This course is intended for Process Simulator (PCS) Professional users who have previously completed PCS Basic or Essentials Training.

Our hope is that this training will teach PCS users several features that increase their skill sets so they can maximize use of the software to benefit their business.

Process Simulator Advanced Features Webinar

Using Arrays to Create Custom Excel Reports from Models

Instructor Info:
Rebecca Santos
Technical Support Engineer
Office: 888.PROMODEL
rdossantos@promodel.com

Poll #1

04/2018 Version 9.3
PCS Refresher Training Webinar
For Software Version: 9.3

556 E Technology Way
Orem, UT 84097
801-223-4600

ProModel®
Better Decisions—Faster

Course Objectives

During this Webinar you will learn how to:

- Set up & use Arrays to capture information about the behavior of a model
- Create some new statements using identifying functions in Free Form logic to populate Arrays as Custom Excel Reports
- Use Subroutines to repeat logic
- Use the Array Export feature
- Answer Attendees' questions (*as time allows*)

PCS Free vs PCS PRO Comparison

Feature	Capability	Free	Pro
2-Dimensional Arrays	Allows the storing of large amounts of data in a single data structure for the model to access and use.		Yes
Import/Export Data via Excel	Allows the population of arrays from an Excel file when the model starts simulating.		Yes
Advanced Subroutine Functionality	Allows passing of parameters to subroutines and returning calculated values, In turn, this enables the leveraging of arrays and parallel process subroutines.		Yes
Advanced Logic Builder	Incorporates advanced statements and functions in the Logic Builder - e.g. referencing entities, activities or resources by index in arrays and subroutines.		Yes
Complex Expressions in Property Fields	Enter subroutines with a return value in place of a numerical value.		Yes
Free Form Logic	An additional window that allows logic to be entered without the Logic Builder, which enables rapid model building.		Yes
Intellisense for Fast Logic Creation	An intelligent list of statements, functions and model elements that pops up when writing logic in free Form Logic.		Yes
Syntax Guide for Quick Logic Help	A tool pit that appears in free Form Logic, which displays the syntax for the statement or function being used.		Yes

Arrays

- An array is a matrix of values
- Each cell works like a variable
- A reference to a cell in an array can be used anywhere a variable can be used
- Refer to a specific array value by using the Array name followed by the specific value's row & column cell address.
- For example, the value 18 located above in row 2 and column 3 has a cell address of [2,3] so it would be referred to as Array1[2,3].

Array1:

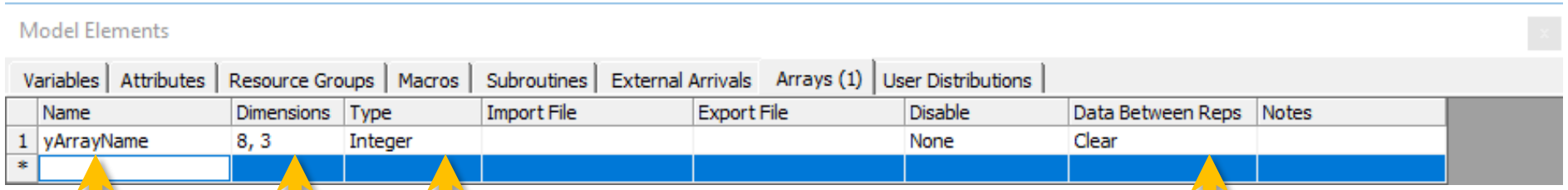
	1	2	3	4
1	10	15	15	20
2	12	15	**18**	25
3	15	15	10	10

Array1 Cell Addresses:

Cell [1,1]	Cell [1,2]	Cell [1,3]	Cell [1,4]
Cell [2,1]	Cell [2,2]	**Cell [2,3]**	Cell [2,4]
Cell [3,1]	Cell [3,2]	Cell [3,3]	Cell [3,4]

