STEPS TO CONVERT TO A SHEDMASTER-COMPATIBLE INDUSTRY

For ease of reference, industries that would benefit from conversion to SM compatibility where the train locomotive remains attached for the duration of the loading or unloading process, are referred to as "ordinary SM-compatible industries" and those where trains are to be loaded and/or unloaded in intervals without the train locomotive attached are referred to as "shunted SM-compatible industries".

Loading and unloading in Trainz is done in the blink of an eye. In many cases this is a gross distortion of the time taken on the real railway. The objective of this conversion is to more closely mimic the interaction between the real railway and certain industries.

IDENTIFYING CANDIDATE SHUNTED SHEDMASTER-COMPATIBLE INDUSTRIES

Many real world industries did and do have quick methods of loading and unloading. For instance, loading bulk commodities from an overhead bin and unloading hopper wagons to a below-track bin are quite quick. The train may not even need to stop for these processes to occur. Industries that can be modelled with loading and unloading that occur on the move can be converted to ordinary SM-compatible industries, but are not candidates for conversion to shunted SM-compatible industries.

An industry with stopped loading and unloading may be suitable for conversion to a shunted SM-compatible industry. Best candidates amongst such industries are those where the transfer of goods between rail wagons and the industry bays is slow in the real world. A good example is where the unloading and loading of vans or box cars occured by hand. This could easily have taken an hour or more in the real world. In these situations it could be inefficient to leave a shunting locomotive attached to the train at the industry bay. The shunter could uncouple and go elsewhere to be productive, then come back to this train when it needs to moved so that other wagons in the train are brought into the industry bay. Of course, it may be that there is no other productive work the shunting locomotive could perform in the immediate area. It is not just the nature of the industry bay that determines if conversion to a shunted SM-compatible industry is worthwhile. The context in which the industry operates is also important. Are there other industries in the vicinity that could share the shunting locomotive? Or are there multiple loading and unloading bays within the one industry complex with ongoing shunting requirements? A steel works would a good example of the latter.

Once an industry asset is converted for SM compatibility, the use the Shedmaster rule to meet the industry's needs is not enforced. The conversion is an added layer to the original industry script and the user chooses whether each deployed instance of an industry is to operate with or without Shedmaster.

WHAT IS THE DIFFERENCE BETWEEN CONVERTING TO AN ORDINARY AND SHUNTED SHEDMASTER-COMPATIBLE INDUSTRY?

The built-in Trainz industry processing scripts assume that industry loading and unloading undertaken using the LOAD and UNLOAD driver commands will proceed until the entire train has been processed with the locomotive remaining attached to the train. In order to convert from an ordinary to a shunted SM-compatible industry, the LOAD and UNLOAD commands must be replaced, and the industry script must have additional functions and variables, above those required for an ordinary SM-compatible industry.

'Shedmaster Command',<kuid2:160293:100114:3> contains "Load at industry" and "Unload at industry" commands in its menu. These should be used in place of the "Load" and "Unload" commands in the schedules of AI drivers assigned by SM to perform tasks at shunted SM-compatible industries.

STEPS TO CONVERT AN INDUSTRY CONFIG FILE

Items in blue text are additional items required by shunted Shedmaster-compatible industries.

1. Add the suffix "_SM" to the 'username' tag to signify the asset is Shedmaster compatible.

2. Add the following sentence to the 'description' tag: "This industry is compatible with the Shedmaster rule <kuid:160293:10065>."

3. Add an 'extensions' container, if one does not already exist.

4. Add the following service request tags to the extensions container:

   num_msgs "<n>" (where 'n' is the total number of messages)

   .

   For a loading bay:

   msg-i "Deliver Empty Wagons*-<product name j>"

   msg-i+1 "Shunt-<product name j>"

   msg-i+2 "Collect Loaded Wagons-<product name j>"

   .

   For an unloading bay:

   msg-i ' "Deliver Loaded Wagons*-<product name k>"

   msg-I'+1 "Shunt-<product name k>"

   msg-I'+2 "Collect Empty Wagons-<product name k>"

   .

   NOTE: The word 'Wagons' above is optional. If the product name is long then the

optional word could be omitted to save space in the SM configuration window.
Generally there will be one product per industry bay, so for an ordinary Shedmaster-compatible industry, the number of messages, n, will equal the number of bays (or the number of products, if this is larger), whereas for a shunted Shedmaster-compatible industry, n will equal 3 x the number of bays (or the number of products, if this is larger).

STEPS TO CONVERT AN INDUSTRY SCRIPT

Items in blue text are additional items required by shunted Shedmaster-compatible industries.

1. DECLARE ADDITIONAL GLOBAL VARIABLES

For each loading and unloading bay of an industry, there will be a set of additional global variables that must be declared at the start of the script. For ensuring that saved sessions restore correctly, many of these global variables must be saved and restored from the session soup.

