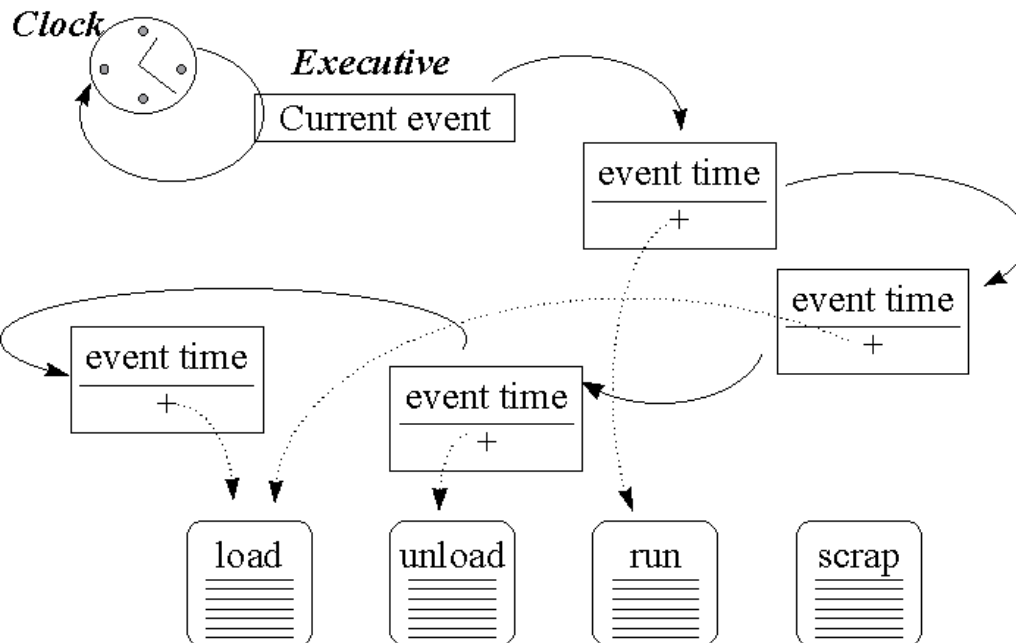


Timothy D. Kuehn
Senior OpenEdge
Consultant
TDK Consulting Services Inc



About Tim Kuehn and TDK Consulting Services Inc

- Full lifecycle Consulting Services in Analysis, Programming, and Related Development Services for Progress ABL applications
- Extensive experience with application modernization as well as white-sheet application design, development, test, and deployment
- Training including sold-out PCA 2014 “Intro to OO training for Procedural Programmers” workshop
- Projects range from short term emergencies and multi-year efforts
- Have a solid track record of multi-year application modernization & upgrade projects

Session Overview

- Definition – what is an event
- What advantages do events bring
- How to setup and run an event
- Implementing a non-event based invoicing system
- Converting to an event-based system
- Refactoring for maximum flexibility
- Using Events for Code Instrumentation

Definition – What is an Event

An event is the notification that “something happened”

- Mouse hover
- Window resized
- Change of user input
- New record being processed
- Logical condition has been met
- Calculation result available
- “Someone might need to do something”

Event Advantages

- Decouples software modules
- Eliminates need for a module to know *what called it*, or *what it's calling*
- Enhances the ability to assemble a collection of component functionality together
- Supports associating an arbitrary number of modules with an event

Setting up an Event

Creating an OO ~~Party~~ Event



Event Demonstration - EventExample/CountTo10.cls

CLASS EventExample.CountTo10:

DEFINE PUBLIC **EVENT CountEvent** SIGNATURE VOID (iCount AS INTEGER).

← Event Definition

METHOD PUBLIC VOID CountValues():
DEFINE VARIABLE iCount AS INTEGER NO-UNDO.

DO iCount = 1 TO 10:

THIS-OBJECT:**CountEvent:Publish**(iCount).

← Event
Implementation

END.

END METHOD.

END CLASS.

Event Demonstration - EventExample/CountTo10Run.p

```
DEFINE VARIABLE oCountTo10 AS EventExample.CountTo10 NO-UNDO.
```

```
DEFINE VARIABLE iCountDisp AS INTEGER NO-UNDO.
```

```
DEFINE FRAME f-Count  
    iCountDisp  
    WITH DOWN.
```

```
oCountTo10 = NEW EventExample.CountTo10().  
oCountTo10:CountEvent:Subscribe("DisplayEvent").  
oCountTo10:CountValues().
```

