingTest'',''SimulationTime'');');'
```

The Exit Action

- Looks up the Product Type based on the the Product Type ID
- Add the product (and its instance ID) to lists on the base workspace for tracking and data manipulation in base
- Create an event listener for the Product Instance (which also creates listeners for the processes too)

```
get_param('RoutingTest/TaskGenerator', 'ExitAction')
```

```
ans =
    'coder.extrinsic('evalin')
```

```
coder.extrinsic('strcat')
coder.extrinsic('num2str')

%productTypeIDSet = evalin('base', 'productTypeIDSet');
productTypeIDSet = {'ProductW001', 'ProductW002','ProductX001', 'ProductY001', 'ProductZ001', 'ProductX002', '

productTypeID = productTypeIDSet{entity.productTypeID};
evalin('base', strcat('productInstances{end+1} = ', productTypeID, '(', num2str(entity.serialNumber), ');'));
evalin('base', strcat('productIDs(end+1) = ', num2str(entity.serialNumber), ';'));

evalin('base', strcat('part = productInstances{[productIDs == ', num2str(entity.serialNumber), ']};', ...
    'RoutingController.createProductListener(part);', ...
    'part.processPlan.actualStartTime = get_param(''RoutingTest'',''SimulationTime'');'));'
```

Queuing

There is one queue in front of each machine and it's sequenced by time in system (prioritizing completing jobs)

```
get_param('RoutingTest/Entity Queue', 'PrioritySource')
```

```
ans =
'enterTime'
```

Machines

Each machine is a single server that gets it's service time from an attribute on the entity

```
get_param('RoutingTest/Machine 1', 'ServiceTimeAttributeName')
```

```
ans =
'ServiceTime'
```

The Entry Action

- Looks up the service time in the global table
- On the base workspace -- finds the part instance by serial number
- For its current process step -- sets the actual start time (as "now") and actual resource (current machine)

```
get_param('RoutingTest/Machine 1', 'EntryAction')
```

```
ans =
    'coder.extrinsic('evalin')
     coder.extrinsic('strcat');
     coder.extrinsic('num2str');


     %entity.ServiceTime = serviceTime(MachineType,ProcessType,ProductType);
     entity.ServiceTime = evalin('base', strcat('serviceTime(', num2str(entity.nextNode), ...
         ', productInstances{[productIDs == ', num2str(entity.serialNumber), ']}.processPlan.currentProcessStep,',
         num2str(entity.productTypeID), ');'));
```

```
    evalin('base', strcat('part = productInstances{[productIDs == ', num2str(entity.serialNumber), ']};', ...
        'part.currentProcessStep.actualStartTime = get_param(''RoutingTest'',''SimulationTime'');', ...
        'part.currentProcessStep.actualResource =', num2str(1),';'));'
```

The Service Complete Action

- For its current process step -- sets the actual complete time (as "now")
- Update the Machine Health (calling machineHealthIndex.m)
- The machine health index function looks at the current health level, run time, and part count and compares it to max set points (essentially when the machine is broken and requires maintenance) to determine the current machine health
- Use the machine health and determine the Entity's output condition (calling probInduceDefect.m)
- Write the entity output condition to the workspace

```
get_param('RoutingTest/Machine 1', 'ServiceCompleteAction')
```

```
ans =
    'coder.extrinsic('evalin')
     coder.extrinsic('strcat');
     coder.extrinsic('num2str');

     evalin('base', strcat('part = productInstances{[productIDs == ', num2str(entity.serialNumber), ']};', ...
         'part.currentProcessStep.actualCompleteTime = get_param(''RoutingTest'',''SimulationTime'');'));

     coder.extrinsic('probInduceDefect');
     coder.extrinsic('machineHealthIndex');