SM_comms_OFF The user can enable and disable Shedmaster mode for a converted industry by toggling the Boolean 'SM_comms_OFF' accessed via Surveyor > Edit Properties.

runningDriver ModuleInitHandler, a handler that receives the broadcast message <"World", "ModuleInit"> should run once when Driver mode is first entered to enable evaluation of a number of other global variables required by Shedmaster. "runningDriver" is a Boolean used within ModuleInitHandler to abort the handler when secondary messages are received.

noAKNcnt is an integer array that records the number of times there has been no acknowledgement from Shedmaster of a service request. Unacknowledged requests are repeated at a much longer interval once noAKNcnt reaches a pre-set number of times.

ext Soup in which to read key Shedmaster terms/variables placed in the config file's 'extensions' container.

ST StringTable that mirrors the content of the config file's 'string-table' container (the industry script may already have an equivalent variable).

There may be multiple unloading (In) and loading (Out) bays within the one industry. The following global variables have an integer 'I' suffix added to their names since there needs to be one or other of them for each bay.

SR_L<i>_Bar, **SR_U<i>_Bar** Once a service request for the i-th industry bay has been issued, no further requests should be allowed until the outstanding request is completed. These Boolean variables indicate whether service requests for each bay are currently allowed to proceed.

SR_InFlag_<i>, **SR_OutFlag_<i>** These Booleans are 'true' when there is an outstanding service

request of any kind that services the i-th bay.

SR_InTrack_<i>, **SR_OutTrack_<i>** Boolean arrays with one array element to record the status of each of 4 stages in the service request directed at the i-th bay. Index 0 indicates whether the bay's track is occupied by wagons. Index 1 indicates whether a service request for more wagons to be brought to the bay is outstanding. Index 2 indicates whether a service request to shunt wagons at the bay is outstanding. Index 3 indicates whether a service request to remove wagons from the bay is outstanding.

numCarsL<i>bay, **numCarsU<i>bay** For each shunting movement to bring new wagons in a train to the i-th industry bay, a certain number of the new wagons at the bay will be able to load or unload at the bay. This integer is recorded in these variables.

numCarsLoaded_<i>, **numCarsUnloaded_<i>** Adjunct to the above variables. As the processing of each wagon at the i-th bay is completed, this count integer is incremented.

consistL<i>, **consistU<i>** The identity of a consist that is attending each industry bay is recorded by these type 'Train' variables

consistL<i>_DN, **consistU<i>_DN** The variables immediately above must be recoverable should a session be saved at a point where loading or unloading is ongoing. These string variables enable storage of the consist's display name in the session soup, from which the Train variables can be recovered when the session is restarted.

Last_Loaded_Vehicle<i>, **Last_Unloaded_Vehicle<i>** are global Vehicle types used to determine if a train has completed loading or unloading so that the i-th industry bay is free to accept another train.

LLV_Name<i>, LUV_Name<i> To support the saving and reloading of sessions, the name string of the 2 global variables listed immediately above are stored to the session soup.

carsToLoad<i>, **carsToUnload<i>** Vehicle arrays that store the identities of those vehicles currently at the i-th industry bay and that can be serviced by this bay.

triggerL<i>, **triggerU<i>** String arrays that record the names of the triggers on which stand each of the vehicles in the Vehicle arrays immediately above.

2. DECLARE ADDITIONAL FUNTIONS AND METHODS

void CheckQueuesHandler (Message msg); Performs routine checking of the industry queues to determine if a service request to Shedmaster is required.

void SendServiceRequest(int messageNumber); Posts a message to Shedmaster that opens a service request.

thread void Load<i>AtShedmasterIndustry(Train myTrain); Shepherds an arriving train to a

stop at the i-th industry loading bay, uncouples the locomotive and restarts its schedule, then enters into a loop to call **PerformSMload<i>** for each wagon at the bay that can load.

void PerformSMload<i>(Vehicle vehicle,string triggerName); Transfers loads from the i-th industry loading bay to 'vehicle'.

thread void Unload<i>AtShedmasterIndustry(Train myTrain); Shepherds an arriving train to a stop at the i-th industry unloading bay, uncouples the locomotive and restarts its schedule, then enters into a loop to call **PerformSMunload<i>** for each wagon at the bay that can unload.

void PerformSMunload<i>(Vehicle vehicle,string triggerName); Transfers loads from 'vehicle' to the i-th industry unloading bay.

3. DEFINE SHEDMASTER MESSAGE HANDLERS in the 'INIT' method

```
AddHandler(me, "World", "ModuleInit", "ModuleInitHandler");
AddHandler(me, "Check_Queues", "", "CheckQueuesHandler");
AddHandler(me, "Service_Acknowledgement", "", "SMHandler");
 AddHandler(me, "Service_Rejection", "", "SMHandler");
 AddHandler(me, "SR_NoAKN", "", "SMHandler");
 AddHandler(me, "SR_L<i>_Bar", "", "SMHandler");    // 1 handler per loading bay
 AddHandler(me, "SR_U<i>_Bar", "", "SMHandler");    //1 handler per unloading bay
 AddHandler(me, "Object", "", "ObjectHandler");
```

