How to use the Auto-Transmation-Config

# Prerequisites

## Software Versions

This example works with inmation Version 1.18.7, and VisualKPI 5.31.4.221. It may work with older and/or newer versions, but this was not tested.

## Environment

It is strongly suggested to run this example only on a blank system to demonstrate the simplicity of adding content to it in inmation and VisualKPI.

The scripts make some assumptions about content explained later, that might prevent them from running properly on systems that already have data on them.

The CSV file imported must not contain any non ASCII character! Such content can result in the example not working properly.

# Setup

## Installation

Run a plain vanilla inmation (all settings default) and VisualKPI installation, with all the configuration required (web service, interface, RC service) to connect them to each other.

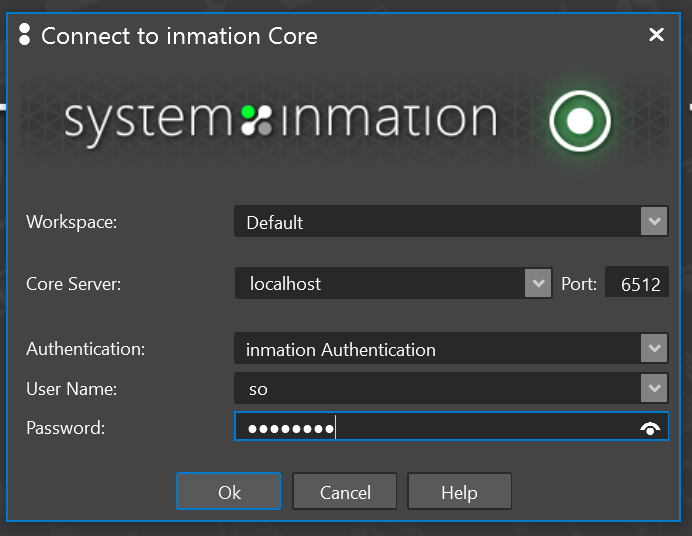
### Save the state (I)

Make a backup copy the folder **C:\inmation.root\img.**

To revert back to this state just replace the img folder with the backup copy you made.

## Configuration

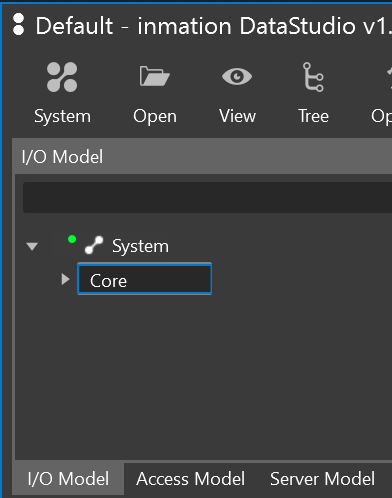
Open DataStudio with admin privileges



Password default is inmation.

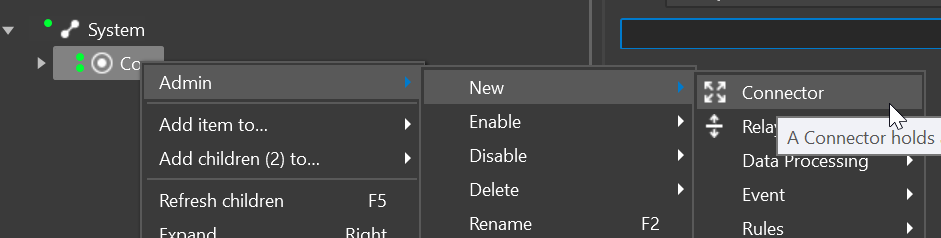
### Rename the Core

In the IO Model tree on the left, rename the Core object to “Core”. Select it and press F2. Enter “Core” in the edit box and hit Enter.

