FLEX SCRIPT FROM BED LOCATION CONFIGURABLE ENTRANCE CRITERIA

```
////////////////////////////////////// CODE HEADER
////////////////////////////////////////////////////////////////////////
treenode current = ownerobject(c);
treenode patient = parnode(1);
int port = parval(2);
int switchcase = parval(3);
////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////

/**Configurable Entrance Criteria*/

// Macros defined for easy reference to this object and its area as a number
#define THIS_LOC tonum(current)
#define THIS_AREA executefsnode(getvarnode(current, VAR_AreaName),NULL)

// A variable set to the destination of the patient evaluated in the code below (an area rank or
numeric pointer to a location).
double PATIENT_DEST = 0;
// A variable that will be set to 1 if the patient being evaluated was assigned to come to either
this location or this location's area.
int ASSIGNED_HERE = 0;
int NOT_USED = VERYSMALLNUMBER;

/**\n\nEntrance Criteria: */
#define SELECTION_CRITERIA /**/ASSIGNED_HERE/**list:ASSIGNED_HERE~getlabel(patient, "PCI") >
3~getcensus(otherArea) < 20~ASSIGNED_HERE && getlabel(patient, "PCI") == 3*/

/**\n\nSelection Priority 1: */
#define SELECTION_PRIORITY1 /**/getlabel(patient, "a_EMS_Pt") == 1
/**list:NOT_USED~getlabel(patient, "Acuity")~time() - get(stats_lastmovetime(patient))~time() -
getcreationtime(patient)~time() - getvarnum(patient, VAR_LastReleaseTime)*/

/**\n\nSelection Priority 2: */
#define SELECTION_PRIORITY2 /**/getlabel(patient,
"a_Rm_Priority")/**list:NOT_USED~getlabel(patient, "Acuity")~time() -
get(stats_lastmovetime(patient))~time() - getcreationtime(patient)~time() - getvarnum(patient,
VAR_LastReleaseTime)*/

/**\n\nSelection Priority 3: */
#define SELECTION_PRIORITY3 /**/time() -
get(stats_lastmovetime(patient))/**list:NOT_USED~getlabel(patient, "Acuity")~time() -
get(stats_lastmovetime(patient))~time() - getcreationtime(patient)~time() - getvarnum(patient,
VAR_LastReleaseTime)*/

/**\n\nIn order for a patient to be accepted into this location, they must first pass the selection
criteria. Those patients passing the entrance criteria, will be accepted according to their
priority as defined by the three selection priority values specified. Selection priority 1 takes
precedence over selection priority 2 which takes precedence over selection priority 3. Enter
"NOT_USED" to remove any one of the selection priorities.*/

// This "Entrance Criteria" function has two cases in which it gets executed: (1) the "Pull From
Port" case
// and (2) the "Pull Requirement" case.

// The 1st case gets executed when this object is ready to receive another patient. It will usually
look
// through each upstream object connected to this object's inputs, and return the input port number
of the
// object containing a patient of choice.

// The 2nd case will be executed for each released patient in the upstream object connected to the
input port
// returned by the "Pull From Port" case.  If the "Pull From Port" case returns a 0, then all input
ports
// would have been opened, and the "Pull Requirement" case will be executed for each released
patient starting
// with patients in objects connected to the first input port and continue looking through all
objects until
```

```
// a patient is found matching the pull requirement.  This 2nd case returns a true/false (1/0)
based on whether
// the patient matches the requirement.  When a patient is found matching the requirement,
execution stops and
// the patient is accepted.  This case is executed immediately following the execution of the 1st
case, and
// continues to be executed until a 1 is returned. This case is also executed each time a new
patient is released
// by one of the upstream objects that is connected to an input port previously opened by the 1st
"Pull From Port" case.

treenode patientdestinationnode = NULL;
treenode specificlocation = NULL;

// Objects have a variable named ReservedForPatient which stores a pointer to the next patient the
object will
// receive. PatientProcessing objects will also use this variable to remember the patient that the
room is
// being held for, the variable is most often used to temporarily store a reference to a patient
chosen during
// execution of the "Pull From Port" case used by subsequent calls of the "Pull Requirement" case.