4. ADD METHODS TO SUPPORT HANDSHAKING WITH SHEDMASTER

Suggested code:

```
Soup ext; // extensions soup in which to store key SM terms/variables
int shedmasterId=0;                    //ID of the service centre for this industry
int numMsgs;

void SMHandler(Message msg) {
   int i;
   // Compare SM ID with recorded ID
   //    Compare message with service request. Record SM ID if not yet done
   if (shedmasterId < 1) {
        GameObject src = msg.src;// get the ID of the Shedmaster asset sending the
acknowledgement
        shedmasterId = src.GetId(); // use this destination in future messages
   }
   if (msg.major == "Service_Acknowledgement") {
      //    Compare service requested with that acknowledged by SM
```

```
        for (i=1;i<=numMsgs;i++) {
                if (ext.GetNamedTag("msg-"+i) == msg.minor) { // Yes, this is one of our
messages
                ClearMessages("SR_NoAKN", (string)i);
                noAKNcnt[i] = 0;
        // there should be one switch case for each message defined in the config file's
        // 'extensions' container
                switch (i) {
        .
        .
        .
        // typical loading bay cases
                case l': //Loading_Wagons-<product name j>
                        SR_OutTrack_<k>[1] = true;
                        SR_OutFlag _<k>= false;
                        break;
                case l'+1: //Loading_Shunt-<product name j>
                        SR_OutTrack[2] = true;
                SR_OutTrack[1] = false;
                AILoad = true;
                for (i=0;i<5;i++) {carsToLoad[i] = null;}
                        break;
                case l'+2: //Loading_Complete-<product name j>
                        SR_OutTrack[3] = true;
                SR_OutTrack[2] = false;
                        break;
        .
        .
        .
        // typical unloading bay cases
                case l": //Unloading_Wagons-<product name j'>
                        SR_InTrack[1] = true;
                        SR_InFlag = false;
                        break;
                case l"+1: //Unloading_Shunt-<product name j'>
                SR_InTrack[2] = true;
                        SR_InTrack[1] = false;
                        AIUnload = true;
                        for (i=0;i<5;i++) {carsToUnload[i] = null;}
                        break;
                case l"+2: //Unloading_Complete-<product name j'>
                SR_InTrack[3] = true;
```

```
                    SR_InTrack[2] = false;
                    break;
          .
          .
          .
                default:
                                    Interface.Print("Industry " + me.GetName() + " has
received this unknown message from Shedmaster: " + msg.minor);
                }
                break;
                }
             }
          }
          else if (msg.major == "SR_U1_Bar") {
           SR_U1_Bar = false;
          }
.         // repeat for U2,U3… if applicable
.
.
          else if (msg.major == "SR_L1_Bar") {
           SR_L1_Bar = false;
          }
.         // repeat for L2,L3… if applicable
.
.
          else if (msg.major == "Service_Rejection") {
             Interface.Print("Industry " + me.GetName() + " has requested Shedmaster
service '" + msg.minor + "' but the request has been rejected.");
          }
          else if (msg.major == "SR_NoAKN") { // Acknowledgement has not been received.
Re-send the message.
           int messageNumber = Str.ToInt(msg.minor);
           SendServiceRequest(messageNumber);
           noAKNcnt[messageNumber]++;
          }
       }

       //    Send router message requesting service
       //    If Shedmaster address known, sends message to Shedmaster for this industry
       //    Else sends broadcast message.
       //    Param "messageNumber" is the suffix of the msg tag in the config file
      //    Returns true if message sent, or false if message already acknowledged
```

```
    void SendServiceRequest(int messageNumber) {
       string message = ext.GetNamedTag("msg-"+messageNumber);
       int myDelay;
       if (shedmasterId < 1) PostMessage(null,"Service_Request",message,0.0);
       else
Router.PostMessage(me.GetId(),shedmasterId,"Service_Request",message,0.0);
       if (noAKNcnt[messageNumber] > 2) myDelay = 300.0;
       else myDelay = 10.0;
       PostMessage(me,"SR_NoAKN",(string)messageNumber,myDelay);
    }

    void ObjectHandler(Message msg) {
        if (msg.minor == "Leave" and ( SR_OutTrack[3] or SR_InTrack[3]) ) {
            if (msg.src != null) {
                Vehicle departingVehicle = cast<Vehicle>msg.src;
                if (departingVehicle == Last_Loaded_Vehicle) { // The consist
loading at the OutTrack has departed
                    SR_OutTrack[0] = false;
                    SR_OutTrack[3] = false;
                }
                else if (departingVehicle == Last_Unloaded_Vehicle) { // The
consist unloading at the InTrack has departed
                    SR_InTrack[0] = false;
                    SR_InTrack[3] = false;
                }
            }
        }
    }
```