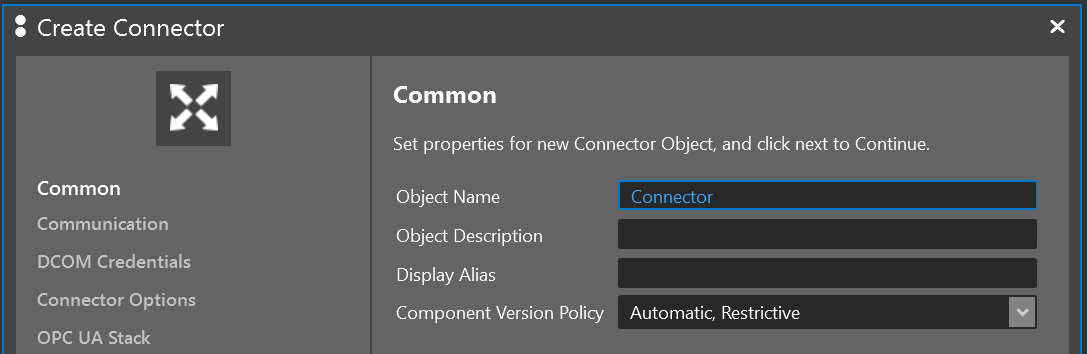


### Add the local connector to the system.

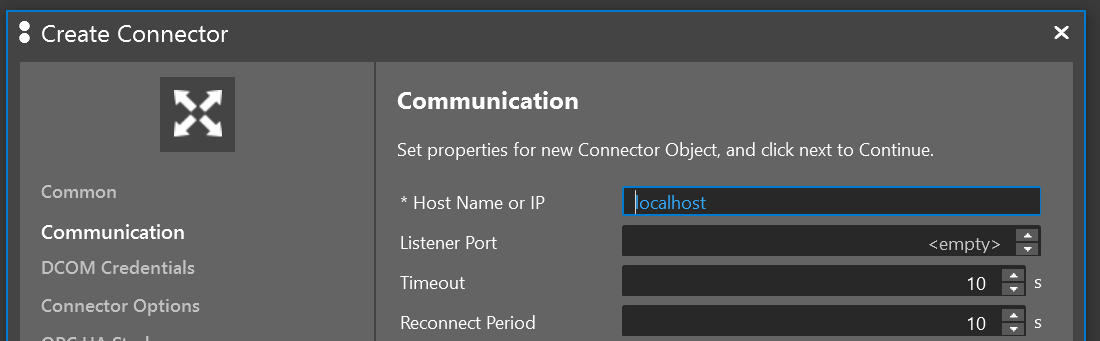
Right click the Core object, and select **Admin🡪New🡪Connector**.



In the Create Connector wizard on the Common page, enter **Connector** in the **ObjectName** field.

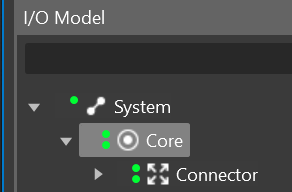


Click **Next**. On the Communication page enter **localhost** in the **Host Name or IP** field



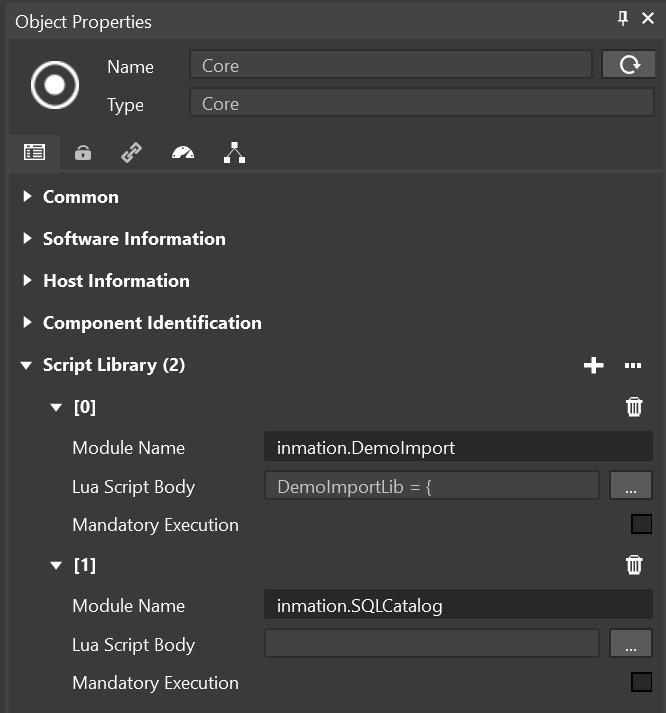
Leave all other options unchanged, and click **Create**

The Connector now shows up in the IO Model.