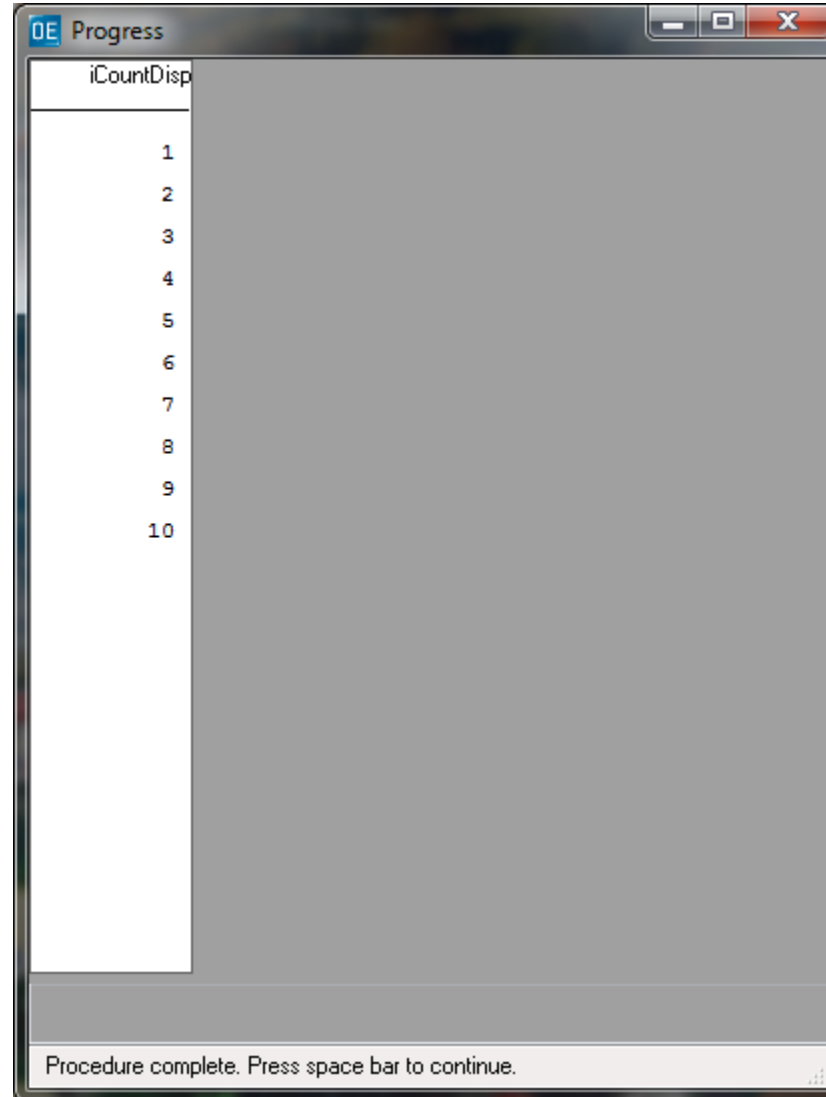
← Event Subscription Source

```
PROCEDURE DisplayEvent:  
    DEFINE INPUT PARAMETER iCountParm AS INTEGER NO-UNDO.
```

← Event Subscription Target

```
DISPLAY iCountParm @ iCountDisp WITH FRAME f-Count.  
DOWN WITH FRAME f-Cnt.  
END PROCEDURE.
```


Event Demonstration - Running EventExample/CountTo10Run.p



Migrating an Object to an Event-Based Paradigm - Before



Migrating an Object to an Event-Based Paradigm

What's coming next:

1. Present a 'non-event' OO structure
2. Change the code to an 'event' OO structure
3. Comparing the before / after dependencies

Migrating to Event Based Paradigm – EventExample/ProcessCustomerTable.cls

CLASS EventExample.ProcessCustomerTable:

```
DEFINE PUBLIC PROPERTY CustomerCount  AS INTEGER NO-UNDO GET. PRIVATE SET.  
DEFINE PUBLIC PROPERTY StateCount    AS INTEGER NO-UNDO GET. PRIVATE SET.
```

METHOD PUBLIC VOID ProcessAllCustomers():

```
FOR EACH Customer  
  NO-LOCK  
  BREAK BY Customer.State:
```

```
  ASSIGN  
    THIS-OBJECT:CustomerCount = THIS-OBJECT:CustomerCount + 1  
    THIS-OBJECT:StateCount    = THIS-OBJECT:StateCount    + 1 WHEN FIRST-OF(Customer.State)  
  .
```

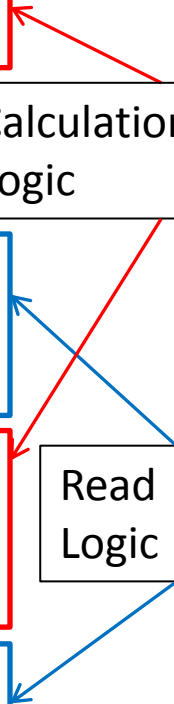
```
END.
```

END METHOD.

END CLASS.

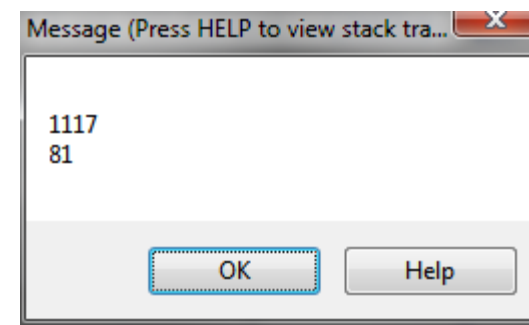
Calculation
Logic

Read
Logic



Defining and Publishing an Event - EventExample/ProcessCustomerTableRun.p

```
/* EventExample/ProcessCustomerTableRun.p */  
  
DEFINE VARIABLE oProcessCustomerTable AS EventExample.ProcessCustomerTable NO-UNDO.  
  
oProcessCustomerTable = NEW EventExample.ProcessCustomerTable().  
  
oProcessCustomerTable:ProcessAllCustomers().  
  
MESSAGE oProcessCustomerTable:CustomerCount SKIP  
        oProcessCustomerTable:StateCount  
VIEW-AS ALERT-BOX.
```