Arrays

- Name and define in the Arrays Tab



Name of the Array

Row and **Column** Dimensions

Allowed types: **Integer**, **Real**, or **Expression**

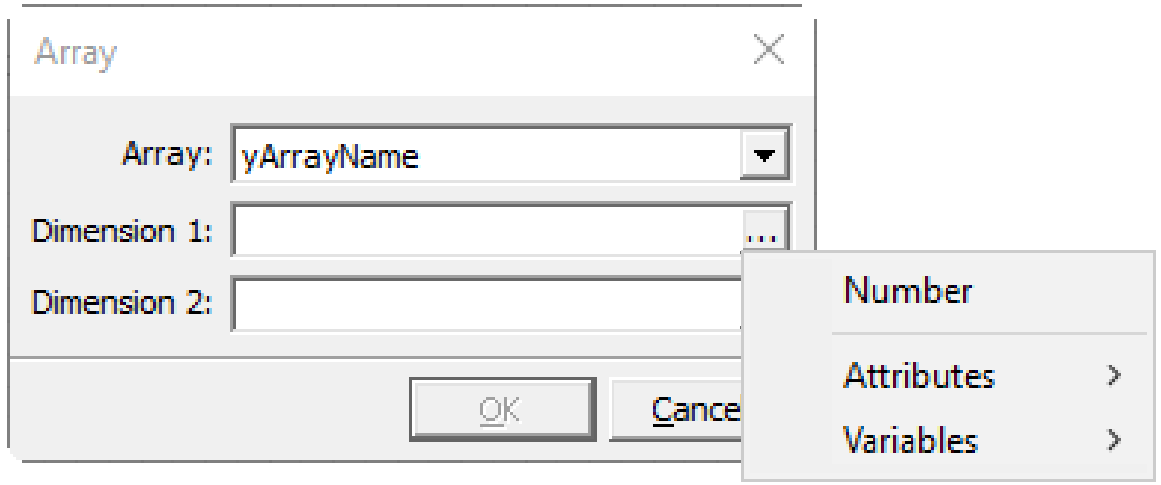
Clear (re-import) or **Keep** array data between replications

Referencing Arrays

- Reference an Array: ArrayName[row, column]
  - Embed attribute or variable references within the array dimensions:
  - Use an Array in Logic:

```
Wait yArray_ProcessTime[3, 45]

Inc yCounter[2, aJobType]
```