### Add two required libraries to the Core object.

Select the Core object, and add two script libraries like show below:



For the body of the inmation.DemoImport Library use the following content:

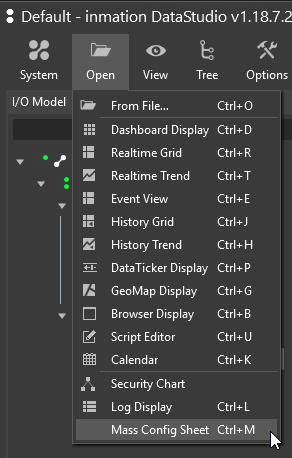
|  |
| --- |
| DemoImportLib = {  -- #####################################################  -- constants describong the import columns  -- #####################################################  COLUMN\_NR\_NAME = 1,  COLUMN\_NR\_DESCRIPTION = 2,  COLUMN\_NR\_ENGINEERINGUNIT = 3,  COLUMN\_NR\_RANGE\_MIN = 4,  COLUMN\_NR\_RANGE\_MAX = 5,  COLUMN\_NR\_GENERATION\_PERIOD = 6,  COLUMN\_NR\_KPI\_LLL = 7,  COLUMN\_NR\_KPI\_LL = 8,  COLUMN\_NR\_KPI\_L = 9,  COLUMN\_NR\_KPI\_T = 10,  COLUMN\_NR\_KPI\_H = 11,  COLUMN\_NR\_KPI\_HH = 12,  COLUMN\_NR\_KPI\_HHH = 13,  COLUMN\_NR\_KPI\_PATH = 14,  COLUMN\_NR\_KPI\_LAT = 15,  COLUMN\_NR\_KPI\_LONG = 16,  --###########################################################################  -- function to split a txt line into a Lua table based on specified separator  --###########################################################################  lineColumnsToTable = function(line,sep)  local res = {}  local pos = 1  sep = sep or ';'  while true do  local c = string.sub(line,pos,pos)  if (c == "") then break end  if (c == '"') then  -- quoted value (ignore separator within)  local txt = ""  repeat  local startp,endp = string.find(line,'^%b""',pos)  txt = txt..string.sub(line,startp+1,endp-1)  pos = endp + 1  c = string.sub(line,pos,pos)  if (c == '"') then txt = txt..'"' end  -- check first char AFTER quoted string, if it is another  -- quoted string without separator, then append it  -- this is the way to "escape" the quote char in a quote. example:  -- value1,"blub""blip""boing",value3 will result in blub"blip"boing for the middle  until (c ~= '"')  table.insert(res,txt)  assert(c == sep or c == "")  pos = pos + 1  else  -- no quotes used, just look for the first separator  local startp,endp = string.find(line,sep,pos)  if (startp) then  table.insert(res,string.sub(line,pos,startp-1))  pos = endp + 1  else  -- no separator found -> use rest of string and terminate  table.insert(res,string.sub(line,pos))  break  end  end  end  return res  end,  --###########################################################################  -- Function to read a txt file into a Lua table of lines  --###########################################################################  readFileLines = function(file)  local ioLib = require'io'  local lines = {}  local f = ioLib.open(file, "rb")  if f ~= nil then  lines = {}  for line in ioLib.lines(file) do  lines[#lines + 1] = line  end  end  f:close()  return lines  end,  -- #####################################################  -- get the next random value between min and max  -- starting from curr and using the offset off  -- #####################################################  GetNextRandomValue = function (min, max, curr, off)  --some safety checks of the inputs  if type(min) ~= "number" and type(max) ~= number then  return nil  else  if type(curr) ~= "number" or curr == nil or curr == "" then  curr = min + ((max - min) \* 0.5)  end  end  if type(off) ~= "number" or off == nil then  --use 1% of range as offset, if not set  off = (max - min) \* 0.01  end  -- Throw a coin to go up or down.  -- Add/Sub random offset to last sample to get new sample  if (math.random() < 0.5) then  curr = curr + math.random() \* off  else  curr = curr - math.random() \* off  if (curr < 0) then  curr = 0  curr = curr + math.random() \* off  end  end  -- If current value goes outside valid min/max force it to be 50%  if (curr < min or curr > max) then  curr = min + ((max - min) \* 0.5)  end  --return the new value  return curr  end,  --###########################################################################  -- Function to create artificial history objects  --###########################################################################  createHistory = function(objectpath, sampleperiod, days)  local now = inmation.now()  local old = now - days \* 24 \* 60 \* 60 \* 1000  local pid = inmation.getpropertyid(objectpath)  local minval = inmation.getvalue(objectpath .. ".Limits.OpcRangeLow")  local maxval = inmation.getvalue(objectpath .. ".Limits.OpcRangeHigh")  local nextval = minval + ((maxval - minval) \* 0.5)  for t = old, now, sampleperiod do  inmation.sethistory(tonumber(pid), nextval, 0, t)  nextval = DemoImportLib.GetNextRandomValue(minval, maxval, nextval);  end  end,  }  return DemoImportLib |