5. ADD OR MODIFY 'MODULEINITHANDLER' METHOD

A third colour, light gray, is introduced in the typical method outlined below to indicate code that could be present before the industry is modified for Shedmaster compatibility.

```
    thread void ModuleInitHandler(Message msg) {
        if (RunningDriver) {
            return;
        }
        if (World.GetCurrentModule() == World.DRIVER_MODULE) {
            RunningDriver = true;
            meAsset = me.GetAsset();
            ST = meAsset.GetStringTable();
```

```
                ext = meAsset.GetConfigSoup().GetNamedSoup("extensions");
                numMsgs = Str.ToInt(ext.GetNamedTag("num_msgs"));
                noAKNcnt = new int[numMsgs + 1];
                <product name 1>Filter = Constructors.NewProductFilter();
                Asset myProduct = meAsset.FindAsset("<product name 1>");
                <product name 1>Filter.AddProduct(myProduct);
                <product name 2>Filter = Constructors.NewProductFilter();
                myProduct = meAsset.FindAsset("<product name 2>");
                <product name 2>Filter.AddProduct(myProduct);
.
.
.
                if (!SM_comms_OFF and carsToUnload[0] != null) {
                        UnloadAtShedmasterIndustry(carsToUnload[0].GetMyTrain());
                        carsU = carsToUnload[0].GetMyTrain().GetVehicles();
                }
                if (!SM_comms_OFF and carsToLoad[0] != null) {
                        LoadAtShedmasterIndustry(carsToLoad[0].GetMyTrain());
                        carsL = carsToLoad[0].GetMyTrain().GetVehicles();
                }
        }
    }
```

6. ADD ‘CHECKQUEUESHANDLER’

```
    void CheckQueuesHandler(Message msg) {
      if (!SR_InTrack[1] and !SR_InTrack[0]) {
        float <input product name>QueueStore = (float)<input product name
>InQueue.GetQueueCount() / (float)<input product name>InQueue.GetQueueSize();
        if (<input product name>QueueStore < 0.25) { // If input in store has fallen below
25% capacity
          SendServiceRequest(4);  //  "Unloading_Wagons"
        }
      }
      if (!SR_OutTrack[1] and !SR_OutTrack[0]) {
        float <output product name>QueueStore = (float) <output product
name>OutQueue.GetQueueCount() / (float) <output product
name>OutQueue.GetQueueSize();
        if (<output product name>QueueStore > 0.75) {// If an output has risen above 75%
capacity
          SendServiceRequest(1);  //  "Empty_Wagons"
        }
```

```
        }
        PostMessage(me,"Check_Queues","",300.0);     // Recheck queues status every 5 minutes
    }
```

