
Actian
OpenROAD 6.2 – New Features in Detail, Part I
New in OpenROAD 6.2 – For OpenROAD developers.
Third of four presentations. Assumes attendance at the first.
Sean Thrower
May, 2015
Confidential © 2014 Actian Corporation

Disclaimer

This document is for informational purposes only and is subject to change at any time without notice. The information in this document is proprietary to Actian and no part of this document may be reproduced, copied, or transmitted in any form or for any purpose without the express prior written permission of Actian.

This document is not intended to be binding upon Actian to any particular course of business, pricing, product strategy, and/or development. Actian assumes no responsibility for errors or omissions in this document. Actian shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials. Actian does not warrant the accuracy or completeness of the information, text, graphics, links, or other items contained within this material. This document is provided without a warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.

Overview

- OpenROAD 6.2 – New generic features in more detail, Part I
  - The third of four presentations covering OpenROAD 6.2
    - This presentation is the first of two reviewing in detail the new features in this release
  - Features illustrated in the presentation will require the first OpenROAD 6.2 patch
    - p14746 or later

This presentation assumes that you have seen the Overview presentation.

If you have not, we recommend that you first view the recording of the Overview presentation, available at
https://actian.webex.com/actian/j.php?MTID=m013e2f1d62551cb8f21d7f92dd145f6f

p14746 OR 6.2.0 (int.w32/00)

OpenROAD 6.2 Objectives

- OpenROAD 6.2 is the outcome of a set of objectives pursued systematically over the last few years:
  - I - Provide specific support for certain in-demand business requirements
  - II - Reduce the cost of OpenROAD development (time, headcount, skill, maintenance)
  - III - Improve the deployment of OpenROAD
  - IV - Provide generic enabling facilities to underpin the new features and also future ones
  - V - Implement all these changes in a way that logically extends the OpenROAD metamodel and fills important gaps in it
- The fourth objective is addressed in this and the subsequent presentation

In the Overview Presentation (reminder):

- Meeting Business Requirements
  - Restyling to up-to-date look-and-feel(s)
  - Same-code (unchanged code) transformations
  - Generated userclasses and displays
  - Active-map and Booking/Allocation capabilities
  - Richer out-of-the-box capabilities
  - Easier deployment
- Improving OpenROAD ROI

Meeting OpenROAD 6.2 Objectives …

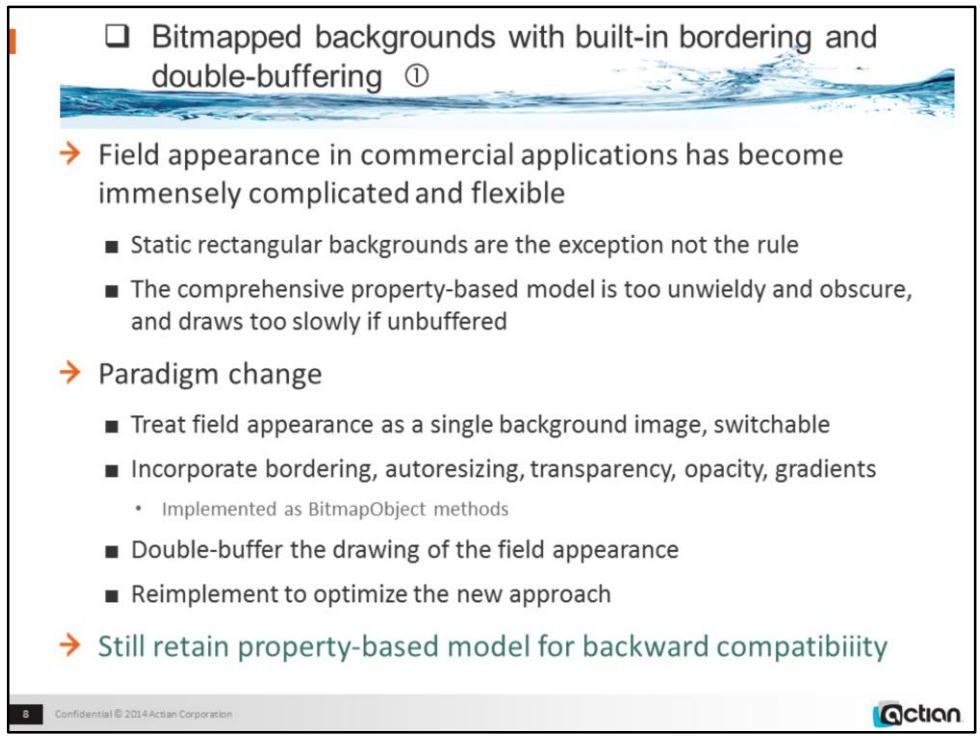
- IV - Providing generic enabling facilities to underpin the new client-oriented features, and future ones

IV - Providing generic enabling facilities to underpin the business-need features, both current and future