ProcessCustomerTable – an Overview

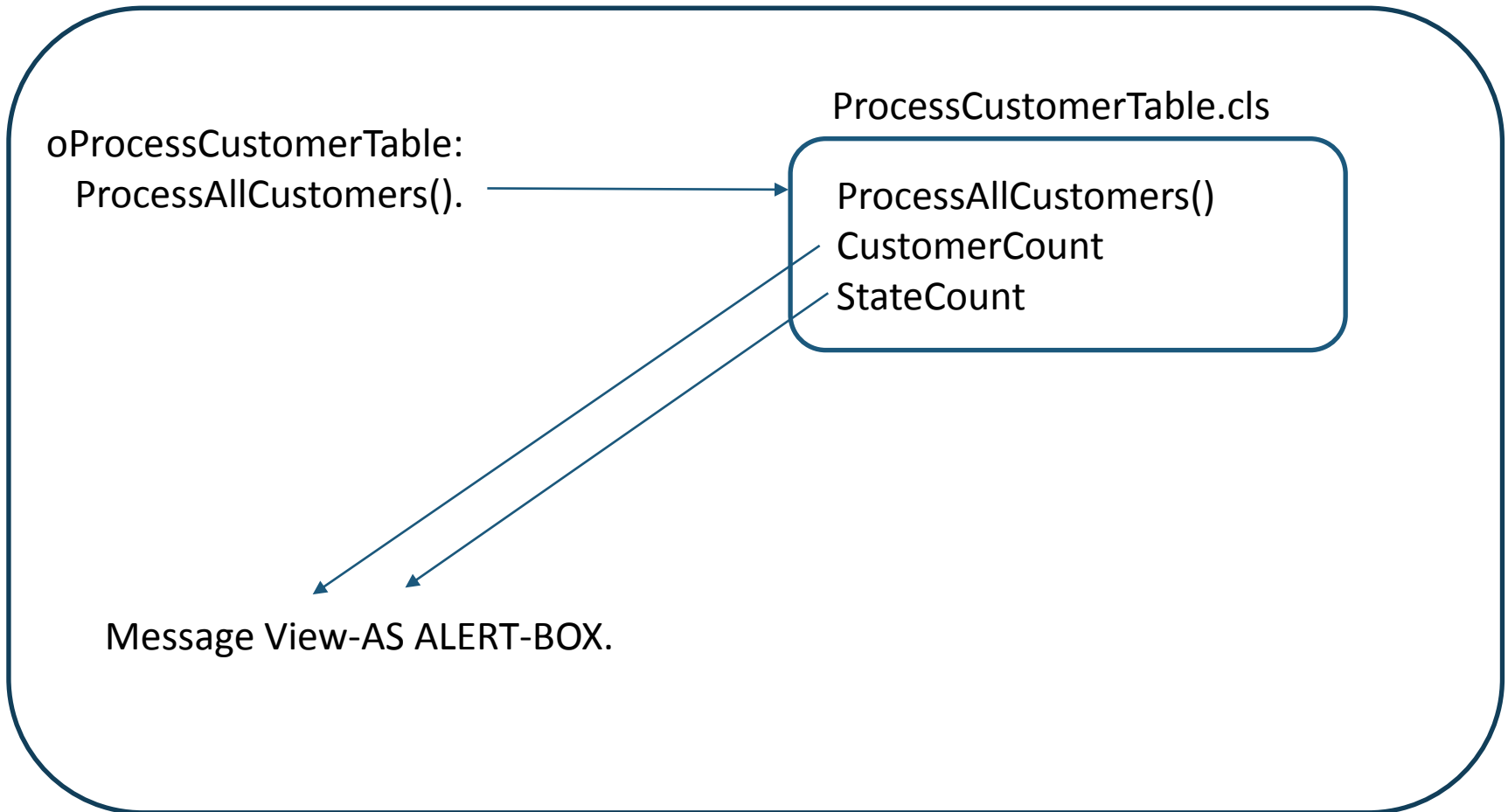
ProcessCustomerTableRun.p

ProcessCustomerTable.cls

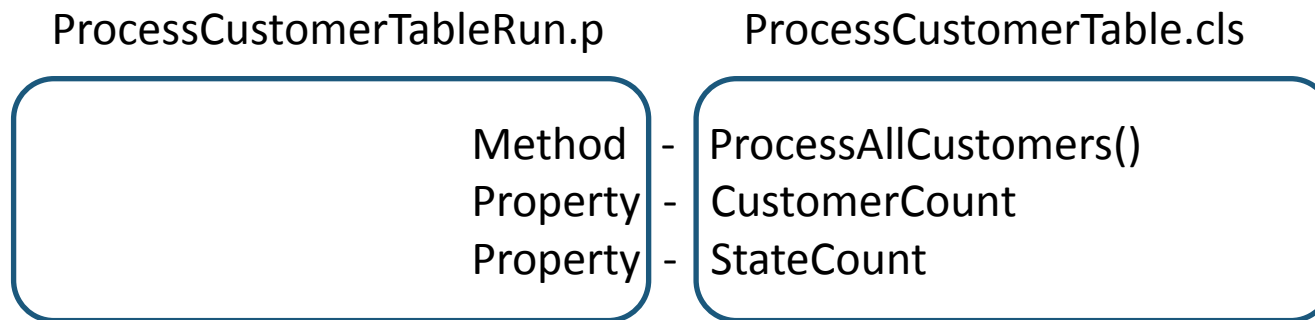
oProcessCustomerTable:
ProcessAllCustomers().

ProcessAllCustomers()
CustomerCount
StateCount

Message View-AS ALERT-BOX.



ProcessCustomerTable – an Overview



ProcessCustomerTableRun.p and ProcessCustomerTable.cls
are tightly coupled together

Put another way – they know more about
each other than is required to do the job

This has implications for development
speed and flexibility

Migrating an Object to an Event-Based Paradigm - After



Migrating an Object to an Event-Based Paradigm - EventExample.ScanCustomerTable

CLASS EventExample.ScanCustomerTable:

DEFINE PUBLIC EVENT CustomerEvent SIGNATURE VOID ().