7. MODIFY FUNCTION 'TriggerSupportsStoppedLoad'

For simplicity, the example is for an industry with one loading bay and one unloading bay. The loading bay has 3 triggers and the unloading bay has 4 triggers.

```
  bool TriggerSupportsStoppedLoad(Vehicle vehicle, string triggerName) {
    int indexEndL, indexEndU, i, j, k;
    Train myTrain = vehicle.GetMyTrain();
    string comd, myMsg;
    if (triggerName == "loading0" or triggerName == "loading1"    or triggerName == "loading2") {
            if (!SM_comms_OFF and SR_OutTrack[3]) return false;    // Shedmaster insert
            carsL = myTrain.GetVehicles();
            if (initialL) {
                if (SM_comms_OFF) goto INI1;
                else goto INI2;
          }
            if (SM_comms_OFF and (LastTrainLId != myTrain.GetId() or carsL.size() != Last_SizeL or carsL[0] != Last_Lead_VehicleL)) {
INI1:           LastTrainLId = myTrain.GetId();
                  Last_SizeL = carsL.size();
                  Last_Lead_VehicleL = carsL[0];
                  initialL = false;
                if(triggerName == "loading0") targetTriggerL = "cheese_loading2";
                else targetTriggerL = "loading0";
                comd = itc.GetTrainCommand(myTrain);
                  AILoad = (comd == "load" or comd == "LOAD");
                  return false;
            }
            else if (!SM_comms_OFF and SR_OutTrack[1] and !SR_OutFlag and LastTrainLId != myTrain.GetId()) {
INI2:           LastTrainLId = myTrain.GetId();
                initialL = false;
                if(triggerName == "loading0") targetTriggerL = "loading2";
                else targetTriggerL = "loading0";
                AILoad = true; // SM only works with trains under AI control
                for (i=0;i<5;i++) {carsToLoad[i] = null;}
                  return false;
```

```
    }
    canLoad = CanLoadUnload(vehicle, <OutQueue>, true);
    if (AILoad) {
        if (triggerName == targetTriggerL) {
            if (!SM_comms_OFF) {
                // Determine that currrent command is SMC LOAD
                DriverCharacter myDriver = myTrain.GetActiveDriver();
                DriverCommands mySchedule = myDriver.GetDriverCommands();
                DriverScheduleCommand[] myCommands = mySchedule.GetDriverScheduleCommands();
                string myCurrentCommand = myCommands[0].GetTooltip();
                if (!WordFound(myCurrentCommand, "Load at Industry", false)) return false;
                if (canLoad and (SR_OutTrack[1] or SR_OutTrack[2])) {
                    SR_OutFlag = true;
                    myDriver.SetRunningCommands(false); // disable schedule
                    mySchedule.ProceedToNextCommand();
                    myTrain.StopTrainGently();
                    carsToLoad[0] = vehicle;
                    LoadAtShedmasterIndustry(myTrain);
                    return false;
                }
            }
            else {
                if (canLoad) return true;
                IsEndL = IsEndCar(vehicle, "out_track", carsL, <OutQueue>);
                if (IsEndL) {
                    PostMessage(myTrain, "HandleTrain", "Release", 1.0);
                    AILoad = false;
                    if (!SM_comms_OFF) SR_OutTrack[1] = false;
                }
            }
        }
    }
    if (myTrain.IsStopped()) {
        if (canLoad) {
            if (SM_comms_OFF) return true;
            else {    // SM is active. Accumulate list of vehicles that can be loaded
                bool stopLoopL;
```

```
                                for (i=0;i<5;i++) {
                                        if (carsToLoad[i] == null) {
                                                for (j=0;j<carsL.size();j++) {
                                                        if (carsL[j] == vehicle) {
                                                                if (i>0) {    // prevent multiple entries
of the same vehicle
                                                                        stopLoopL = false;
                                                                        for (k=0;k<i;k++) {
                                                                                if (carsToLoad[k]==vehicle)
{
                                                                                        stopLoopL = true;
                                                                                        break;
                                                                                }
                                                                        }
                                                                if (!stopLoopL) {
                                                                        carsToLoad[i] = vehicle;
                                                                        triggerL[i] = triggerName;
                                                                }
                                                                break;
                                                        }
                                                }
                                                break;
                                        }
                                }
                        }
                }
        }
        return false;
    }

    else if (triggerName == "unloading0" or triggerName == "unloading1" or triggerName
== "unloading2" or triggerName == "unloading3") {
            if (!SM_comms_OFF and SR_InTrack[3]) return false;    // Shedmaster insert
            carsU = myTrain.GetVehicles();
            if (initialU) {
                if (SM_comms_OFF) goto INI3;
                else goto INI4;
        }
            if (SM_comms_OFF and (LastTrainUId != myTrain.GetId() or carsU.size() !=
Last_SizeU or Last_Lead_VehicleU != carsU[0])) {
INI3:           LastTrainUId = myTrain.GetId();
```

```
                Last_SizeU = carsU.size();
                Last_Lead_VehicleU = carsU[0];
                initialU = false;
                if(triggerName == "milk_entry0") targetTriggerU = "milk_entry3";
                else targetTriggerU = "milk_entry0";
                comd = itc.GetTrainCommand(myTrain);
                AIUnload = (comd == "unload" or comd == "UNLOAD");
                  return false;
              }
              else if (!SM_comms_OFF and SR_InTrack[1] and !SR_InFlag and LastTrainUId !=
myTrain.GetId()) {
INI4:          LastTrainUId = myTrain.GetId();
                initialU = false;
                if(triggerName == "unloading0") targetTriggerU = "unloading3";
                else targetTriggerU = "unloading0";
                AIUnload = true; // SM only works with trains under AI control
                for (i=0;i<5;i++) {carsToUnload[i] = null;}
                return false;
 }
  canUnload = CanLoadUnload(vehicle, <InQueue>, false);
  if (AIUnload) {
                  if (triggerName == targetTriggerU) {
                      if (!SM_comms_OFF) {
                          // Determine that currrent command is SMC LOAD
                          DriverCharacter myDriver = myTrain.GetActiveDriver();
                          DriverCommands mySchedule =
myDriver.GetDriverCommands();
                          DriverScheduleCommand[] myCommands =
mySchedule.GetDriverScheduleCommands();
                          string myCurrentCommand = myCommands[0].GetTooltip();
                          if (!WordFound(myCurrentCommand, "Unload at Industry",
false)) return false                    if (canUnload and (SR_InTrack[1] or
SR_InTrack[2])) {
                            SR_InFlag = true;
                              myDriver.SetRunningCommands(false); // disable
schedule
                              mySchedule.ProceedToNextCommand();
                              myTrain.StopTrainGently();
                              carsToUnload[0] = vehicle;
                              UnloadAtShedmasterIndustry(myTrain);
                              return false;
                      }
```

```
                        }
                        else {    // SM_comms_OFF is 'true'
                            if (canUnload) return true;
                            IsEndU = IsEndCar(vehicle, "in_track", carsU, milkInQueue);
                            if (IsEndU) {
                             PostMessage(myTrain, "HandleTrain", "Release", 1.0);
                    AIUnload = false;
                    if (!SM_comms_OFF) SR_InTrack[1] = false;
                                }
                     }
                }
        }
        //unloading trigger activated but AI UNLOAD command is not targetting this trigger
point
        if (myTrain.IsStopped()) {
                if (canUnload) {
                        if (SM_comms_OFF) return true;
                        else {    // SM is active. Accumulate list of vehicles that can be
unloaded
                                bool stopLoopU;
                                for (i=0;i<5;i++) {
                                        if (carsToUnload[i] == null) {
                                                for (j=0;j<carsU.size();j++) {
                                                        if (carsU[j] == vehicle) {
                                                                if (i>0) {    // prevent multiple entries
of the same vehicle
                                                                        stopLoopU = false;
                                                                        for (k=0;k<i;k++) {
                                                                                if
(carsToUnload[k]==vehicle) {
                                                                                        stopLoopU = true;
                                                                                        break;
                                                                                }
                                                                        }
                                                                }
                                                                if (!stopLoopU) {
                                                                        carsToUnload[i] = vehicle;
                                                                        triggerU[i] = triggerName;
                                                                }
                                                                break;
                                                        }
                                                }
```

```
                                        break;
                                    }
                                }
                            }
                        }
                    }
                }
                return false;
            }
```