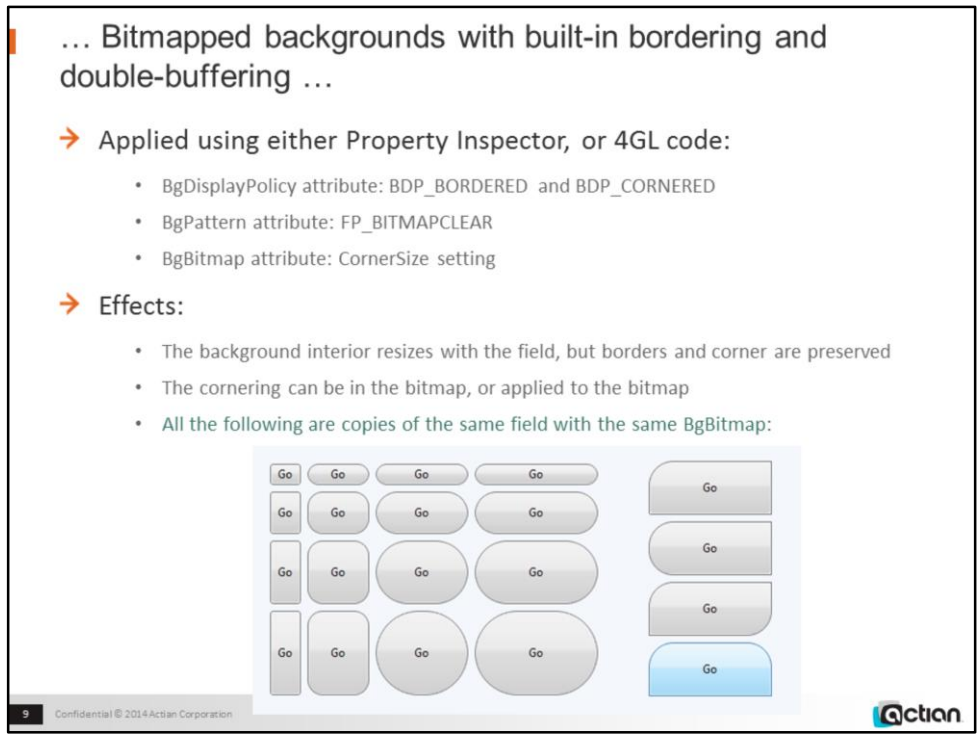
1. Bitmapped backgrounds with built-in bordering and double-buffering
2. Compound bitmaps, sprites and animations
3. Tagged Values/Items
4. Storable defined-behaviours
5. Helper classes and Setup frames
6. Many enabling property and method changes to field and data classes
   - TreeViews, TableFields, TabFolders; String & BitmapObjects; and much more
7. Enhanced PropertyChanger facilities
8. Database and display heuristics
9. Downloadable IngresNet
10. LoadnRun deployment

Items 6 to 8 are covered in the fourth presentation

Items 9 and 10 were covered in the second presentation



Numbers in circles in the title refer to the numbering of the sub-objectives on the previous page



Demo:

D201504_BitmappedBkgFields -cBitmappedBkg_Rounding

Note that all these fields are identical apart from size: same bitmap, same BgDisplayPolicy (BDP_CORNERED), BgPattern (FP_BITMAPCLEAR), CornerSize (3 pixels).

Note that the appearance of the field, other than the text, is entirely due to the bitmap, including the edges and corners.

Click the Go button (to apply a variety of cornersizes to the fields)