For the body of the inmation.SQLCatalog Library use the following content:

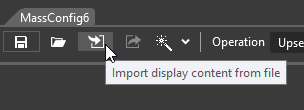
|  |
| --- |
| sqlcatalogLib = {  db = inmation.getsystemdb(),  customstr = {'@', 'CustomPropertyName.'},  ------------------ PROPERTIES --------------------  --------------------------------------------------  --- Filters objects depending on their properties' values  --- The properties, operators and values can be provided as tables, in which case they are mapped one to one.  --- When the properties table is empty, an empty table is returned.  --- When there are less operators than properties, the missing positions are filled with the default operator '='  --- When there are less values than properties, the missing positions are filled with the last provided value  -- @param properties It can be a string representing the property's name or a table with such strings  -- A custom property can be specified with its defined name or, in case this collides with an already existing  -- property, it can be prepended with "CustomPropertyName." or "@"  -- @param values It can be nil or a single value or a table with values  -- @param operators It can be nil or a string representing an SQL operator (e.g. IS, LIKE etc.) or a table with such strings  -- @return A table with the filtered objects \_OR\_ an error message  --------------------------------------------------  filterByPropValue = function(self, properties, values, operators)  if properties == nil or (type(properties) ~= 'table' and type(properties) ~= 'string') then  return "Invalid argument! The properties must be suplied as a table or a string."  end    if type(properties) ~= 'table' then  properties = {properties}  end    if type(operators) ~= 'table' then  if operators == nil then  operators = {}  else  operators = {operators}  end  end  for i = #operators + 1, #properties do  operators[i] = '='  end    if type(values) ~= 'table' then  values = {values}  end  local lastval = values[#values]  for i = #values + 1, #properties do  values[i] = lastval  end  local objects = {}  local query = "SELECT p1.objid as oid FROM properties p1"    -- build the first part of the query  local first\_part\_simple = " INNER JOIN properties p%s ON p%s.objid = p1.objid"  local first\_part\_custom = " INNER JOIN properties p%s ON p%s.objid = p%s.objid AND p%s.position = p%s.position"    if inmation.model.properties[properties[1]] == nil then  query = query .. string.format(first\_part\_custom, 11, 11, 1, 11, 1)  end    for i=2, #properties do  if inmation.model.properties[properties[i]] ~= nil then -- simple property  query = query .. string.format(first\_part\_simple, i, i)  else -- custom property  query = query .. string.format(first\_part\_simple, i, i)  query = query .. string.format(first\_part\_custom, i..i, i..i, i, i..i, i)  end  end    -- build second part of the query  local where = " WHERE p1.code = %s AND p1.value %s %s"  local second\_part\_simple = " AND p%s.code = %s AND p%s.value %s %s"    local codekey = inmation.model.properties.CustomPropertyName  local codeval = inmation.model.properties.CustomPropertyValue    local code = inmation.model.properties[properties[1]]  if type(values[1]) == 'string' then  values[1] = "'" .. values[1] .. "'"  elseif type(values[1]) == "boolean" then  values[1] = values[1] and 1 or 0  elseif type(values[1]) == 'table' and string.find(operators[1], 'BETWEEN') ~= nil then -- BETWEEN or NOT BETWEEN  values[1] = values[1][1] .. ' AND ' .. values[1][2]  end  if code ~= nil then  query = query .. string.format(where, code, operators[1], values[1])  else  local key = string.gsub(properties[1], '@', '')  key = string.gsub(key, self.customstr[2], '')  query = query .. string.format(where, codekey, "=", "'" .. key .. "'")  query = query .. string.format(second\_part\_simple, 11, codeval, 11, operators[1], values[1])  end    local second\_part\_custom = " AND p%s.code = %s AND p%s.value = %s AND p%s.code = %s AND p%s.value %s %s"    for i=2, #properties do  code = inmation.model.properties[properties[i]]  if type(values[i]) == 'string' then  values[i] = "'" .. values[i] .. "'"  elseif type(values[i]) == "boolean" then  values[i] = values[i] and 1 or 0  elseif type(values[i]) == 'table' and string.find(operators[i], 'BETWEEN') ~= nil then  values[i] = values[i][1] .. ' AND ' .. values[i][2]  end  if code ~= nil then  query = query .. string.format(second\_part\_simple, i, code, i, operators[i], values[i])  else  local key = string.gsub(properties[i], self.customstr[1], '')  key = string.gsub(key, self.customstr[2], '')  query = query .. string.format(second\_part\_custom, i, codekey, i, "'" .. key .. "'", i..i, codeval, i..i, operators[i], values[i])  end  end    local cur, err = self.db:query(query)  if cur == nil then  return err  else  local row = cur:fetch ({}, "a")  while row do  table.insert(objects, inmation.getobject(row.oid))  row = cur:fetch (row, "a") -- get the next row  end  end    return objects  end,  ---------------- LIVE PROPERTIES -----------------  --------------------------------------------------  --- Filters objects depending on their dynamic or volatile properties' values  -- @param value A value representing the argument of the SQl operator  -- @param operator It can be nil or a string representing an SQL operator (e.g. IS, LIKE etc.)  -- @return A table with the filtered objects or an error message  --------------------------------------------------  filterByLivePropValue = function(self, property, value, operator)  if property == nil or type(property) ~= 'string' then  return "Invalid argument! The property must be suplied as a string."  end    if operator == nil then  operator = '='  end  local objects = {}  local query = "SELECT objid FROM live\_properties WHERE code = %s AND value %s %s"    local code = inmation.model.properties[property]  if type(value) == 'string' then  value = "'" .. value .. "'"  end    query = string.format(query, code, operator, value)    local cur, err = self.db:query(query)  if cur == nil then  return err  else  local row = cur:fetch ({}, "a")  while row do  table.insert(objects, inmation.getobject(row.objid))  row = cur:fetch (row, "a") -- get the next row  end  end    return objects  end  }  return sqlcatalogLib |