8. ADD UTILITY FUNCTION 'WORDFOUND'

```
//  Is a specified word contained within a string?
    bool WordFound (string myString, string myWord, bool firstWord) {
        if (myString.size()<myWord.size()) return false;
        int i,j, mySize;
        bool misMatch;
        Str.ToLower(myString);
        Str.ToLower(myWord);
        if (firstWord) mySize = 1;
        else mySize = myString.size()-myWord.size()+1;
        for (i=0;i<mySize;i++) {
            if (myString[i] == myWord[0]) {
                if (myWord.size() == 1) return true;
                else {
                    misMatch = false;
                    for (j=1;j<myWord.size();j++) {
                        if (myString[i+j] != myWord[j]) {
                            misMatch = true;
                            break;
                        }
                    }
                    if (!misMatch) return true;
                }
            }
        }
        return false;
    }
```

9. ADD METHODS FOR SHUNTED LOADING AND UNLOADING

Ordinary SM-compatible industries can use the already-present 'PerformStoppedLoad' method, but shunted SM-compatible industry conversions require new methods. A pair of

new methods is required for each industry bay. For simplicity, methods for an industry with one loading bay and one unloading bay is given.

For the loading bay:

```
thread void Load<i>AtShedmasterIndustry(Train myTrain)
{
        int i, j, numCars;
        while (!myTrain.IsStopped()) Sleep(1.0);
        Sleep(3.0); // wait until all non-target triggers have assessed wagons stopped over them
        // Uncouple loco from train
        if (carsL[0].GetVehicleTypeFlags()==1 or carsL[0].GetVehicleTypeFlags()==2 or carsL[0].GetVehicleTypeFlags()==5) {
                for (i=1;i<carsL.size();i++) {
                        if (carsL[i].GetVehicleTypeFlags()==0) {
                                carsL[i-1].Uncouple(carsL[i]);
                                break;
                        }
                }
        }
        else if (carsL[carsL.size()-1].GetVehicleTypeFlags()==1 or carsL[carsL.size()-1].GetVehicleTypeFlags()==2 or carsL[carsL.size()-1].GetVehicleTypeFlags()==5) {
                for (i=carsL.size()-2;i>=0;i--) {
                        if (carsL[i].GetVehicleTypeFlags()==0) {
                                carsL[i+1].Uncouple(carsL[i]);
                         break;
                        }
                }
        }
        else goto SKIP1; // no loco is attached to the train

        myTrain.GetActiveDriver().SetRunningCommands(true); // Restore schedule of loco
        // Load each vehicle standing at industry bay
SKIP1: for (i=0;i<5;i++) {
                if (carsToLoad[i]==null) break;
                else numCars++;
        }
        numCarsLbay = numCars;
        numCarsLoaded = 0;
        for (i=0;i<numCars;i++) {
```

```
                PerformSMload(carsToLoad[i],triggerL[i]);
                Sleep(1.0);
            }
      AILoad = false;
    }

    void PerformSMload(Vehicle vehicle,string triggerName) {
            int productAvailable, direction, queueMag;
            SR_OutTrack[0] = true; // Flag OutTrack as occupied
            SR_OutTrack[1] = false;
            SR_OutTrack[2] = false;
            productAvailable = <product name>OutQueue.GetQueueCount();
            LoadingReport report = CreateLoadingReport(<product name>OutQueue,
productAvailable);
            direction = vehicle.GetRelationToTrack(me, "out_track");
            if (direction == vehicle.DIRECTION_FORWARD) report.sideFlags =
report.RIGHT_SIDE;
            else if (direction == vehicle.DIRECTION_BACKWARD) report.sideFlags =
report.LEFT_SIDE;
            consistL = vehicle.GetMyTrain();
            consistL_DN = consistL.GetTrainDisplayName();
            Vehicle[] carsSL = consistL.GetVehicles();    // consist ID has changed due to
uncoupling of loco
            vehicle.LoadProduct(report);
            numCarsLoaded++;
            queueMag = <product name>OutQueue.GetQueueCount();
            if (numCarsLoaded == numCarsLbay or queueMag == 0) {
                Last_Loaded_Vehicle = vehicle;
                LLV_Name = Last_Loaded_Vehicle.GetLocalisedName();
                if (!SR_OutTrack[3] and !SR_L_Bar) {
                    SendServiceRequest(3);  // Shedmaster:  "Loading_Complete-<product
name>"
                    SR_L_Bar = true;
                    PostMessage(me,"SR_L_Bar","",10.0);
                }
                else if (!SR_OutTrack[2] and !SR_L_Bar){
                    SendServiceRequest(2);   //   Shedmaster:   "Loading_Shunt-<product
name>".
                    SR_L_Bar = true;
                    PostMessage(me,"SR_L_Bar","",10.0);
                }
            }
```

```
}
```