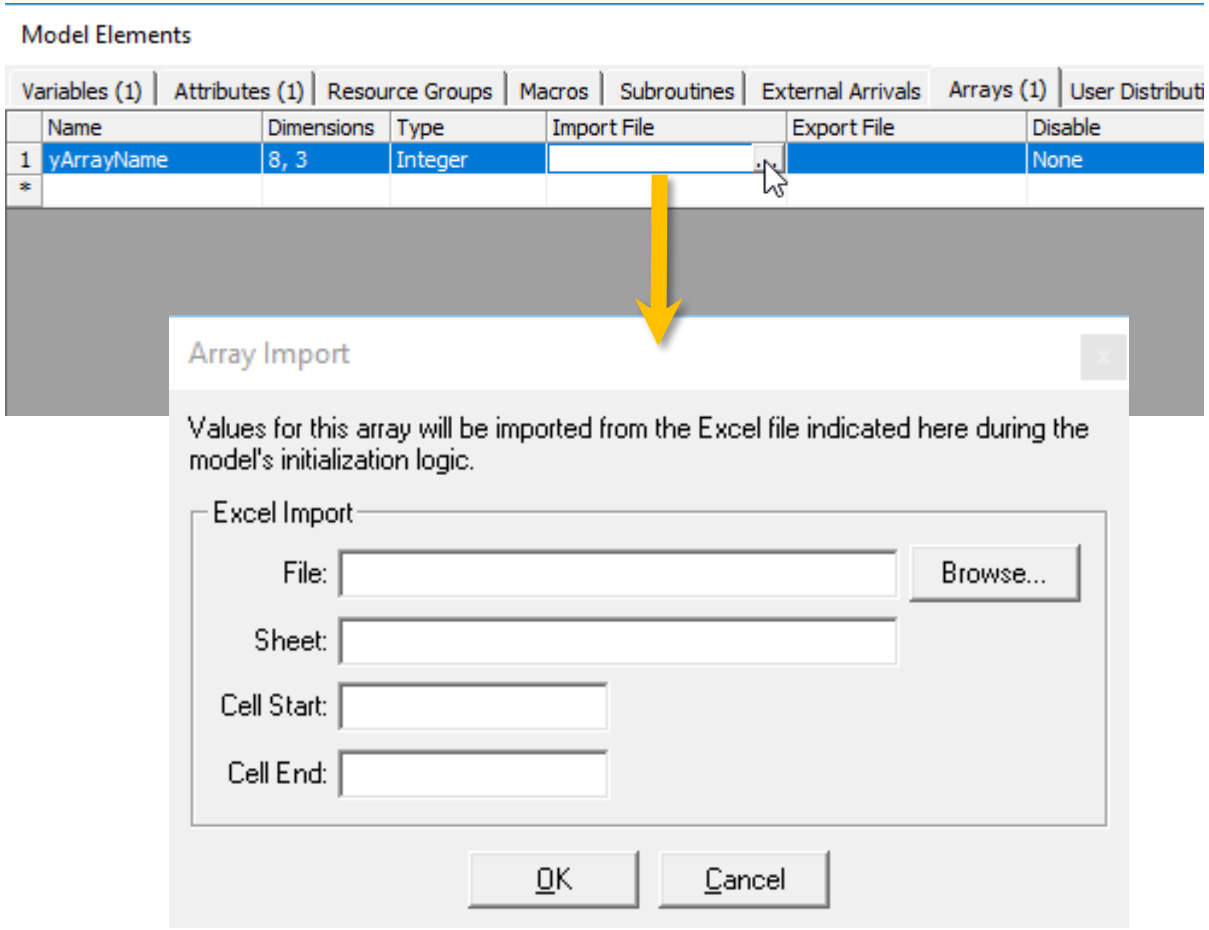


Array Notes

- Statistics are not generated for arrays
- All array cells are initially 0 by default
- An array may be referenced from any logic
- Frequently initialized directly from a spreadsheet

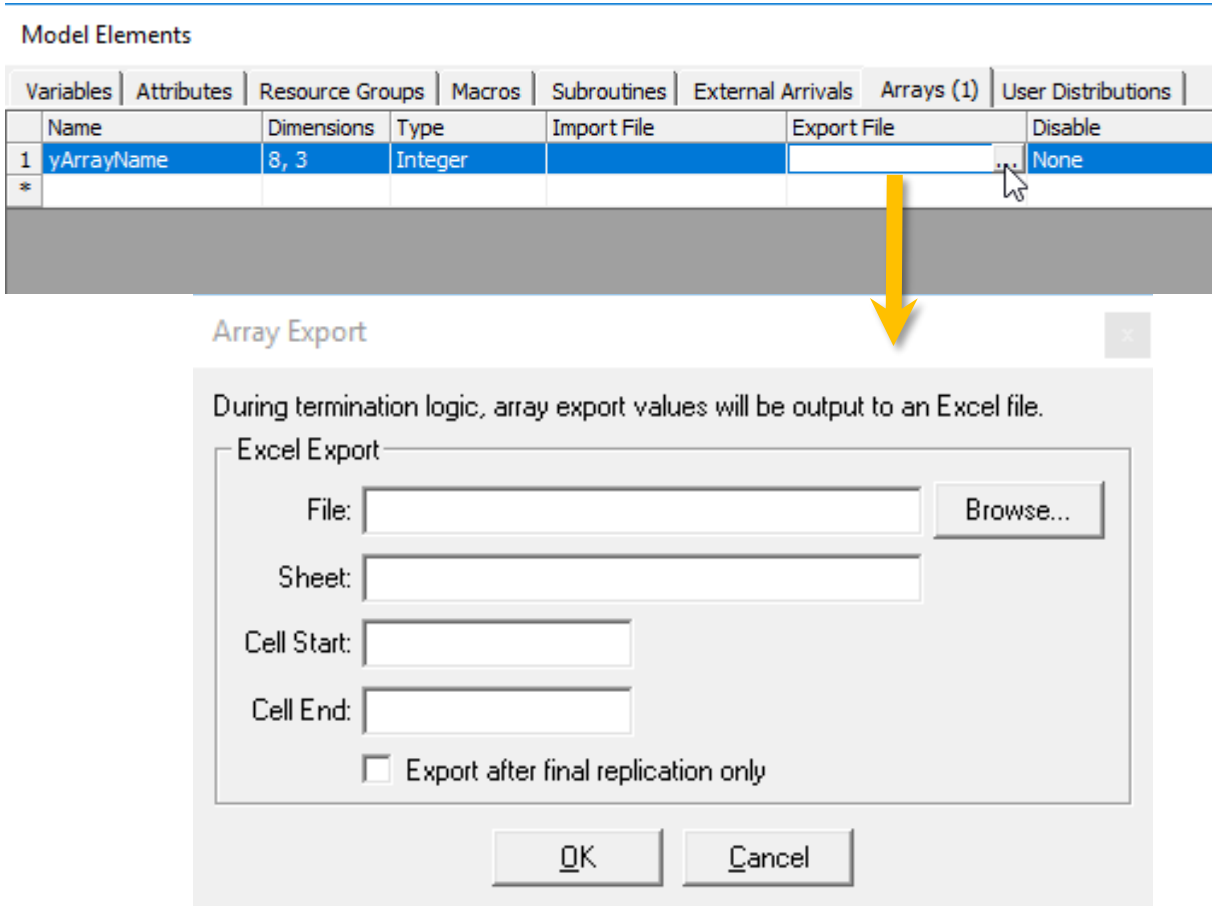
Import Spreadsheet to Array

- Import requires user to specify start and end cells to form a range (end cell is optional)



Export Array to Spreadsheet

- The spreadsheet is populated at model termination
- When exporting multiple replications or scenarios, data from each one is saved to its own worksheet



Poll #2

ProModel®
Better Decisions—Faster

Custom Reports Scenario 1

Scenario:

We have a model with multiple resources and **Get** statements. Oftentimes we call a resource and there is a delay in the time we request the resource until they are available. How can we track and report specifics about these delays?

Goals:

1. Capture the *elapsed* time from when the **Get** statement was issued until the Resource arrived (and processing then continues)
2. In addition to this elapsed time, record the Activity from which the request was issued, and which Resource was actually captured.
3. Capture this data in an Excel spreadsheet (through the use of arrays)

Model Overview

- Let's start by talking through the steps required.

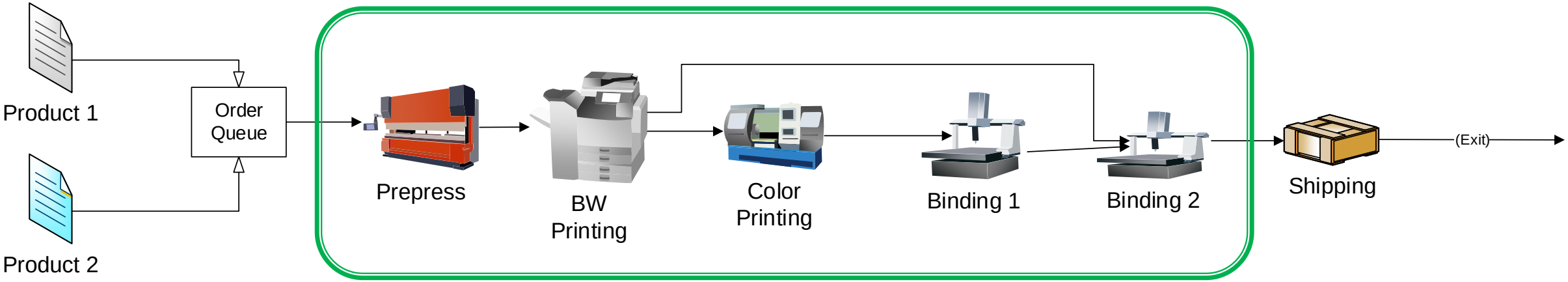
  (Your instructor will step you through an overview of the model)

- The starting model has two resources defined:
  - Copy_Boy (3 units) and Copy_Girl (2 units)

- In the Processing at each station (except Order_Que and Shipping), there is a statement to **Get** a resource before waiting the defined processing time. We release the captured resource in Routing logic.

Get Copy_Boy Or Copy_Girl

Model


Product 1
Product 2
Order Queue
Prepress
BW Printing
Color Printing
Binding 1
Binding 2
Shipping
(Exit)


Copy Boy
Copy Boy.2
Copy Boy.3
Copy Girl
Copy Girl.2

Custom Reports Step 1

- Step 1: Row Counter

  Each entity entering the logic will increment a Variable named vProcessStep. This variable will then be assigned to the Entity Attribute aRequestOrder.

  Inc vProcessStep

  aRequestOrder = vProcessStep

- We do this because multiple entities will be calling this logic possibly simultaneously. This gives each call of this logic its own row number in the array—a unique row identifier within the array.

Custom Reports Step 2

- Step 2: Elapsed Time

  Because we want to capture the time that has elapsed, we will need a REAL attribute, aRequestTime, to record the time that the **Get** statement was issued. This assignment occurs right *before* the **Get** statement is issued. Because there is no processing time required for the Assignment, the **Get** statement is issued at the exact same simulation clock time as the aRequestTime is assigned the current clock time:

  aRequestTime = Clock()

  Get Copy_Boy Or Copy_Girl

Step 3: Array Column Design

Our array will be defined so that each row records the data from each unique process step. Each column will hold specific categories of data:

- Column One: Counter number (Row number)
  This is optional, but basically provides an index number to your array results.

- Column Two: Location Name
  This will record the name of the location (Activity) from which the **Get** statement was issued.

- Column Three: Elapsed Time
  This will be a calculation, based on the difference between the new clock time (after the Resource arrived) and the time the **Get** statement was issued.

- Column Four: Captured Resource Name
  This will record the name of the Resource that responded to the **Get** request.

If the Activity name and the Resource Name vary,
how will we know what they are?

Identifying Functions

Syntax for use within expression arrays:

- **Loc(Location())** → Returns the name of the Activity where the entity is currently processing
- **Res(OwnedResource())** → Returns the name of the most recently captured Resource
- Note that these Functions (**Loc** & **Res**) can only be used in Free Form logic.

Column Design w/Syntax

- Our array will be defined so that each row records the data from each unique process step. Each column will hold specific categories of data:
- Column One: Counter number (Row number)

  This is optional, but basically provides an index number to your array results.

  =aRequestOrder
- Column Two: Location Name

  This will record the name of the location from which the **Get** statement was issued.

  =Loc(Location())
- Column Three: Elapsed Time

  This will be a calculation, based on the difference between the new clock time (after the Resource arrived) and the time the **Get** statement was issued.

  =Clock() – aRequestTime
- Column Four: Captured Resource Name

  This will record the name of the Resource that responded to the **Get** request.

  =Res(OwnedResource())

Step 4: Defining the Array

- Since we know we have four columns of data, our array will be four columns wide. We don't, however, know how many rows we need. We will approximate with 3000 rows.

Array Name: **yResourceWaitStats**
Array Dimensions: **3000, 4**
Array Type: **Expression**
Export File: **CustomReport.xlsx**

Step 5: Pulling it all Together

Inc vProcessStep
aRequestOrder = vProcessStep
aRequestTime = Clock()
Get Copy_Boy OR Copy_Girl
//firstcolumn: counter number
yResourceWaitStats[aRequestOrder,1] = **aRequestOrder**
//second column: location name
yResourceWaitStats[aRequestOrder,2] = **Loc(Location())**
//third column: elapsed time
yResourceWaitStats[aRequestOrder,3] = **Clock() – aRequestTime**
//fourth column: owned resource (most recently captured)
yResourceWaitStats[aRequestOrder,4] = **Res(OwnedResource())**

- ***Wait*** (this takes place in the general time field or multi-entity fields)
- ***Free All*** will be added in the Routings to release the captured Resource

Poll #3

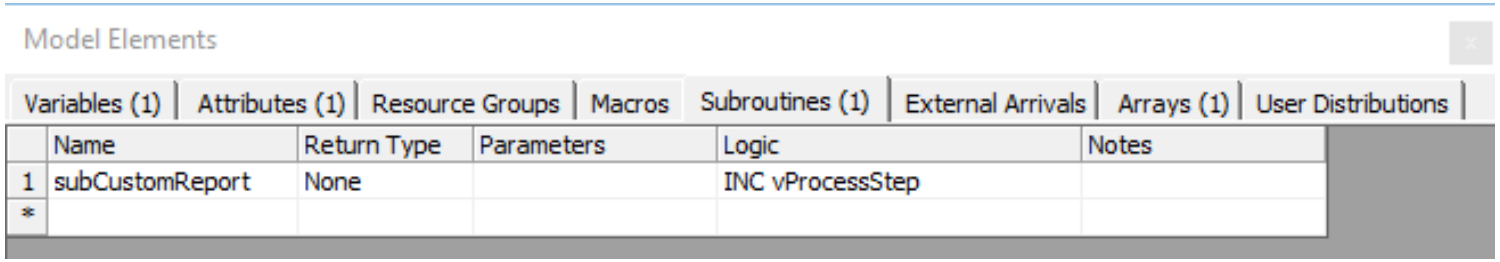
Repeating the Logic

- You could copy and paste the report steps to each Process record where we are getting the Copy_Boy or Copy_Girl.

- OR… you could create the lines of code within a Subroutine and call the Subroutine as needed.

Subroutines

- User defined block of logic
- Useful for calling identical logic from multiple places
- Changes can be made in the subroutine code and the logic is reflected through the entire model.
- Similar to macros, but with added functionality of lines of logic (not just value substitution).



- Called by entering the Subroutine name in calling logic, followed by parentheses.

Types of Subroutines

- ✓ Most basic:
  1. Calling logic starts the subroutine logic by calling sub name.

     sNameOfSub()
  2. Subroutine logic is executed
  3. Simulation returns to executing the next line of code (in the calling logic)

- ❑ More advanced options:
  - Activate Option
  - Pass parameter values to subroutine
  - Return a value back to the calling logic

Exercise: Subroutine with Report Logic

- Define a Subroutine (subCustomReport)
- Use the Array design for the Subroutine Logic:

```
Inc vProcessStep
aRequestOrder = vProcessStep
aRequestTime = Clock()
Get Copy_Boy OR Copy_Girl
//firstcolumn: counter number
yResource_Wait_Stats[aRequestOrder,1] = aRequestOrder
//second column: location name
yResource_Wait_Stats[aRequestOrder,2] = Loc(Location())
//third column: elapsed time
yResource_Wait_Stats[aRequestOrder,3] = Clock() - aRequestTime
//fourth column: owned resource (most recently captured)
yResource_Wait_Stats[aRequestOrder,4] = Res(OwnedResource())
```

Logic: subCustomReport

```
1  Inc vProcessStep
2  aRequestOrder = vProcessStep
3  aRequestTime = Clock()
4  Get Copy_Boy or Copy_Girl
5  //firstcolumn: counter number
6  yResourceWaitStats[aRequestOrder,1] = aRequestOrder
7
8  //second column: location name
9  yResourceWaitStats[aRequestOrder,2] = Loc(Location())
10
11 //third column: elapsed time
12 yResourceWaitStats[aRequestOrder,3] = Clock() - aRequestTime
13
14 //fourth column: owned resource (most recently captured)
15 yResourceWaitStats[aRequestOrder,4] = Res(OwnedResource())
16
```

- Call the Subroutine from Process Operation logic at Prepress, BW_Printing, Color_Printing, etc., replacing the **Get** statement currently there.

Custom Reports Scenario 1 - Recap

- Define the Array:
  - Array Name: **yResourceWaitStats**
  - Array Dimensions: **3000, 4**
  - Array Type: **Expression**
  - Export File: **CustomReport.xlsx**
- Define the Subroutine(subCustomReport)
- Use the Array logic for the Subroutine Logic:

```
Inc vProcess_Step
aRequestOrder = vProcess_Step
aRequestTime = Clock()
Get Copy_Boy OR Copy_Girl
//firstcolumn: counter number
yResourceWaitStats[aRequestOrder,1] = aRequestOrder
//second column: location name
yResourceWaitStats[aRequestOrder,2] = Loc(Location())
//third column: elapsed time
yResourceWaitStats[aRequestOrder,3] = Clock() - aRequestTime
//fourth column: owned resource (most recently captured)
yResourceWaitStats[aRequestOrder,4] = Res(OwnedResource())
```

- Call the Subroutine from Activity Logic

Poll #4

Custom Reports Scenario 1 - Results

- Exercise Results

	A	B	C	D
1	Index	Activity	Response Time	Resource
2	1	Prepress	0	Copy_Boy
3	2	Prepress	0	Copy_Boy
4	3	Prepress	0	Copy_Boy
5	4	Prepress	0	Copy_Girl
6	5	Prepress	0	Copy_Girl
7	6	BW_Printing	0	Copy_Boy
8	7	Prepress	0.602	Copy_Boy
9	8	BW_Printing	0	Copy_Boy
10	9	Prepress	0	Copy_Girl
11	10	BW_Printing	0	Copy_Girl
12	11	Prepress	2.875	Copy_Girl
13	12	BW_Printing	3.285	Copy_Boy
14	13	Prepress	8.163	Copy_Boy
15	14	BW_Printing	7.555	Copy_Girl
16	15	Prepress	9.016	Copy_Boy
17	16	BW_Printing	6.125	Copy_Girl
18	17	Prepress	8.954	Copy_Boy
19	18	BW_Printing	9	Copy_Boy
20	19	Prepress	10.539	Copy_Boy
21	20	Binding_2	6.349	Copy_Boy
22	21	BW_Printing	9	Copy_Girl
23	22	Prepress	10.621	Copy_Girl
24	23	Binding_2	9.446	Copy_Boy
25	24	Binding_2	12.36	Copy_Boy
26	25	BW_Printing	12.164	Copy_Girl
27	26	Prepress	14.55	Copy_Girl
28	27	BW_Printing	17.319	Copy_Girl

Sheet1

Custom Output Reports Scenario 2

- This scenario will be a modification of the last example. We have an additional Resource: **Manager** -- ADD a Manager Resource of your choosing

- And, instead of:
  Get Copy_Boy Or Copy_Girl

- We will have:
  Jointly Get Copy_Boy And (Copy_Girl Or Manager)

Note the syntax. Free form logic uses "Jointly Get." Builder logic uses "Get Jointly"

- Now, when we record which Resource was actually captured, our statement only shows the most recently captured resource.

  =Res(OwnedResource())

- To get the list of all resources currently captured we will need to loop through "the list" of owned resources.