### Add CSV Datasource and Core Logic via MassConfig

Open a MassConfig display

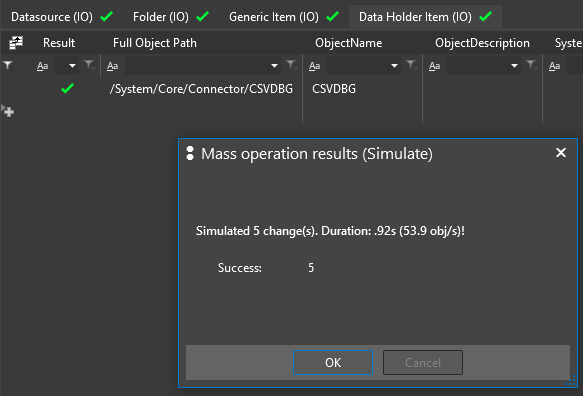


In the opened sheet import the **MassConfig\_csv.xslx** provided with this document.



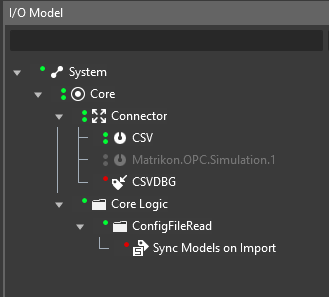
After the import was executed, click the **Simulate** button at bottom of the MassConfig display.

It should be successful like shown below:



Click OK to close the simulation results, then click Apply next to the Simulate button at the bottom of the MassConfig display.

The import should be successful and some objects should get created in the IO Model.



The red state of CSVDBG and Sync Models on Import is ok at this stage.

### Save the state (II)

Make another backup copy the folder **C:\inmation.root\img.**

To revert back to this state just replace the img folder with the backup copy you made.

# Run the demo

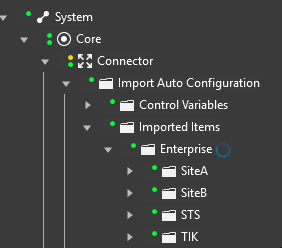
## File Copy

The configured Dropzone datasource **CSV** watched the folder **C:\inmation.root\drop\zone\_log\_import** for incoming files.

Copy the file “CSV Sample Import.txt” provided with this document to the above mentioned folder, and watch what happens in the IO Model in DataStudio.