treenode reservedforpatient = getvarnode(current, VAR_ReservedForPatient);
treenode reservedpatient = tonode(getnodenum(reservedforpatient));

if(switchcase==1) // "Pull From Port" case
{
	// If the "reservedforpatient" was not set by an activity requiring that the patient hold the
room while away,
	// then find the patient best matching the criteria, and set that patient as the reserved
patient so the
	// "Pull Requirement" will know which patient to pull based on the logic of this Pull From
Port switch case.
	if(!objectexists(reservedpatient) && inputopen(current))
	{
		treenode location = NULL;
		treenode patient = NULL;
		treenode chosenpatient = NULL;
		double priority1Max = VERYSMALLNUMBER;
		double priority2Max = VERYSMALLNUMBER;
		double priority3Max = VERYSMALLNUMBER;
		int chosenportnum = 0;
		int j = 1;
		int k = 1;

		// Look through upstream objects
		for(j = 1; j <= nrip(current); j++)
		{
			location = inobject(current, j);
			// If upstream object contains patients, has its output port open, has not been
closed with closeoutput() command
			if(
				content(location) > 0 &&
				opopen(location, ipopno(current, j)) &&
				get(connectionsout(location))==0
			)
			{
				// Search through the released patients assigned to come to this
location's area or specifically to this location,
				// and choose the one who has been waiting the longest of those who have
the highest acuity.
				for(k = 1; k<=content(location); k++)
				{
					patient = rank(location, k);
					int reeval = 0;
					patientdestinationnode = getvarnode(patient,
VAR_PatientDestination);
					PATIENT_DEST = getnodenum(patientdestinationnode);
					// If this patient has been assigned 0 for its patient
destination, it is because at the time the activity first started, an available
```

```
                                // area wasn't found, so I will reevaluate the PatientDestination
nodefunction for the patient's active transfer activity to see if it
                                // can now be resolved to an available area. The reason we open
the input of this object that is connected to the object
                                // with the candidate patient is because at this point all this
object's input ports are closed, and yet the PatientDestination function's
                                // decision might be dependent upon what is open at the moment.
                                if(PATIENT_DEST == 0)
                                {
                                        treenode activitytable = getvarnode(patient,
VAR_ActivityTable);
                                        int curactivityrow = getvarnum(patient, VAR_ActivityRow);
                                        setnodenum(first(rank(connectionsin(current),j)), 1);
//openip
                                        treenode patientdestinationfunction =
gettablecell(activitytable, curactivityrow, COL_PatientDestination);
                                        PATIENT_DEST = nodefunction(patientdestinationfunction,
tonum(patient), curactivityrow, current); // Passing current tells the PatientDestination function
that it is being reevaluated, not really called
                                        setnodenum(patientdestinationnode, PATIENT_DEST);
                                        reeval = 1;
                                        setnodenum(first(rank(connectionsin(current),j)), 0);
//closeip
                                }
                                // The subnode of the patient's PatientDestination variable will
hold a reference to a specific location if some location is being held
                                // for the patient to return to. The specificlocation reference is
set at the time the activity for transfering the patient out
                                // of the reserved room was first started, and it is reset to 0
when the patient reenters the reserved room. It is possible that
                                // the patient under consideration has a specificlocation defined,
but because at this point in the code we know that this location
                                // doesn't have a reference to a patient in its ReservedForPatient
variable, we know that the candidate patient's specificlocation
                                // should not be a reference to this location. It's important to
also check that the area of the specificlocation object for the
                                // candidate patient is not the same as the area of this location,
because in the case where a patient is returning to this area,
                                // we only want the patient to go to the location within the area
that they reserved previously.
                                specificlocation = tonode(get(first(patientdestinationnode)));
                                ASSIGNED_HERE = PATIENT_DEST == THIS_AREA || PATIENT_DEST ==
THIS_LOC;
                                // If patient matches criteria for consideration
                                if(
                                        getitemstate(patient) == FRSTATE_READY && //patient has
been released to travel to their next destination by one of the three transfer activities
                                        (!objectexists(specificlocation) ||
executefsnode(getvarnode(specificlocation, VAR_AreaName),NULL) != THIS_AREA || specificlocation ==
current) && //if patient has a reserved room in this area, it must match this room
                                        SELECTION_CRITERIA
                                )
                                {
                                        double priority1Value = SELECTION_PRIORITY1;
                                        // If the patient beats or matches the P1 max...