Looping OwnedResource

- OwnedResource() returns the most recently captured Resource
- OwnedResource(1) returns the first Resource captured (longest held)
- OwnedResource(2) returns the second Resource captured (if more than one)
- OwnedResource(3) returns the third Resource (if any)

- To get a list of Resources, we need to create a counting loop to cycle through:

- **Int counter = 1**
  While counter < 4 do //use a value up to max number of resources you expect
  {
  yArrayName[row, counter]=Res(OwnedResource(counter)) //column 1 lists resource 1, etc.
  Inc counter
  }

Owned Resource Loop

- **Scenario 1 method:**

  //fourth column: owned resource (most recently captured)

  yResourceWaitStats[aRequestOrder,4] = **Res(OwnedResource())**

- **New method:**

  //fourth column (and 5th and 6th): owned resource (loop through up to 3)

```
INT counter = 1
While counter < 4 do
{
   yResourceWaitStats[aRequestOrder,3+counter]=Res(OwnedResource(counter))
   Inc counter
}
```

Note: You will need to increase the dimensions of your array for the additional columns.

Custom Reports Scenario 2 - Results

- Exercise Results

	A	B	C	D	E	F	G	H
1	Index	Activity	Response Time	Resource				
2	1	Prepress	0	Copy_Girl	Copy_Boy	[Unknown Resource Name]		
3	2	Prepress	0	Copy_Girl	Copy_Boy	[Unknown Resource Name]		
4	3	Prepress	0	Manager	Copy_Boy	[Unknown Resource Name]		
5	4	Prepress	3.017	Copy_Girl	Copy_Boy	[Unknown Resource Name]		
6	5	Prepress	4.619	Copy_Girl	Copy_Boy	[Unknown Resource Name]		
7	6	BW_Printing	0.837	Manager	Copy_Boy	[Unknown Resource Name]		
8	7	Prepress	3.779	Copy_Girl	Copy_Boy	[Unknown Resource Name]		
9	8	BW_Printing	4.603	Copy_Girl	Copy_Boy	[Unknown Resource Name]		
10	9	Prepress	6.697	Copy_Girl	Copy_Boy	[Unknown Resource Name]		
11	10	BW_Printing	9	Manager	Copy_Boy	[Unknown Resource Name]		
12	11	Prepress	9.497	Copy_Girl	Copy_Boy	[Unknown Resource Name]		
13	12	BW_Printing	11.426	Copy_Girl	Copy_Boy	[Unknown Resource Name]		
14	13	Prepress	12.059	Copy_Girl	Copy_Boy	[Unknown Resource Name]		
15	14	BW_Printing	13.632	Manager	Copy_Boy	[Unknown Resource Name]		
16	15	Prepress	14.224	Copy_Girl	Copy_Boy	[Unknown Resource Name]		
17	16	BW_Printing	16.906	Copy_Girl	Copy_Boy	[Unknown Resource Name]		
18	17	Prepress	17.189	Copy_Girl	Copy_Boy	[Unknown Resource Name]		
19	18	Binding_2	17.409	Copy_Girl	Copy_Boy	[Unknown Resource Name]		
20	19	BW_Printing	18.503	Manager	Copy_Boy	[Unknown Resource Name]		
21	20	Prepress	23.871	Copy_Girl	Copy_Boy	[Unknown Resource Name]		
22	21	Binding_2	23.632	Manager	Copy_Boy	[Unknown Resource Name]		
23	22	BW_Printing	23.754	Copy_Girl	Copy_Boy	[Unknown Resource Name]		
24	23	Prepress	33.754	Copy_Girl	Copy_Boy	[Unknown Resource Name]		
25	24	Binding_2	35.748	Copy_Girl	Copy_Boy	[Unknown Resource Name]		
26	25	BW_Printing	36.817	Copy_Girl	Copy_Boy	[Unknown Resource Name]		
27	26	Prepress	46.817	Copy_Girl	Copy_Boy	[Unknown Resource Name]		
28	27	BW_Printing	43.349	Manager	Copy_Boy	[Unknown Resource Name]		

Sheet1

FINISHED

- Thanks for attending this PCS Advanced Features Webinar! We hope it was helpful.
- The complete one day PCS Advanced course is also available. For more information, contact the ProModel Sales Director that works with your company.
- Remember, help is only an email or phone call away.
- Good luck and happy modeling!

Technical Support
888-776-6633
support@promodel.com
6 am - 6 pm M-F, Mountain Time

Poll #5

ProModel®
Better Decisions—Faster