DEFINE PUBLIC EVENT StateEvent SIGNATURE VOID ().

METHOD PUBLIC VOID ScanAllCustomers():

FOR EACH Customer

BREAK BY Customer.State:

IF FIRST-OF(Customer.State) THEN

THIS-OBJECT:StateEvent:Publish().



Publish when a new state is found

THIS-OBJECT:CustomerEvent:Publish().



Publish for every customer

END.

END METHOD.

END CLASS.

Migrating an Object to an Event-Based Paradigm - EventExample.ScanCount

```
/* EventExample/ScanCount.cls */
```

```
CLASS EventExample.ScanCount:
```

```
DEFINE PUBLIC PROPERTY EventCount AS INTEGER NO-UNDO  
GET. PRIVATE SET.
```

```
METHOD PUBLIC VOID EventIncrement():
```



```
ASSIGN
```

```
THIS-OBJECT:EventCount = THIS-OBJECT:EventCount + 1
```

```
.
```

```
END METHOD.
```

```
END CLASS.
```

Referenced to get the final count



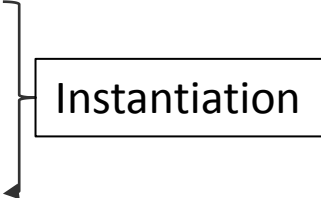
Linked to a Publish() event

Migrating an Object to an Event-Based Paradigm - EventExample/ScanCustomerTableRun.p

```
/* EventExample/ScanCustomerTableRun.p */
```

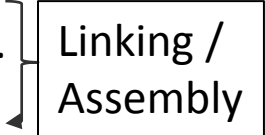
```
DEFINE VARIABLE oScanCustomerTable AS EventExample.ScanCustomerTable NO-UNDO.  
DEFINE VARIABLE oCustomerCount      AS EventExample.ScanCount          NO-UNDO.  
DEFINE VARIABLE oStateCount          AS EventExample.ScanCount          NO-UNDO.
```

```
oScanCustomerTable = NEW EventExample.ScanCustomerTable().  
oCustomerCount      = NEW EventExample.ScanCount().  
oStateCount          = NEW EventExample.ScanCount().
```



Instantiation

```
oScanCustomerTable:CustomerEvent:Subscribe(oCustomerCount:EventIncrement).  
oScanCustomerTable:StateEvent:Subscribe(oStateCount:EventIncrement).
```



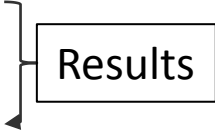
Linking /
Assembly

```
oScanCustomerTable:ScanAllCustomers().
```



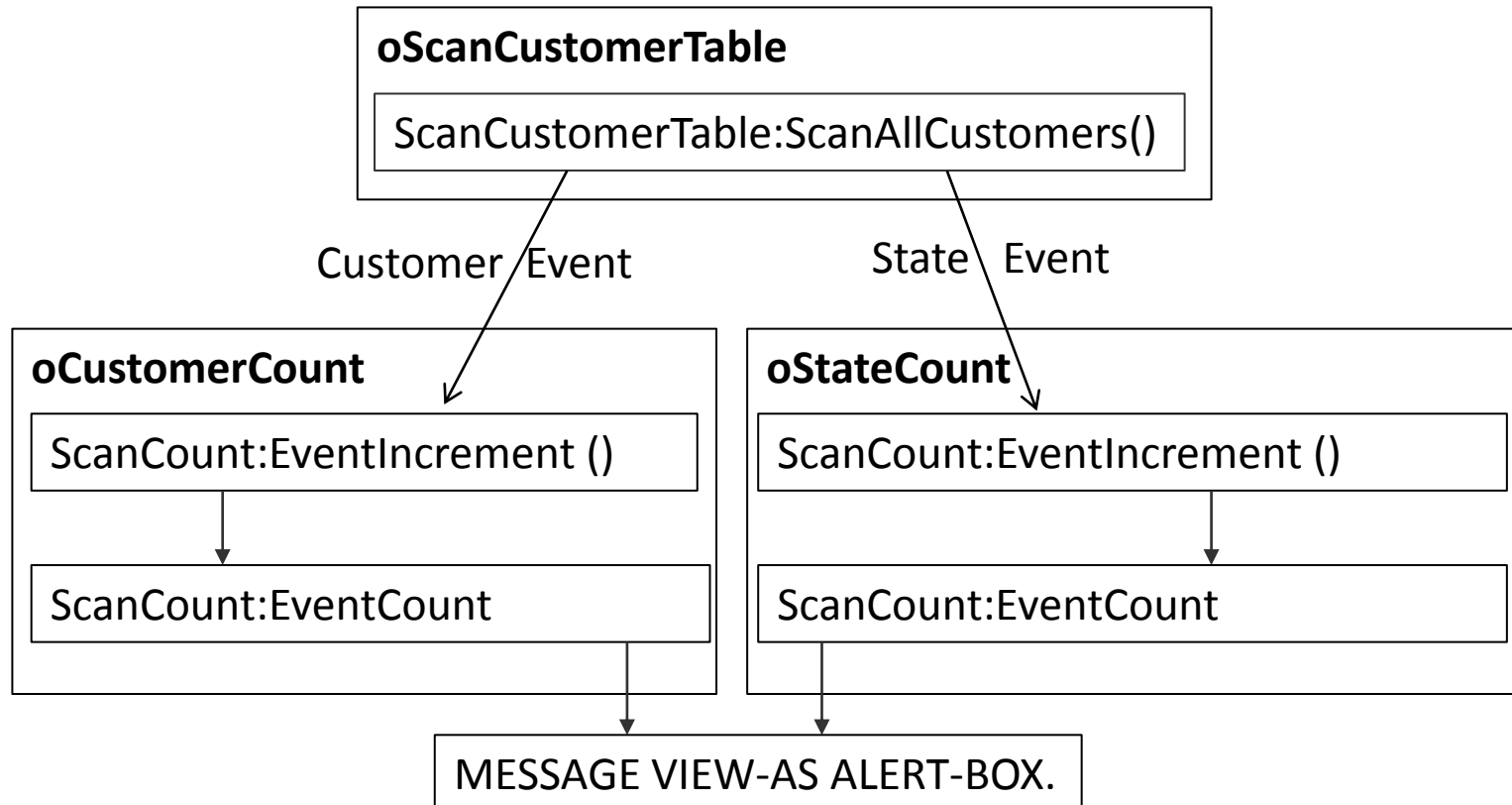
Execution

```
MESSAGE oCustomerCount:EventCount SKIP  
        oStateCount:EventCount     SKIP  
VIEW-AS ALERT-BOX.
```