     % Equipment health
     persistent healthIndex;
     persistent runTime;
     persistent partCount;
     if isempty(healthIndex)
             %Initial health level.
         healthIndex = 5;
     end
     if isempty(runTime)
         runTime = 0;
     end
     if isempty(partCount)
         partCount = 0;
     end

     runTime = runTime + entity.ServiceTime;
     partCount = partCount + 1;
     maxRunTime = 1500;
     maxPartCount = 1000;
     maxHealthIndex = 10;
     healthIndex = machineHealthIndex(1, healthIndex,maxHealthIndex, runTime, maxRunTime, partCount, maxPartCount);

     if healthIndex >= maxHealthIndex
         entity.condition = probInduceDefect(entity.condition, maxHealthIndex, entity.serialNumber);
         healthIndex = 2;
     else
         entity.condition = probInduceDefect(entity.condition, healthIndex, entity.serialNumber);
     end

     % Write part condition to part
```

```
    evalin('base', strcat('part = productInstances{[productIDs == ', num2str(entity.serialNumber), ']};', ...
        'part.actualOutputCondition = ', num2str(entity.condition),';'));

    %Write machine condition to machine
    evalin('base', strcat('machine1.machineHealth = ', num2str(healthIndex), ''));'
```

Pack Ship Close

This step closes out the task and writes completion times to the part and process plan

```
get_param('RoutingTest/PackShipClose', 'EntryAction')
```

```
ans =
    'coder.extrinsic('evalin')
     coder.extrinsic('strcat')
     coder.extrinsic('num2str')

     simTime = evalin('base', 'get_param(''RoutingTest'',''SimulationTime'');');
     evalin('base', strcat('part = productInstances{[productIDs == ', num2str(entity.serialNumber), ']};', ...
         'part.actualCompleteTime = ', num2str(simTime), ';', ...
         'part.processPlan.actualCompleteTime = ', num2str(simTime), ';', ...
         'part.currentProcessStep.actualCompleteTime = ', num2str(simTime), ';', ...
         'part.currentProcessStep.actualStartTime = ', num2str(simTime), ';', ...
         'part.currentProcessStep.typeID = ''PackShipClose'' ;', ...
         'part.actualOutputCondition = ', num2str(entity.condition,5), ';'));

         evalin('base', strcat('part = productInstances{[productIDs == ', num2str(entity.serialNumber), ']};', ...
         'part.currentProcessStep.actualCompleteTime = get_param(''RoutingTest'',''SimulationTime'');'));'
```

Routing Decision Support

The routing decision support is going to determine the next node that the task should go to. This based on both *dynamic process planning* and *resource assignment*

```
get_param('RoutingTest/Routing Decision Support', 'EntryAction')
```

```
ans =
    'coder.extrinsic('evalin')
     coder.extrinsic('strcat')
     coder.extrinsic('num2str')


     entity.nextNode = evalin('base', strcat('RoutingController.Routing.Routing(productInstances{[productIDs == ',
```

The block delegates to the Routing Controller object on the base workspace: (RoutingController.Routing.Routing). The Controller has a property for holding Routing decision support class objects, which implement a uniform interface called "Routing".

```
open('Controller.m')
open('Routing.m')
```

Controller

The controller has three additional functions (aside from providing access to the decision support)

- Create Product and Process listeners
- Publish the "assets" -- units of information
- The Listeners: The simulation writes current values to objects in the workspace, the controller listens to changes to these values and then publishes the change to the "asset buffer"
- Publish the Assets: The publish function adds a timestamp, uuid , a typeID, and sequence number to the record and then stores them in a list.

Run Simulation

Note: don't run the simulation in the live editor without supressing the output. The data stream causes the editor to crash.

```
evalc('sim(''RoutingTest'', ''StopTime'', ''500'');');
```

Queries

The cellfun and strcmp functions allow you to search the AssetBuffer which is a cell array of structs. The structs are all different types (product, process, resource). I'd suggest filtering by Asset typeID first because if you search for a field that doesn't exist, it'll error out.

```
machineRecords = strcmp(cellfun( @(sas) sas.typeID, RoutingController.AssetBuffer, 'uni', false
machineRecords = RoutingController.AssetBuffer(machineRecords);
```