For the unloading bay:

```
thread void UnloadAtShedmasterIndustry(Train myTrain)
{
        int i, j, numCars;
        while (!myTrain.IsStopped()) Sleep(1.0);
        Sleep(3.0); // wait until all non-target triggers have assessed wagons stopped over
them
        // Uncouple loco from train
        if (carsU[0].GetVehicleTypeFlags()==1 or carsU[0].GetVehicleTypeFlags()==2 or
carsU[0].GetVehicleTypeFlags()==5) {
                for (i=1;i<carsU.size();i++) {
                        if (carsU[i].GetVehicleTypeFlags()==0) {
                                carsU[i-1].Uncouple(carsU[i]);
                                break;
                        }
                }
        }
        else if (carsU[carsU.size()-1].GetVehicleTypeFlags()==1 or
carsU[carsU.size()-1].GetVehicleTypeFlags()==2 or
carsU[carsU.size()-1].GetVehicleTypeFlags()==5) {
                for (i=carsU.size()-2;i>=0;i--) {
                        if (carsU[i].GetVehicleTypeFlags()==0) {
                                carsU[i+1].Uncouple(carsU[i]);
                                break;
                        }
                }
        }
        else goto SKIP2; // no loco is attached to the train

        myTrain.GetActiveDriver().SetRunningCommands(true); // Restore schedule of loco
        myTrain.SetAdvisoryLimit(0.0);
        // Load each vehicle standing at industry bay
SKIP2: for (i=0;i<5;i++) {
                if (carsToUnload[i]==null) break;
                else numCars++;
        }
        numCarsUbay = numCars;
        numCarsUnloaded = 0;
        for (i=0;i<numCars;i++) {
```

```
                PerformSMunload(carsToUnload[i],triggerU[i]);
                Sleep(1.0);
            }
        AIUnload = false;
    }

    void PerformSMunload(Vehicle vehicle,string triggerName) {
            int spaceAvailable, direction, queueMag;
            SR_InTrack[0] = true; // Flag InTrack as occupied
            SR_InTrack[1] = false;
            SR_InTrack[2] = false;
            spaceAvailable = <product name>InQueue.GetQueueSpace();
            LoadingReport report = CreateUnloadingReport(<product name>InQueue,
spaceAvailable);
            direction = vehicle.GetRelationToTrack(me, "in_track");
            if (direction == vehicle.DIRECTION_FORWARD) report.sideFlags = report.LEFT_SIDE;
            else if (direction == vehicle.DIRECTION_BACKWARD) report.sideFlags =
report.RIGHT_SIDE;
            consistU = vehicle.GetMyTrain();
            consistU_DN = consistU.GetTrainDisplayName();
            Vehicle[] carsSU = consistU.GetVehicles();    // consist ID has changed due to
uncoupling of loco
            vehicle.UnloadProduct(report);
            numCarsUnloaded++;
            if (!IsEndU) IsEndU = IsEndCar(vehicle, "in_track", carsSU, <product
name>InQueue);
            queueMag = <product name>InQueue.GetQueueSpace();
            if (numCarsUnloaded == numCarsUbay or queueMag == 0) {
                Last_Unloaded_Vehicle = vehicle;
                LUV_Name = Last_Unloaded_Vehicle.GetLocalisedName();
                if (!SR_InTrack[3] and !SR_U_Bar) {
                    SendServiceRequest(6); // Shedmaster: "Unloading_Complete-<product
name>"
                    SR_U_Bar = true;
                    PostMessage(me,"SR_U_Bar","",10.0);
                }
                else if (!SR_InTrack[2] and !SR_U_Bar) {
                    SendServiceRequest(5);  //  Shedmaster:  "Unloading_Shunt-<product
name>".
                    SR_U_Bar = true;
                    PostMessage(me,"SR_U_Bar","",10.0);
                }
```

```
        }
    }
```

10. ADD FUNCTIONS TO SET LOADING AND UNLOADING DELAYS if not already existing

Note that if different delays are required for different industry bays of the same type, then the report compound variable will have to be unpacked and different delays allocated according to the value of 'report.srcQueue'.

```
    float BeginLoad(LoadingReport report) {
      return PRELOAD_DURATION;    // Delay before loading commences
    }
    float BeginUnload(LoadingReport report) {
      return PRELOAD_DURATION; // Delay before unloading commences
    }

    float EndLoad(LoadingReport report) {
      return END_DURATION; // Delay after loading completes
    }
    float EndUnload(LoadingReport report) {
      return END_DURATION; // Delay after unloading completes
    }
```