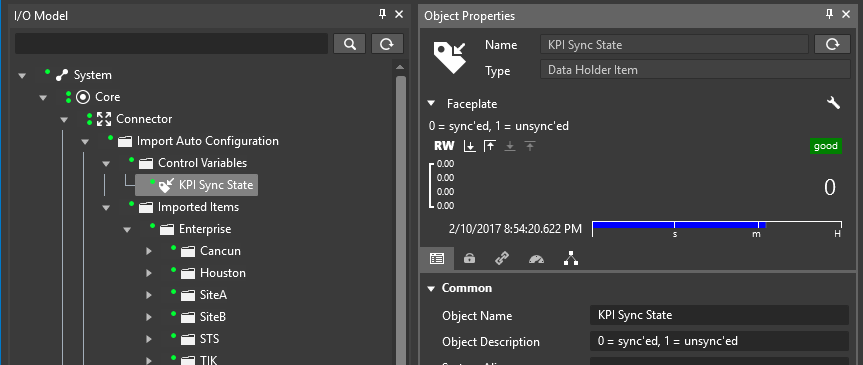
## Import Flow

The import of the file can take 30+ seconds, because of the creation of artificial history for the items in the IO Model. While the import is taking place you might catch a blue spinner next to a folder, which means that there are currently items being added to this node.

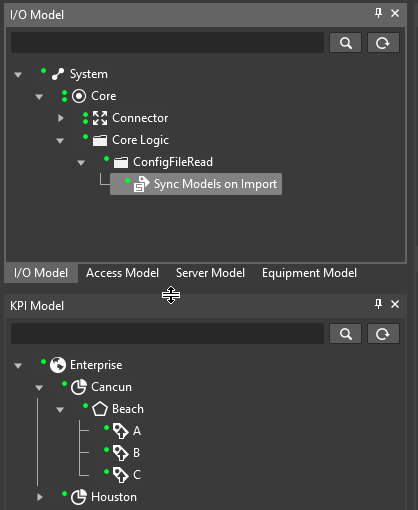


Also you might notice the Connector going to a yellow/green state for about a minute, which signals that he needs to buffer history data because the current intake is higher than what can be send to the Core. This is expected due to the artificial creation of a lot of history.

The last step of the creation of items in the IO Model is setting the KPI Sync State item value to 1. If this is set, the Core logic to create the KPI model items kicks in.

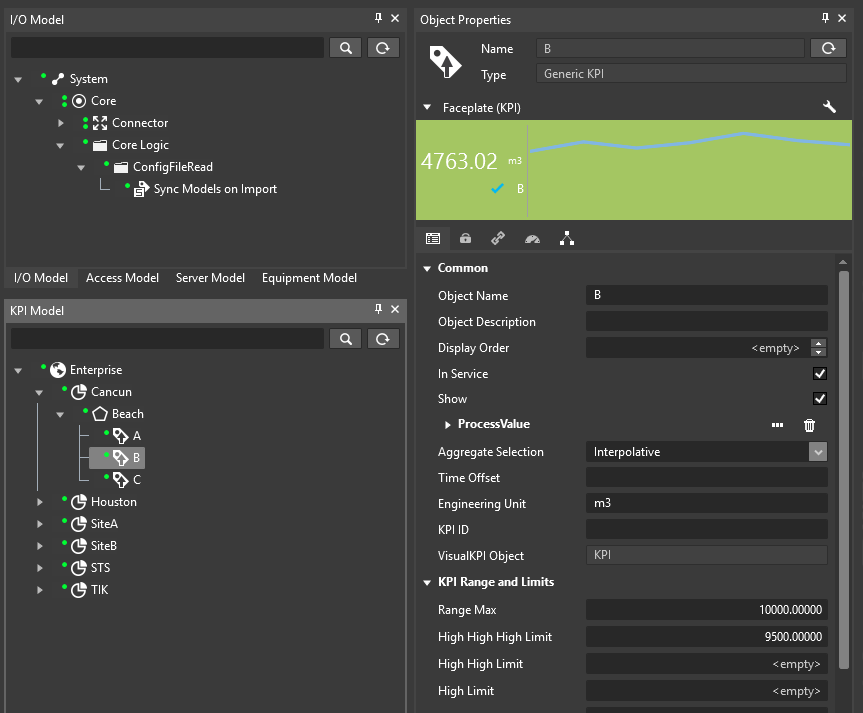


This is located in the Sync Models on Import item. It runs every 10 seconds checking the sync state item mentioned above. If it finds a 1, it starts setting up the KPI Model.



As a last step after creating the KPI Model it set the sync state item back to 0.

Click on one of the KPIs to check the linking to an IO Item.



If that is working fine (Faceplate shows data, limits set), the KPIs should get picked up by VKPI RC automatically:

