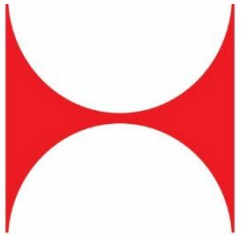
An Introduction to OO ABL Events

by Tim Kuehn



TDK CONSULTING

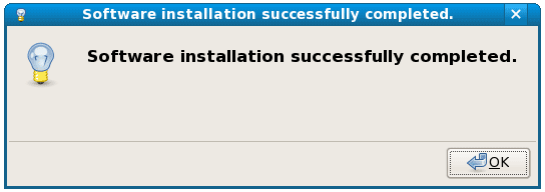
www.tdkcs.com

About TDK Consulting

TDK Consulting Services

..delivers full life cycle development from a clean sheet of paper

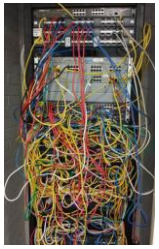
to a finished product



…specializes in the development of software engines

that wrap complex business logic in easy to use APIs

..can refactor code like this:



into code like this:

…develops and delivers training solutions to turn questions

into knowledge and understanding

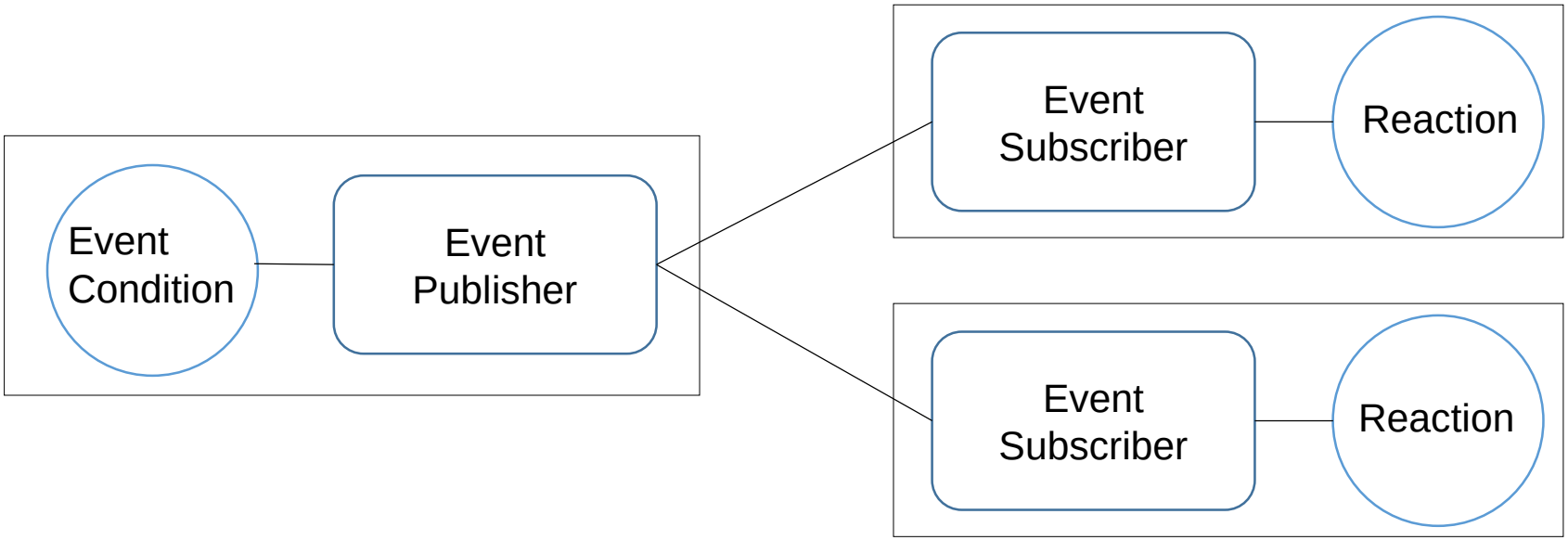
"Innovation you can build on."

www.tdkcs.com

What is Publish/Subscribe?

In it's simplest form – Publish / Subscribe is a way to

a) detect an event condition that will
b) raise an event notification to
c) inform interested code blocks so they can react to it
d) in a loosely coupled fashion



What is Publish/Subscribe?

OO Publish Subscribe is a:

- mechanism for sending messages from a *publisher* to zero or more *subscribers*,
- way to decouple a *publisher* from it's *subscribers*,
- way to decouple a *subscriber* from it's *publisher*,
- way to create a *one-to-any* relationship between the *publisher* and it's *subscribers*

What is Publish/Subscribe used for?

OO Publish / Subscribe is used for:

- chaining one event to one or more successive events,
- creating arbitrary collections of logic on the fly,
- creating systems which detect and react to events,
- handling UI actions,
- Whatever your mind dreams up!

What are some Publish/Subscribe examples?

Example of Publish / Subscribe:

Twitter:

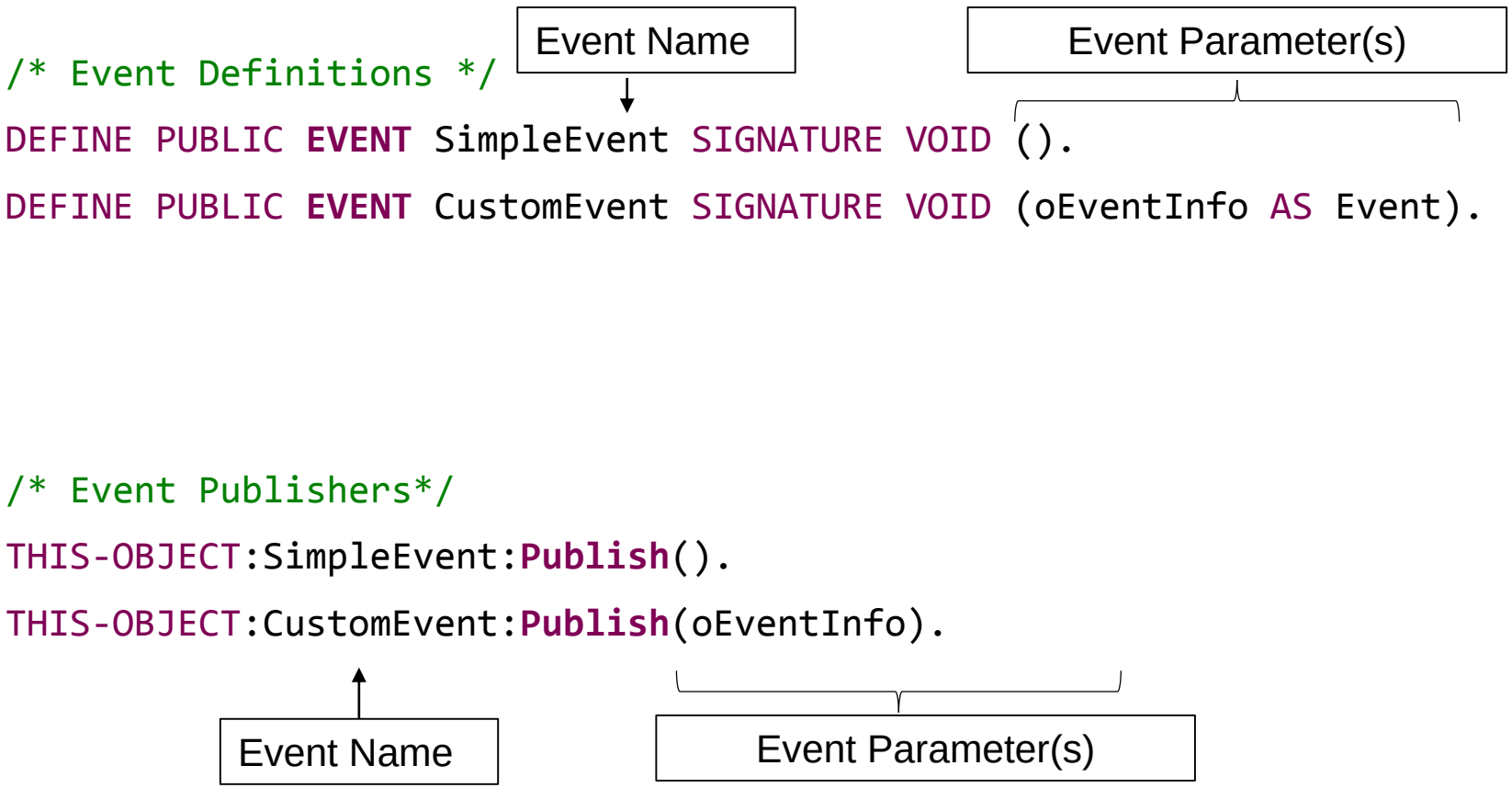
- Progress Developer “follows” Progress Software
- Progress Software “tweets” on Twitter
- Twitter sends an *update event* to all of Progress Software followers,
- Progress Developer gets the tweet

Windows Explorer:

- Two explorer windows are opened
- Each window subscribes to an OS “update event” for the directory
- One window is used to delete files
- The explorer window gets list of files and instructs Windows to delete each file
- After deleting a file, Windows sends *update event* to the explorer windows
- Each explorer windows updates the list of displayed files

What does Publish/Subscribe code look like?

OOABL Event code looks like:

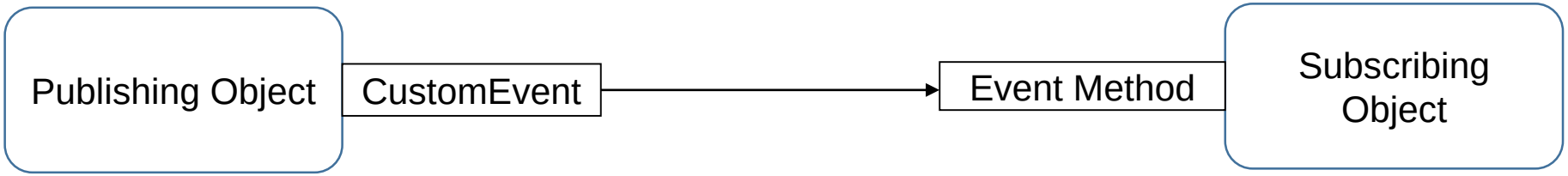


What does Publish/Subscribe code look like?

Linking an Event Publisher to an Event Subscriber:

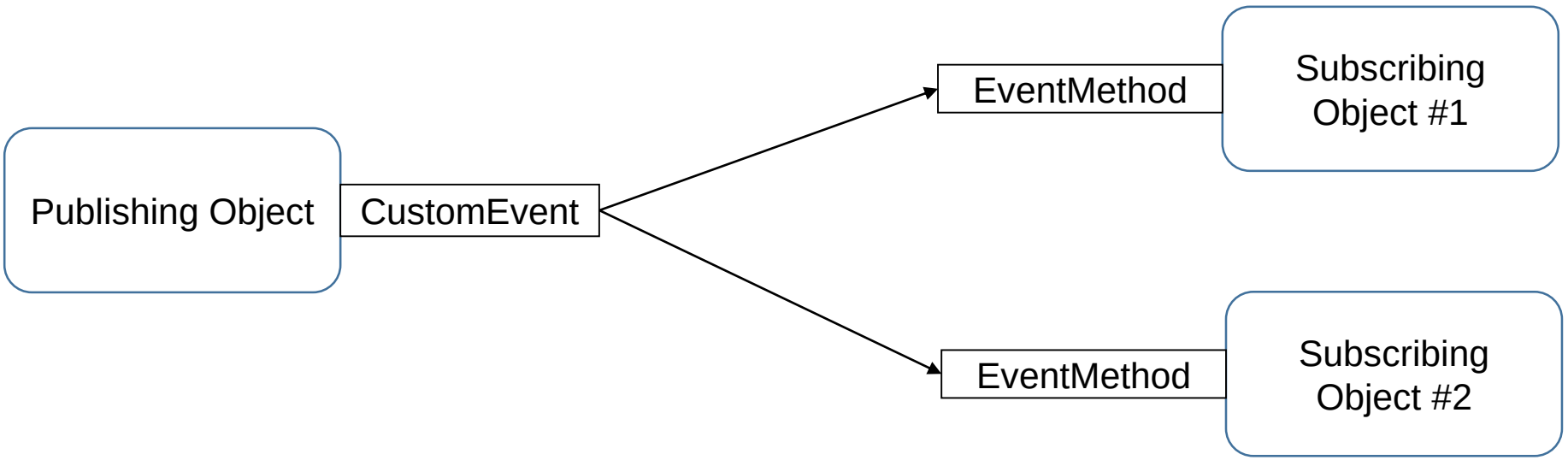
```
/* Event Definitions */
DEFINE PUBLIC EVENT CustomEvent SIGNATURE VOID (oEventInfo AS Event).

/* Event Subscriber */
oPublishingClass:CustomEvent:Subscribe(oSubscribingClass:EventMethod).
```

- Publishing Object
- Publishing Object Event Name
- Subscribing Object
- Subscribing Object Method Name



What does Publish/Subscribe look like?

Linking a single Event Publisher to multiple Event Subscribers



Publish / Subscribe Implementation Example


```
/* Answers/PublishSubscribe/Base/Demo/Publisher.cls */
CLASS Presentation.PublishSubscribe.Base.Demo.Publisher:
DEFINE PUBLIC EVENT PublishExample  SIGNATURE VOID ().

METHOD PUBLIC VOID PublishEvent():
THIS-OBJECT:PublishExample:Publish().
END METHOD.
END CLASS.

/* Answers/PublishSubscribe/Base/Demo/Subscriber.cls */
CLASS Presentation.PublishSubscribe.Base.Demo.Subscriber:
METHOD PUBLIC VOID EventSubscriber():
MESSAGE "Event!"
   VIEW-AS ALERT-BOX.
END METHOD.
END CLASS.
```

Event Definition

Event Publisher

Event subscriber method

Publish / Subscribe Implementation Example


```
/* Presentation/PublishSubscribe/Base/Demo/PubSubExample.p   */
USING Presentation.PublishSubscribe.Base.Demo.*.

DEFINE VARIABLE oPublish    AS Publisher    NO-UNDO.
DEFINE VARIABLE oSubscribe  AS Subscriber   NO-UNDO.
DEFINE VARIABLE iCnt        AS INTEGER      NO-UNDO.

oPublish    = NEW Publisher().
oSubscribe  = NEW Subscriber().

oPublish:PublishExample:Subscribe(oSubscribe:EventSubscriber).

DO iCnt = 1 TO 3:
    oPublish:PublishEvent().
END.
```

Publisher Class

Subscriber Class

Link Publisher Event to Subscriber Method

Publish Event to Subscribers

Demo Time!

Code
Demo

Building an event processing system a class at a time

Building an Event Processing System

Modelling the interaction between a Train, a Railfan, and his Camera

Railfan

Camera

Train Engine

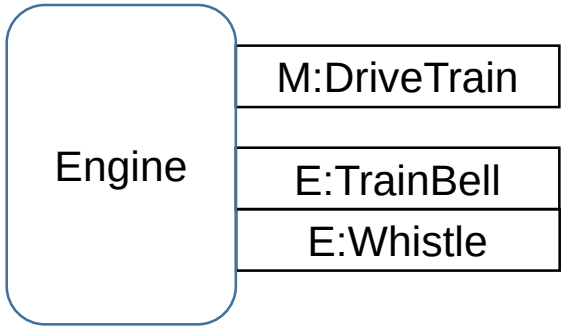
Model the:

- engine ringing it's bell and blowing it's whistle
- railfan hearing the bell and whistle
- railfan taking a picture
- sound the camera makes
- when enough pictures have been taken, the camera will display a "memory full" status

Images from http://blog.traingeek.ca/

Building an Event Processing System

Train Engine Model:



Building an Event Processing System

Train Engine Code:

```
CLASS Answers.PublishSubscribe.Base.Train.Engine:

DEFINE PUBLIC EVENT Whistle     SIGNATURE VOID(chSound AS CHARACTER).
DEFINE PUBLIC EVENT TrainBell   SIGNATURE VOID(chSound AS CHARACTER).

/*************************************************************************/
METHOD PUBLIC VOID DriveTrain():
DO WHILE TRUE:

    PAUSE 1 NO-MESSAGE.
    THIS-OBJECT:TrainBell:Publish("Ding!").

    IF RANDOM(1, 10) > 7 THEN
        THIS-OBJECT:Whistle:Publish("Toot!").

END.
END METHOD.

END CLASS.
```

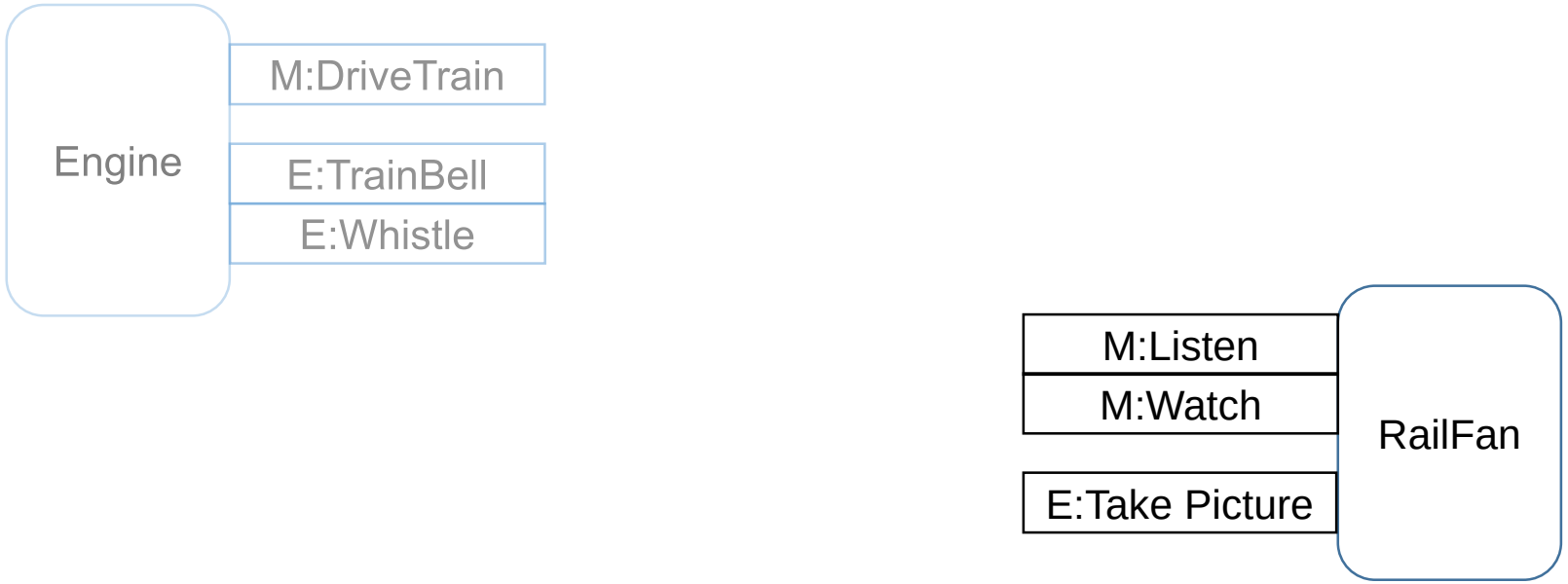
Train "event sources"

"Simulate Train" method

Simulate train bell

Simulate train whistle

Building an Event Processing System



Building an Event Processing System

RailFan Code:

```
CLASS Answers.PublishSubscribe.Base.Train.RailFan:

DEFINE PUBLIC EVENT TakePicture SIGNATURE VOID().

DEFINE VARIABLE iHeardCount     AS INTEGER      NO-UNDO.

    /* Frame Definition */

METHOD PRIVATE VOID ShowMessage(chMsg AS CHARACTER):
   /* DISPLAY statement */
END METHOD.

 /* Event handler methods on the next frame… */

END CLASS.
```

RailFan “event source”

Report what the railfan heard and saw

Building an Event Processing System

More RailFan Code:

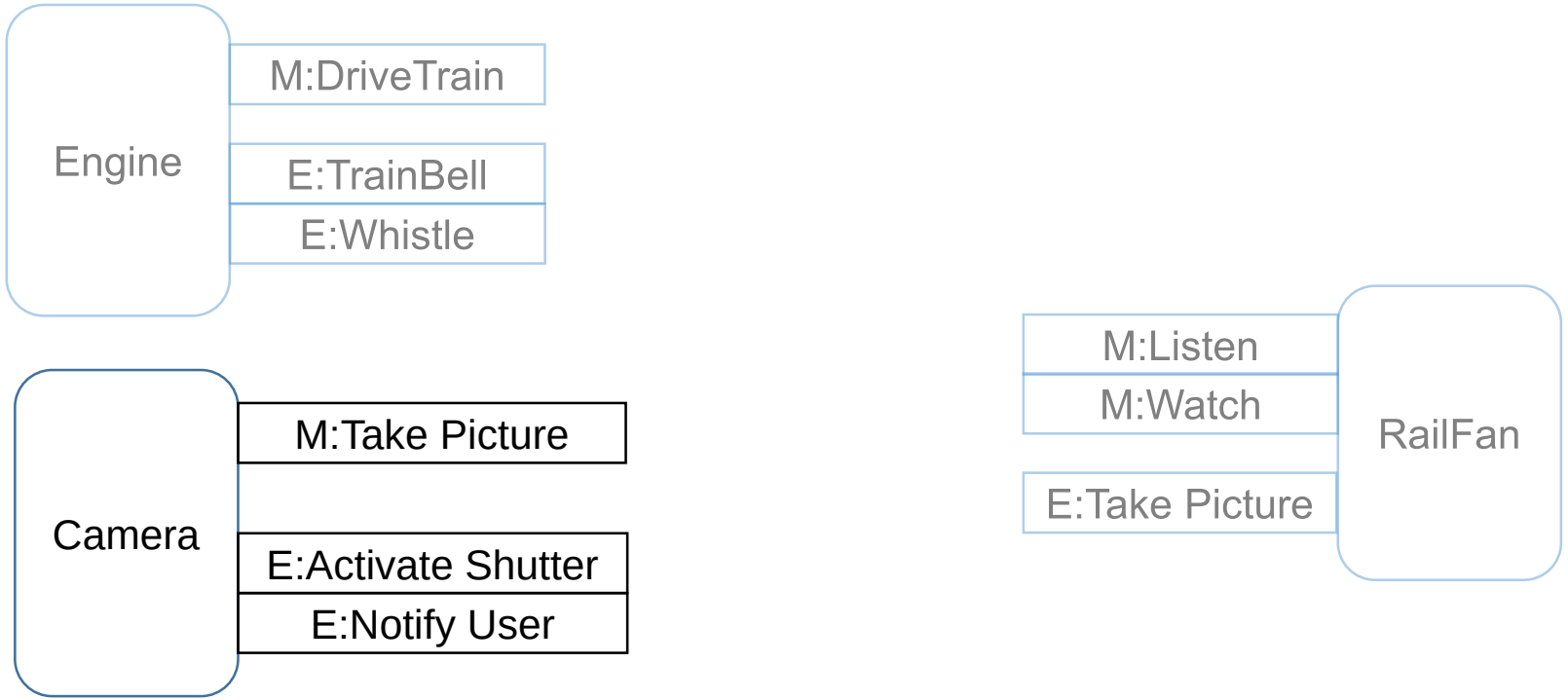
```
METHOD PUBLIC VOID Watch(chWatch AS CHARACTER):
THIS-OBJECT:ShowMessage(chWatch).
END METHOD.

METHOD PUBLIC VOID Listen(chSoundParm AS CHARACTER):
THIS-OBJECT:ShowMessage(chSoundParm).

ASSIGN iHeardCount= iHeardCount + 1.

IF iHeardCount MODULO 10 = 0 THEN
    THIS-OBJECT:TakePicture:Publish().

END METHOD.
```

Simulate the Railfan seeing something

Simulate the Railfan hearing something

After every 10 sounds heard, take a picture

Building an Event Processing System



Building an Event Processing System

The Camera:

```
CLASS Answers.PublishSubscribe.Base.Train.Camera:

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

DEFINE PUBLIC EVENT ActivateShutter SIGNATURE VOID(chSound AS CHARACTER).
DEFINE PUBLIC EVENT NotifyUser  SIGNATURE VOID(chNotifcation AS CHARACTER).

METHOD PUBLIC VOID TakePicture():

IF THIS-OBJECT:PictureCount < 5 THEN
    DO:
    THIS-OBJECT:PictureCount = THIS-OBJECT:PictureCount + 1.
    THIS-OBJECT:ActivateShutter:Publish("Snap " + STRING(THIS-OBJECT:PictureCount)).
    END.
ELSE
    THIS-OBJECT:NotifyUser:Publish("Camera Full.").

END METHOD.

END CLASS.
```

Camera “event sources”

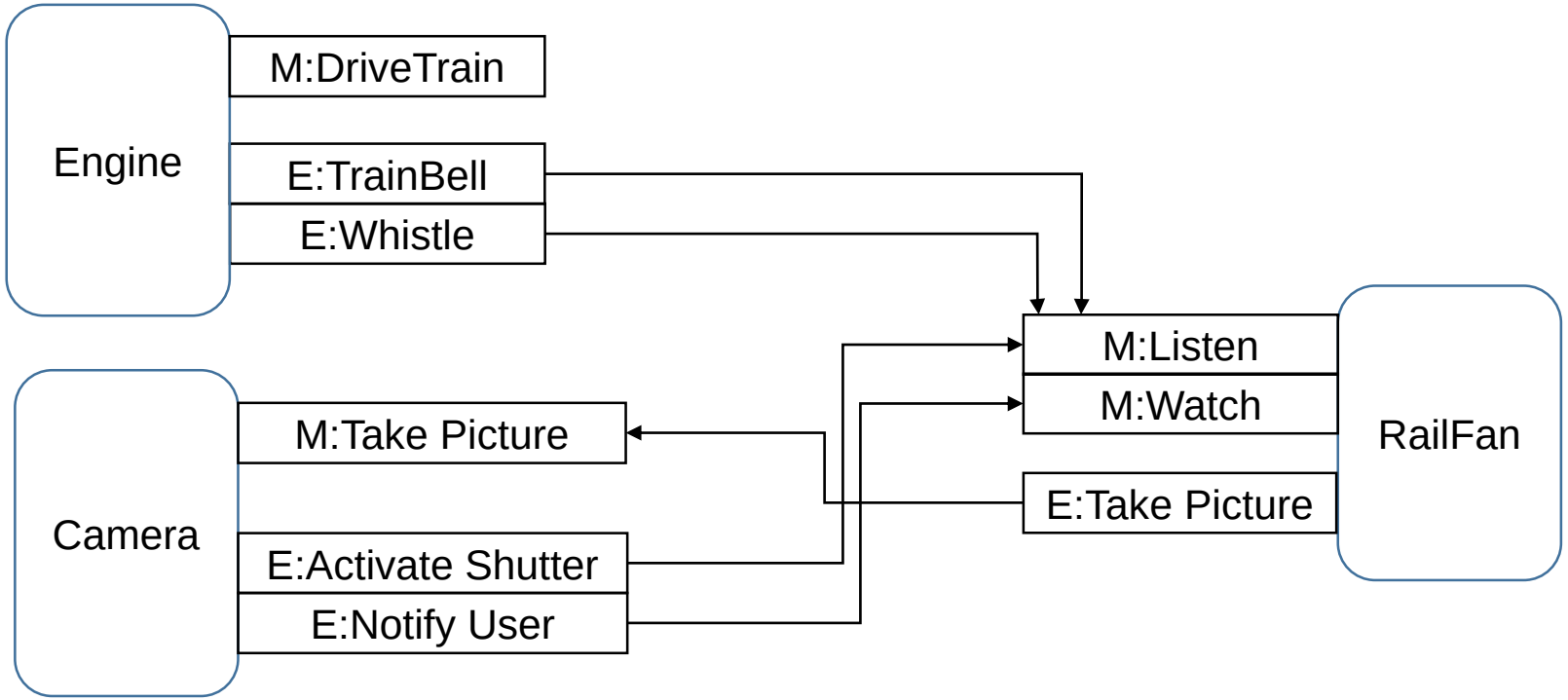
Simulate taking a picture

If less than 5 pics taken, then take the picture

otherwise publish a “memory full” event

Building an Event Processing System

Gluing all the classes together:

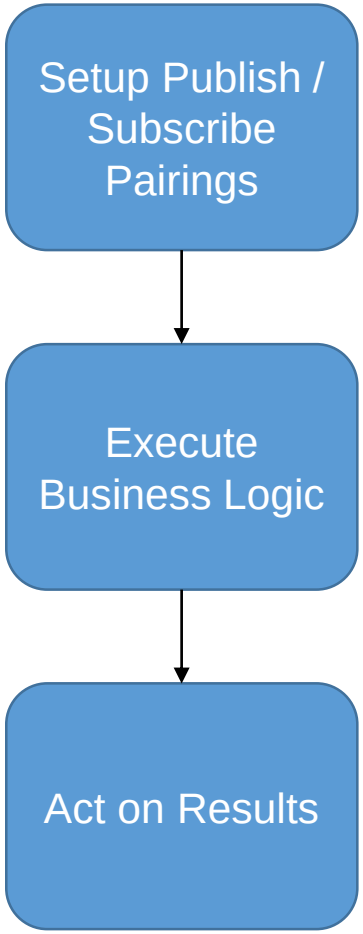



```
oEngine:Whistle:Subscribe(oRailFan:Listen).
oEngine:TrainBell:Subscribe(oRailFan:Listen).

oRailFan:TakePicture:Subscribe(oCamera:TakePicture).

oCamera:ActivateShutter:Subscribe(oRailFan:Listen).
oCamera:NotifyUser:Subscribe(oRailFan:Watch).
```

Code in Action

Publish/Subscribe within a Class

Events can be used to configure business logic process within a class



Publish/Subscribe within a Class

Execute logic based on type of employee

```
CLASS Answers.PublishSubscribe.Base.EventLogic.EventLogic:

/* Class Events */
DEFINE PRIVATE EVENT DayShift   SIGNATURE VOID ().
DEFINE PRIVATE EVENT NightShift SIGNATURE VOID ().

/* Class Variables */
DEFINE VARIABLE iStraightPay AS INTEGER NO-UNDO.
DEFINE VARIABLE iDoublePay   AS INTEGER NO-UNDO.


METHOD PRIVATE VOID StraightPay():
ASSIGN iStraightPay = iStraightPay + 1.
END METHOD.

METHOD PRIVATE VOID DoublePay():
ASSIGN iDoublePay = iDoublePay + 2.
END METHOD.

   /* Example methods on next slide */

END CLASS.
```

Events

Counters

Event Listeners / Business Logic

Publish/Subscribe within a Class

Simulate processing 10 dayshift employees and 20 nightshift employees

```
METHOD PRIVATE VOID ExecuteLogic(chMsg AS CHARACTER):
DEFINE VARIABLE iCnt AS INTEGER NO-UNDO.

DO iCnt = 1 TO 10:
    THIS-OBJECT:DayShift:Publish().
END.

DO iCnt = 1 TO 20:
    THIS-OBJECT:NightShift:Publish().
END.

MESSAGE chMsg                           SKIP
        "Straight Pay"  iStraightPay    SKIP
        "Double Pay"    iDoublePay
   VIEW-AS ALERT-BOX.

END METHOD.
```

Publish/Subscribe within a Class

Execute logic based on type of employee

```
/*************************************************************************/
METHOD PUBLIC VOID RunHumanExample():
THIS-OBJECT:DayShift:Subscribe(THIS-OBJECT:StraightPay()).
THIS-OBJECT:NightShift:Subscribe(THIS-OBJECT:DoublePay()).

THIS-OBJECT:ExecuteLogic("Human").

END METHOD.

/*************************************************************************/
METHOD PUBLIC VOID RunVampireExample():
THIS-OBJECT:DayShift:Subscribe(THIS-OBJECT:DoublePay()).
THIS-OBJECT:NightShift:Subscribe(THIS-OBJECT:StraightPay()).

THIS-OBJECT:ExecuteLogic("Vampire").

END METHOD.
```

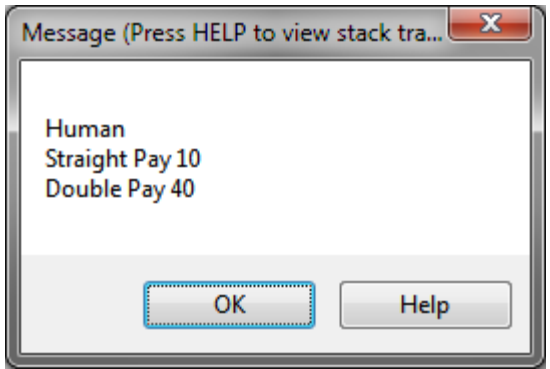
Setup Links

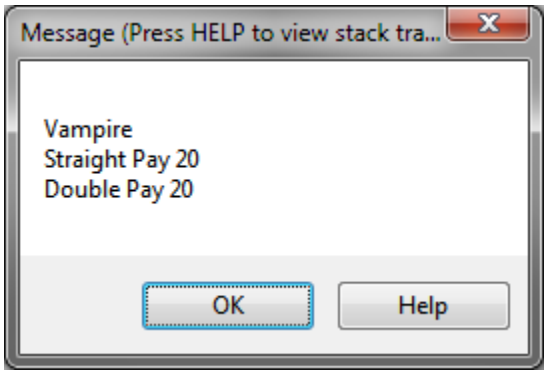
Execute Logic

Setup Links

Execute Logic

Results





Publish/Subscribe “Gotchas”

What are some potential concerns with Publish / Subscribe?

- Subscriber methods are not run in any particular order,
- Decreased ability to modify a Publisher API and the published data’s structure,
- Inflexible semantic coupling – changing the publish data structure can be difficult,
- Can be confused with network messaging “pub / sub” architectural pattern
- Certain pub/sub arrangements can stop garbage collection from working

Note: Publish / Subscribe should *not* be used to bypass inheritance

Questions?

Sessions by TDK Consulting

Session	Title	Description
198	OO ABL Events	A talk on OO "Events" w/bonus material
206	Migrating to Multi-tenancy	A step-by-step walk-through of merging two Sports2000 databases into a single MultiTenant database.
207	ABL Buffers and Queries in OO Wrappers	A research report on the development of an OO based Data Access layer

About this presentation

This presentation is based on the "OO Events" portion of the "Introduction to OO for Procedural Developers" training course offered by TDK Consulting.

For course information see www.tdkcs.com

Thank you for your time!

Tim Kuehn
TDK Consulting Services Inc.
www.tdkcs.com

Email: tim.kuehn@tdkcs.com
Ph: 519-576-8100
Mobile: 519-781-0081
Twitter: @tdkcs