                                        if(priority1Value >= priority1Max)
                                        {
                                                // If the patient beats the previous P1 max...
                                                if(priority1Value > priority1Max)
                                                {
                                                        // Update P1 max
                                                        priority1Max = priority1Value;
                                                        // Update P2 max
                                                        priority2Max = SELECTION_PRIORITY2;
                                                        // Update P3 max
                                                        priority3Max = SELECTION_PRIORITY3;
                                                        // Record current port and patient as the
chosen ones so far
                                                        chosenportnum = j;
                                                        chosenpatient = patient;
```

```
                                                }
                                                // Else the patient matches the current P1 max, so
start comparing P2
                                                else
                                                {
                                                       double priority2Value = SELECTION_PRIORITY2;
                                                       // If the patient beats or matches the P2
max...
                                                       if(priority2Value >= priority2Max)
                                                       {
                                                              // If the patient beats the previous P2
max...
                                                              if(priority2Value > priority2Max)
                                                              {
                                                                     // Update P2 max
                                                                     priority2Max = priority2Value;
                                                                     // Update P3 max
                                                                     priority3Max =
SELECTION_PRIORITY3;
                                                                     // Record current port and
patient as the chosen ones so far
                                                                     chosenportnum = j;
                                                                     chosenpatient = patient;
                                                              }
                                                              // Else the patient matches the current
P2 max, so start comparing P3
                                                              else
                                                              {
                                                                     double priority3Value =
SELECTION_PRIORITY3;
                                                                     // If the patient beats the
previous P3 max...
                                                                     if(priority3Value > priority3Max)
                                                                     {
                                                                            // Update P3 max
                                                                            priority3Max =
priority3Value;
                                                                            // Record current port and
patient as the chosen ones so far
                                                                            chosenportnum = j;
                                                                            chosenpatient = patient;
                                                                     }
                                                              }
                                                       }
                                                }
                                         }
                                  }

                                  if(reeval == 1)
                                  {
                                         setnodenum(patientdestinationnode, 0);
                                  }
                           }
                    }
             }
             // If a candidate patient was found, record it, and set its rank to 1, so that it will
be evaluated
             // first by the Pull Requirement case so as to speed up program execution.
             if(chosenportnum > 0)
             {
                    treenode callbacks = node(">callbacks", current);
                    if(objectexists(callbacks))
                    {
                           for(int j = 1; j <= content(callbacks); j++)
                           {
                                  if(node("+/~", rank(callbacks, j)) == chosenpatient)
                                  {
                                         nodefunction(node("+", rank(callbacks, j)),
tonum(chosenpatient), getrank(up(node("+", rank(callbacks, j)))), current, 1);
                                         break;
```

```
                                    }
                              }
                              destroyobject(callbacks);
                        }

                        set(reservedforpatient, tonum(chosenpatient));
                        setrank(chosenpatient, 1);
                        // At this point in the code, if patientdestination is 0 for the chosen
patient, then initially it was 0 causing
                        // it to be reevaluated with a valid reference to this location/area and the
patient was of course
                        // chosen, but because I reset the patientdestination node of any patients that
were initially 0 back to 0 at the
                        // end of the above for loop, I need to set it once again to this area at this
time, so it will pass the
                        // pending pullrequirement check.
                        if (getvarnum(chosenpatient, VAR_PatientDestination) == 0)
                        {
                              // I'm setting patientdestination to this area because I know that it is
going this location since
                              // it is the one that just became available.
                              setvarnum(chosenpatient, VAR_PatientDestination, THIS_LOC);

                              // In cases like this where the Patient Destination field originally
returned a zero indicating
                              // no selection and the need to reevaluate the field later, it is
necessary to update the number of patients
                              // assigned to the location/area now since it would not have been done at
the time the field was first
                              // evaluated, and a destination chosen. Since I now know that the chosen
patient will be going to this object,
                              // I can increment the assigned census for both this location and this
location's area.