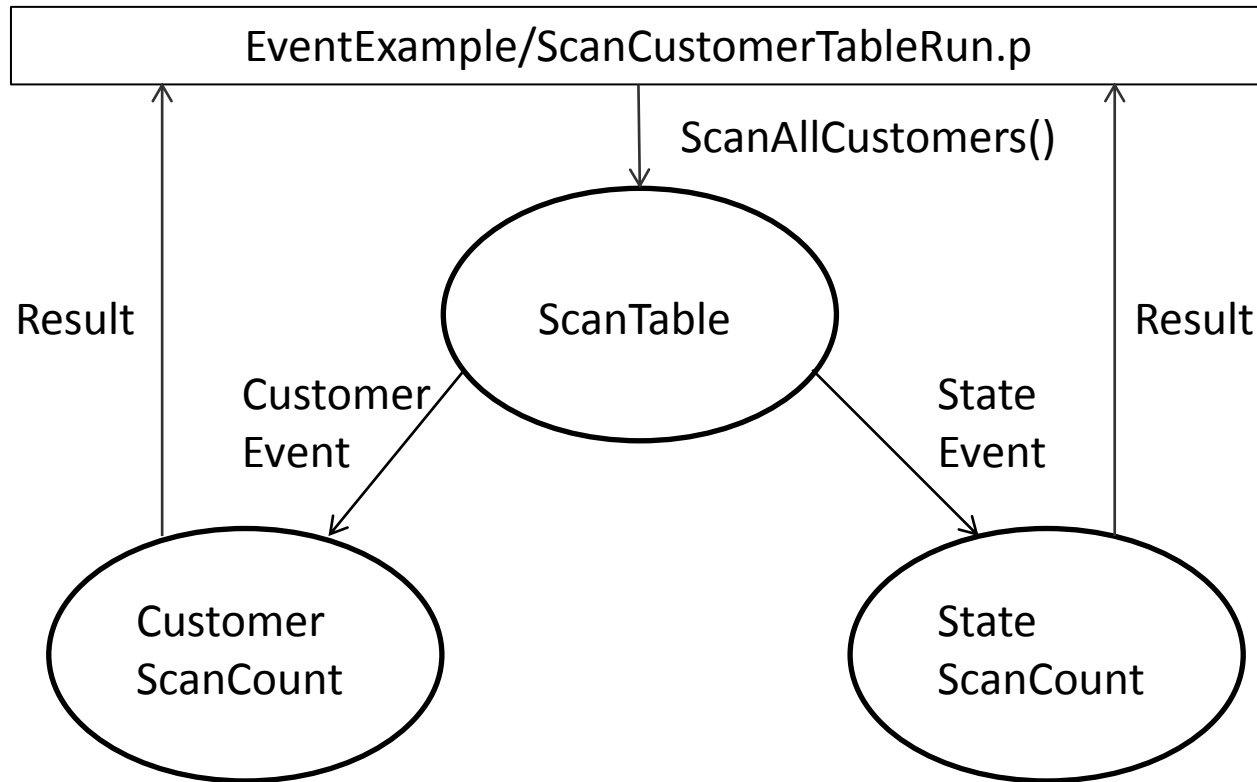


Results

How Everything Interacts



EventExample/ScanCustomerTableRun.p – Who Knows What When?



And then things change....

Boss announces they can charge double for states –
after the new contract date

The Contract Has Changed....

Now What?

It's a New Contract: ScanCountDouble.cls

```
/* EventExample/ScanCountDouble.cls */
```

```
CLASS EventExample.ScanCountDouble:
```

```
    DEFINE PUBLIC PROPERTY EventCount AS INTEGER NO-UNDO  
        GET. PRIVATE SET.
```

```
    METHOD PUBLIC VOID EventIncrement():
```

```
        ASSIGN
```

```
            THIS-OBJECT:EventCount = THIS-OBJECT:EventCount + 2
```

```
        .
```

```
    END METHOD.
```

```
END CLASS.
```

It's a New Contract - EventExample/ScanCustomerTableRun.p

```
/* EventExample/ScanCustomerTableRun.p */
```

```
DEFINE VARIABLE oScanCustomerTable AS EventExample.ScanCustomerTable NO-UNDO.  
DEFINE VARIABLE oCustomerCount      AS EventExample.ScanCount          NO-UNDO.  
DEFINE VARIABLE oStateCount          AS EventExample.ScanCountDouble    NO-UNDO.
```

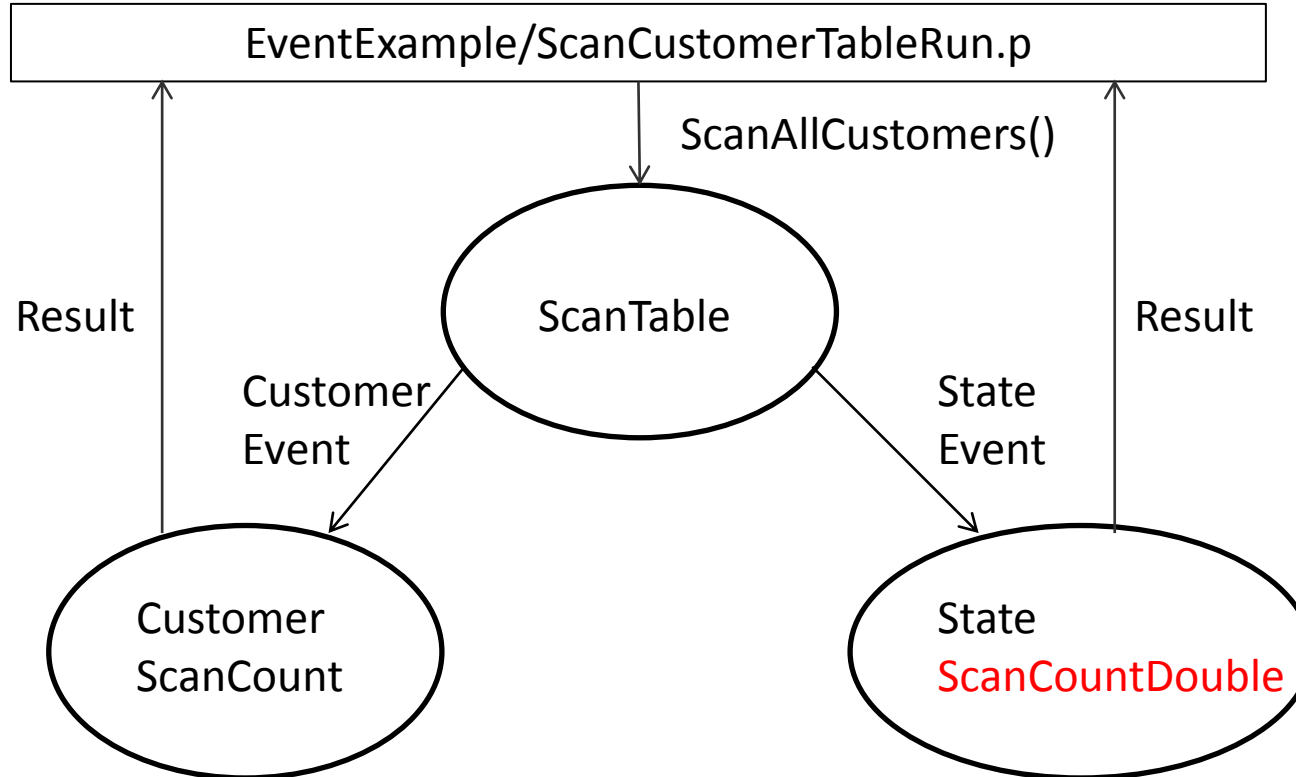
```
oScanCustomerTable = NEW EventExample.ScanCustomerTable().  
oCustomerCount      = NEW EventExample.ScanCount().  
oStateCount          = NEW EventExample.ScanCountDouble().
```

```
oScanCustomerTable:CustomerEvent:Subscribe(oCustomerCount:EventIncrement).  
oScanCustomerTable:StateEvent:Subscribe(oStateCount:EventIncrement).
```

```
oScanCustomerTable:ScanAllCustomers().
```

```
MESSAGE oCustomerCount:EventCount SKIP  
        oStateCount:EventCount     SKIP  
VIEW-AS ALERT-BOX.
```


EventExample/ScanCustomerTableRun.p – Who Knows What?



Refactoring for Additional Flexibility

Replace Class Declarations with Interfaces

Refactoring for Additional Flexibility

That's not bad....but we can do better.

Refactoring for Additional Flexibility- Migrating To Interfaces

1. Define an “iScanCount” interface:

```
INTERFACE EventExample.Interface.iScanCount:
```

```
DEFINE PUBLIC PROPERTY EventCount AS INTEGER NO-UNDO GET. SET.
```

```
METHOD PUBLIC VOID EventIncrement().
```

2. Define an “iScanCustomers” interface:

```
INTERFACE EventExample.Interface.iScanCustomers:
```

```
DEFINE PUBLIC EVENT CustomerEvent SIGNATURE VOID ().
```

```
DEFINE PUBLIC EVENT StateEvent SIGNATURE VOID ().
```

```
METHOD PUBLIC VOID ScanAllCustomers().
```

Refactoring for Additional Flexibility- Migrating To Interfaces

3. Implement an “iScanCount” interface:

CLASS EventExample.Interface.ScanCount

 IMPLEMENTS EventExample.Interface.iScanCount

CLASS EventExample.Interface.ScanCountDouble

 IMPLEMENTS EventExample.Interface.iScanCount

4. Implement an “iScanCustomers” interface:

CLASS EventExample.Interface.ScanCustomerTable

 IMPLEMENTS EventExample.Interface.iScanCustomers

Refactoring for Additional Flexibility- Migrating To Interfaces

```
/* EventExample/Interface/ScanCustomerTableRun.p */

DEFINE VARIABLE oScanCustomerTable AS EventExample.Interface.iScanCustomers NO-UNDO.
DEFINE VARIABLE oCustomerCount      AS EventExample.Interface.iScanCount      NO-UNDO.
DEFINE VARIABLE oStateCount          AS EventExample.Interface.iScanCount      NO-UNDO.

oScanCustomerTable = NEW EventExample.Interface.ScanCustomerTable().
oCustomerCount      = NEW EventExample.Interface.ScanCount().
oStateCount         = NEW EventExample.Interface.ScanCount().

oScanCustomerTable:CustomerEvent:Subscribe(oCustomerCount:EventIncrement).
oScanCustomerTable:StateEvent:Subscribe(oStateCount:EventIncrement).

oScanCustomerTable:ScanAllCustomers().

MESSAGE oCustomerCount:EventCount SKIP
      oStateCount:EventCount      SKIP
VIEW-AS ALERT-BOX.
```

Refactoring for Additional Flexibility – Replace Definitions with Parameters

Replace Class Declarations and NEW with Parameters
Instantiate the classes in an external procedure

Refactoring for Additional Flexibility - Replace Definitions with Parameters

```
/* EventExample/Parm/ScanCustomerTableRun.p */

DEFINE INPUT PARAMETER oScanCustomerTable AS EventExample.Interface.iScanCustomers
                                          NO-UNDO.
DEFINE INPUT PARAMETER oCustomerCount    AS EventExample.Interface.iScanCount
                                          NO-UNDO.
DEFINE INPUT PARAMETER oStateCount       AS EventExample.Interface.iScanCount
                                          NO-UNDO.

oScanCustomerTable:CustomerEvent:Subscribe(oCustomerCount:EventIncrement).
oScanCustomerTable:StateEvent:Subscribe(oStateCount:EventIncrement).

oScanCustomerTable:ScanAllCustomers().

MESSAGE oCustomerCount:EventCount  SKIP
        oStateCount:EventCount     SKIP
VIEW-AS ALERT-BOX.
```


Refactoring for Additional Flexibility - Replace Definitions with Parameters

```
/* EventExample/Parm/ScanCustomerTableRun.p */
```

```

DEFINE VARIABLE oScanCustomerTable AS EventExample.Interface.iScanCustomers NO-UNDO.
DEFINE VARIABLE oCustomerCount      AS EventExample.Interface.iScanCount      NO-UNDO.
DEFINE VARIABLE oStateCount          AS EventExample.Interface.iScanCount      NO-UNDO.

```

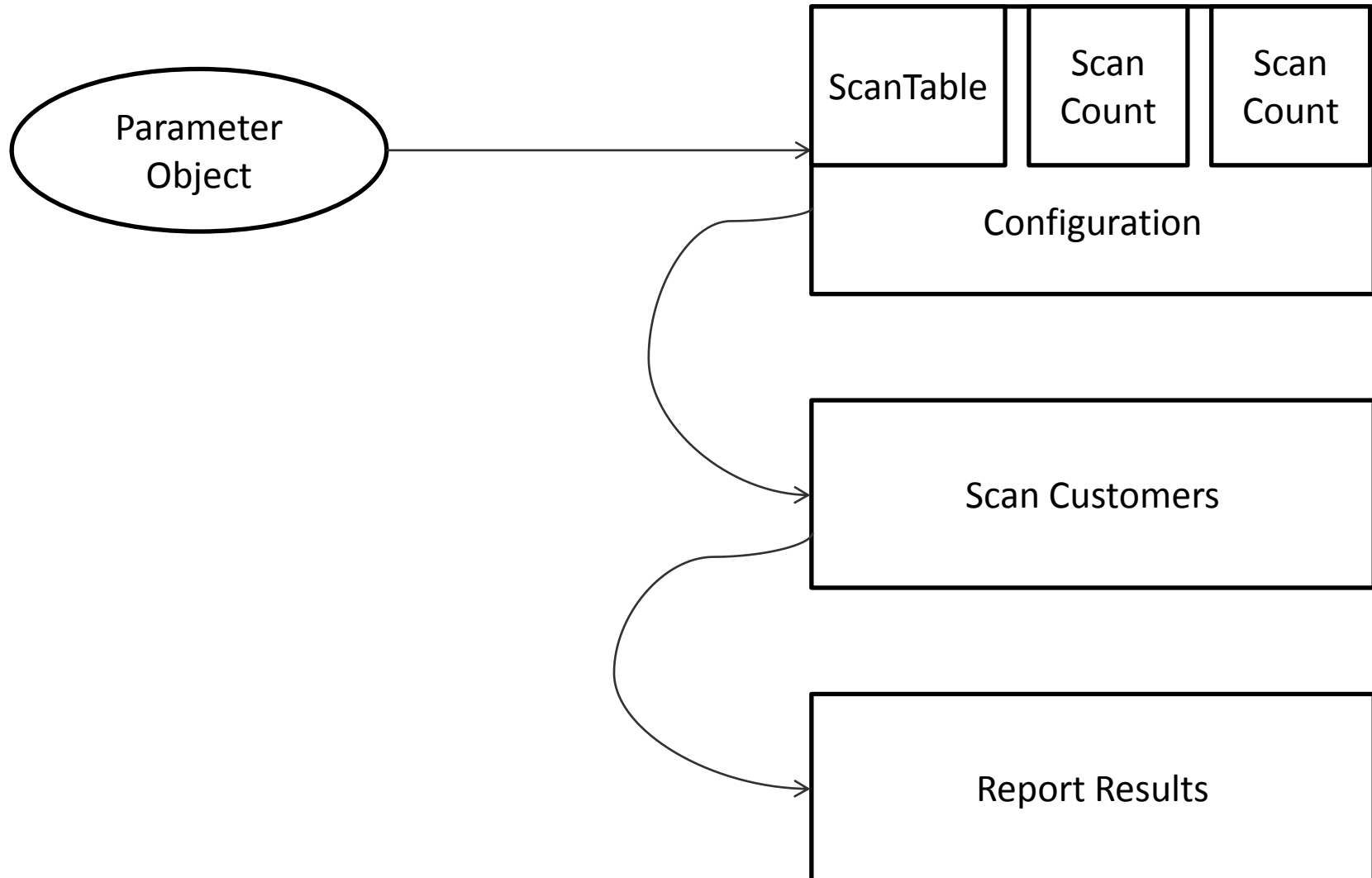
```
oScanCustomerTable = NEW EventExample.Interface.ScanCustomerTable().
oCustomerCount      = NEW EventExample.Interface.ScanCount().
oStateCount          = NEW EventExample.Interface.ScanCount().
```

RUN EventExample/Parm/ScanCustomerTable.p(oScanCustomerTable,
oCustomerCount,
oStateCount).

Refactoring for Additional Flexibility – Move to Parameter Object

- Move object definitions into a Parameter Object
- Call a Configure program to setup the Parameter Object
- Call Invoicing program with Parameter Object
- Call UI program with Parameter Object

Refactoring for Additional Flexibility – Move to Parameter Object



Refactoring for Additional Flexibility

```
/* EventExample/Interface/ScanCustomerTablerun.p */

/* Parm Object declaration */
DEFINE VARIABLE oScanCustomerTableParm AS EventExample.ParmObject.ScanParmObject NO-UNDO.

/* Instantiate the Parm Object */
oScanCustomerTableParm = NEW EventExample.ParmObject.ScanParmObject().

/* Setup the configuration */
RUN EventExample/ParmObject/ScanCustomerTableConfig.p(oScanCustomerTableParm).

/* Get the results */
RUN EventExample/ParmObject/ScanCustomerTable.p(oScanCustomerTableParm).

/* Display the results */
RUN EventExample/ParmObject/ScanCustomerTableMessage.p(oScanCustomerTableParm).
```

Refactoring for Additional Flexibility – Move to Parameter Object

```
/* File: EventExample/Interface/ScanParmObject.cls */
```

```
CLASS EventExample.ParmObject.ScanParmObject:
```

```
DEFINE PUBLIC PROPERTY ScanCustomerTable  
    AS EventExample.Interface.iScanCustomers    GET. SET.
```

```
DEFINE PUBLIC PROPERTY CustomerCount  
    AS EventExample.Interface.iScanCount    GET. SET.
```

```
DEFINE PUBLIC PROPERTY StateCount  
    AS EventExample.Interface.iScanCount    GET. SET.
```

```
END CLASS.
```

Refactoring for Additional Flexibility – Move to Parameter Object

```
/* EventExample/ParmObject/ScanCustomerTableConfig.p */
```

```
DEFINE INPUT PARAMETER oScanCustomerTableParm  
                        AS EventExample.ParmObject.ScanParmObject NO-UNDO.
```

```
/* Setup the configuration */
```

```
oScanCustomerTableParm:ScanCustomerTable  
                        = NEW EventExample.Interface.ScanCustomerTable().
```

```
oScanCustomerTableParm:CustomerCount  
                        = NEW EventExample.Interface.ScanCount().
```

```
oScanCustomerTableParm:StateCount  
                        = NEW EventExample.Interface.ScanCount().
```

Refactoring for Additional Flexibility – Move to Parameter Object

```
/* EventExample/Interface/ScanCustomerTable.p */
```

```
DEFINE INPUT PARAMETER oScanCustomerTableParm  
                        AS EventExample.ParmObject.ScanParmObject NO-UNDO.
```

```
/* Establish event subscriptions */
```

```
oScanCustomerTableParm:ScanCustomerTable:  
    CustomerEvent:Subscribe(oScanCustomerTableParm:CustomerCount:EventIncrement).
```

```
oScanCustomerTableParm:ScanCustomerTable:  
    StateEvent:Subscribe(oScanCustomerTableParm:StateCount:EventIncrement).
```

```
/* Scan the table and tally the results */
```

```
oScanCustomerTableParm:ScanCustomerTable:ScanAllCustomers().
```

Refactoring for Additional Flexibility – Move to Parameter Object

```
/* EventExample/ParmObject/ScanUsingParmObjectMessage.p */
```

```
DEFINE INPUT PARAMETER oScanCustomerTableParm  
                        AS EventExample.ParmObject.ScanParmObject NO-UNDO.
```

```
MESSAGE oScanCustomerTableParm:CustomerCount:EventCount SKIP  
        oScanCustomerTableParm:StateCount:EventCount      SKIP  
VIEW-AS ALERT-BOX.
```


The General Case for Events –Code Instrumentation

Walk through of Notify code to instrument
code using Events if there's time

The General Case for Events -Conclusion

- Provides de-coupling of software modules
- Allows for writing objects without worrying about what's calling it, or what it's calling
- Allows for late-binding of objects and assembling arbitrary collections of object functionality on the fly
- Supports linking widely separated objects
- Increases development flexibility and can reduce overall development and maintenance time

Thanks for your Time and Attention!

Thank you for your time and attention!

**For all your Progress ABL Development
Service and Training needs:**

**Tim Kuehn
TDK Consulting Services Inc.
tim.kuehn@gmail.com
519-576-8100**