11. ADD OR MODIFY SESSION SOUP MAINTENANCE METHODS

For simplicity, methods for an industry with one loading bay and one unloading bay is given.

```
  public void SetProperties(Soup soup) {
        inherited(soup);
        string vehicleName;
        SM_comms_OFF = soup.GetNamedTagAsBool("SM_comms_OFF",
SM_comms_OFF);
        int i;
        for (i=0;i<=3;i++) {    // restore track status flags
          SR_InTrack[i] = soup.GetNamedTagAsBool("intrack_"+i);
          SR_OutTrack[i] = soup.GetNamedTagAsBool("outtrack_"+i);
        }
        for (i=0;i<5;i++) {
          triggerL[i] = soup.GetNamedTag("trigger_l_"+i);
          triggerU[i] = soup.GetNamedTag("trigger_u_"+i);
          vehicleName = soup.GetNamedTag("cars_to_load_"+i);
          if (vehicleName == "") carsToLoad[i] = null;
```

```
        else carsToLoad[i] = cast<Vehicle> Router.GetGameObject(vehicleName);
        vehicleName = soup.GetNamedTag("cars_to_unload_"+i);
        if (vehicleName == "") carsToUnload[i] = null;
        else carsToUnload[i] = cast<Vehicle> Router.GetGameObject(vehicleName);
      }
      initialL = soup.GetNamedTagAsBool("initialL", initialL);
      initialU = soup.GetNamedTagAsBool("initialU", initialU);
      AILoad = soup.GetNamedTagAsBool("AILoad", AILoad);
      AIUnload = soup.GetNamedTagAsBool("AIUnload", AIUnload);
      SR_InFlag = soup.GetNamedTagAsBool("SR_InFlag", SR_InFlag);
      SR_OutFlag = soup.GetNamedTagAsBool("SR_OutFlag", SR_OutFlag);
      targetTriggerL = soup.GetNamedTag("target_trigger_load");
      targetTriggerU = soup.GetNamedTag("target_trigger_unload");
      LastTrainLId = soup.GetNamedTagAsInt("last_train_l_id", LastTrainLId);
      LastTrainUId = soup.GetNamedTagAsInt("last_train_u_id", LastTrainUId);
      consistL_DN = soup.GetNamedTag("consistL");
      GameObject MyGO = Router.GetGameObject(consistL_DN);
      consistL = cast<Train> MyGO;
      consistU_DN = soup.GetNamedTag("consistU");
      MyGO = Router.GetGameObject(consistU_DN);
      consistL = cast<Train> MyGO;
      LLV_Name = soup.GetNamedTag("LLV");
      MyGO = Router.GetGameObject(LLV_Name);
      Last_Loaded_Vehicle = cast<Vehicle> MyGO;
      LUV_Name = soup.GetNamedTag("LUV");
      MyGO = Router.GetGameObject(LUV_Name);
      Last_Unloaded_Vehicle = cast<Vehicle> MyGO;
}

public Soup GetProperties(void) {
      Soup soup = inherited();
      soup.SetNamedTag("SM_comms_OFF", SM_comms_OFF);
      int i;
      for (i=0;i<=3;i++) {    // save track status flags
        soup.SetNamedTag("intrack_"+i, SR_InTrack[i]);
        soup.SetNamedTag("outtrack_"+i, SR_OutTrack[i]);
      }
      for (i=0;i<5;i++) {
            if (carsToLoad[i] == null) soup.SetNamedTag("cars_to_load_"+i, "");
            else soup.SetNamedTag("cars_to_load_"+i, carsToLoad[i].GetName());
            if (carsToUnload[i] == null) soup.SetNamedTag("cars_to_unload_"+i, "");
            else soup.SetNamedTag("cars_to_unload_"+i, carsToUnload[i].GetName());
```

```
                soup.SetNamedTag("trigger_l_"+i, triggerL[i]);
                soup.SetNamedTag("trigger_u_"+i, triggerU[i]);
            }
            soup.SetNamedTag("SR_InFlag", SR_InFlag);
            soup.SetNamedTag("SR_OutFlag", SR_OutFlag);
            soup.SetNamedTag("target_trigger_load", targetTriggerL);
            soup.SetNamedTag("last_train_u_id", LastTrainUId);
            soup.SetNamedTag("last_train_l_id", LastTrainLId);
            soup.SetNamedTag("target_trigger_unload", targetTriggerU);
            soup.SetNamedTag("consistL", consistL_DN);
            soup.SetNamedTag("consistU", consistU_DN);
            soup.SetNamedTag("LLV", LLV_Name);
            soup.SetNamedTag("LUV", LUV_Name);
            return soup;
        }

    string GetPropertyType(string p_propertyID) {
        string s;
        if (p_propertyID == "SM_comms_OFF_state")                {s = "link";}
        if ( s!= "link") {s = inherited (p_propertyID);}
        return s;
    }

    void LinkPropertyValue(string p_propertyID) {
        if ( p_propertyID ==    "SM_comms_OFF_state") {
            if (!SM_comms_OFF) {
                SM_comms_OFF = true;
            }
            else {
                SM_comms_OFF = false;
            }
        }
        else {
            inherited (p_propertyID);
        }
    } // end LinkPropertyValue

    public string GetDescriptionHTML(void) {
        string html        = inherited();
        ST = me.GetAsset().GetStringTable();
        string        s1        =        "<br>"        +        ST.GetString("prompt_sm")        +        "<a
href=live://property/SM_comms_OFF_state>";
```

```
    string s2 = "";
    if (SM_comms_OFF) {
        s2 = ST.GetString("Load_0") + "</a><br><br>";
    }
    else {
        s2 = ST.GetString("Load_1") + "</a><br><br>";
    }
    return html + s1 + s2;
}
```