                              treenode bundlelist = getvarnode(OutputObject,VAR_DataBundles);
                              // Increment the destination area's assigned census
                              treenode bundle = rank(bundlelist, RANK_AreaCensusByPCI);
                              int pci = getlabel(chosenpatient, LABEL_PCI);
                              // for chosen patient PCI
                              int prevcensus = getbundlevalue(bundle, pci, FIELD_CensusAssigned + 3 +
(3 * (THIS_AREA - 1)));
                              setbundlevalue(bundle, pci, FIELD_CensusAssigned + 3 + (3 * (THIS_AREA -
1)), prevcensus + 1);
                              // total
                              prevcensus = getbundlevalue(bundle, 0, FIELD_CensusAssigned + 3 + (3 *
(THIS_AREA - 1)));
                              setbundlevalue(bundle, 0, FIELD_CensusAssigned + 3 + (3 * (THIS_AREA -
1)), prevcensus + 1);
                              // Increment this location's assigned census
                              treenode locbundlelist = getvarnode(current, VAR_LocDataBundles);
                              bundle = rank(locbundlelist, RANK_LocCensusByPCI);
                              // for chosen patient PCI
                              prevcensus = getbundlevalue(bundle, pci, FIELD_CensusAssigned);
                              setbundlevalue(bundle, pci, FIELD_CensusAssigned, prevcensus + 1);
                              // total
                              prevcensus = getbundlevalue(bundle, 0, FIELD_CensusAssigned);
                              setbundlevalue(bundle, 0, FIELD_CensusAssigned, prevcensus + 1);
                        }
                        else if (getvarnum(chosenpatient, VAR_PatientDestination) == -1)
                              setvarnum(chosenpatient, VAR_PatientDestination, 0);
                  }
                  // Return the input port number to the upstream object containing the patient. If no
patient was found,
                  // then chosenportnum will equal 0 which signals this location to open all its input
ports and simply wait.
                  return chosenportnum;
            }
            // Otherwise the room is being held for a reserved patient already, so return 0 to open all
inputs, and either
            // wait and/or let the Pull Requirement case handle it bringing in the correct patient
specified by the reservation.
```

```
	return 0;
}
else // "Pull Requirement" case
{
	int requirement = 0;
	patientdestinationnode = getvarnode(patient, VAR_PatientDestination);
	PATIENT_DEST = getnodenum(patientdestinationnode);
	ASSIGNED_HERE = PATIENT_DEST == THIS_AREA || PATIENT_DEST == THIS_LOC;
	// If this object has been reserved for a specific patient, then it is of course necessary
that the patient being considered be the patient
	// this object has been reserved for. This should be true in the situation where the
pullrequirement case fires immediately after the
	// pullfromport case fires (because the reservedpatient gets set with the patient chosen
during the pullfromport case), and in the situation
	// where the patient for whom a room is reserved is attempting to return. The patient's
PatientDestination is also checked because the patient for whom
	// this object is being reserved for may be released in one of this object's upstream
objects, but is meant to go to some other area at this time.
	if(objectexists(reservedpatient))
	{
		requirement = ASSIGNED_HERE && patient == reservedpatient;
		// Regardless of whether the ReservedForPatient variable of this location was set by a
room reservation or the "Pull From Port" case,
		// it can now safely be reset to 0. This is because if it was a room reservation, the
patient it was reserved for is now returning;
		// and if it was set by the "Pull From Port" case, then it means the room never had a
reservation, and therefore the ReservedForPatient
		// variable was simply used to temporarily record the patient chosen for entry in the
"Pull From Port" case.
		if(requirement == true)
			set(reservedforpatient, 0);
	}
	else
	// If this object has NOT been reserved for a specific patient, then it means that the Pull
From Port returned a zero due to no patients
	// meeting the requirement in any of the upstream objects at the time the Pull From Port
fired, and instead a new patient has just been
	// released inside one of the upstream objects and now needs to be evaluated for allowed
entrance into this location. The first requirement
	// for entry is that the specificlocation recorded on a subnode of the PatientDestination
variable when patients begin their transfer
	// activity out of a room they need to reserve must either be a reference to this exact
location, or else not reference any other locations
	// within this location's same area.
	{
		specificlocation = tonode(get(first(patientdestinationnode)));
		requirement = (!objectexists(specificlocation) ||
executefsnode(getvarnode(specificlocation, VAR_AreaName),NULL) != THIS_AREA || specificlocation ==
current)
					&&
					SELECTION_CRITERIA;
	}
	return requirement;
}
```