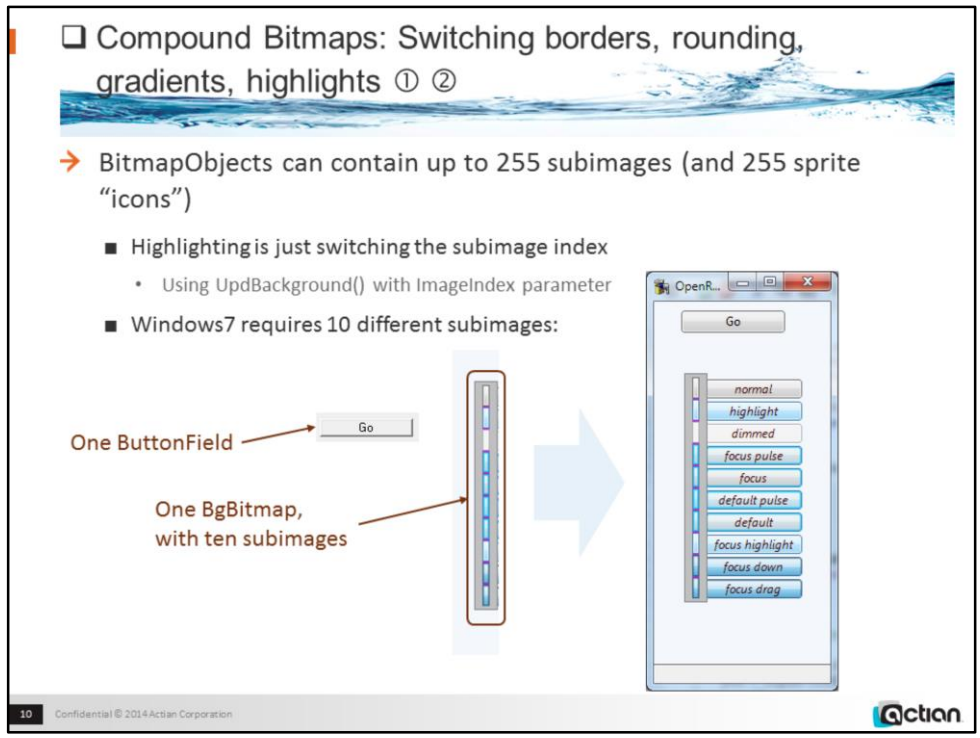
Note that the borders are still preserved, the corners are transparent and antialiased at all radii, the corners can be individually rounded.

```
/*
**      Set a rounding of 5 pixels for each corner
*/
btn.UpdBackground(cornersize=5);

/*
**      Set independent roundings for each corner
**      (this one curves the top two corners, rightangles the bottom two)
*/
#define $TL             '256**0'        -- topleft
#define $TR             '256**1'        -- topright
#define $BR             '256**2'        -- bottomright
#define $BL             '256**3'        -- bottomleft
```

```
btn.UpdBackground(cornersize=25*$TL + 25*$TR + 0*$BR + 0*$BL);
```

Can also define BorderWidth, BorderStyle.



Demo:

D201504_BitmappedBkgFields -cBitmappedBkg_ImageSwitch

Click the Go button

Note that on the left is the single bitmap (containing 10 images) that all the buttons on the right use.

Note that initially the imageindex is unset, so all of the fields are using the first image

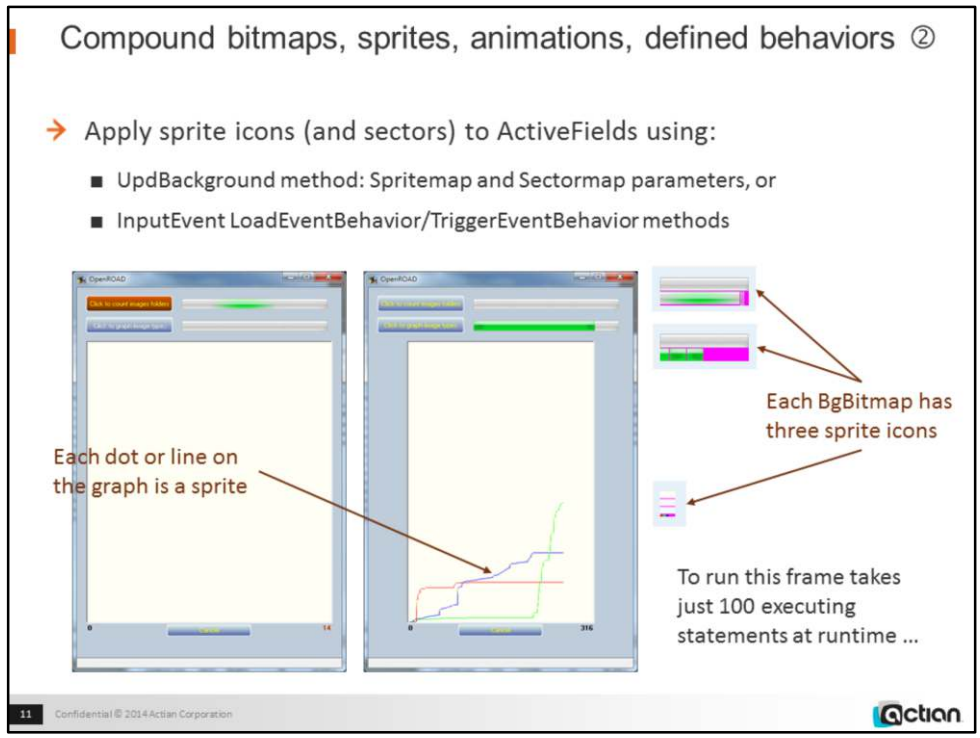
Click the Go button

Note that each field now has a different imageindex between 1 and 10; that is the only thing that has changed.

```
/*
**      Display the buttonstyles
*/
for i = 1 to styler.ChildFields.LastRow do
        btn = styler.ChildFields[i];
        /*
        **      Change the imageindex
        */
        btn.UpdBackground(imageindex=i);
endfor;
```

Borders can be range of styles (plain, concentric, 3-D, adhoc, none); plain borders can be any width; concentric borders can have up to 3 layers.

Key properties are: UpdBackground method; BgBitmap, BgDisplayPolicy and BgPattern attributes.



Demo:

D201504_DefinedResponses_Sprites -cProgressBars_coded

Click the first button; when second button activates, click it

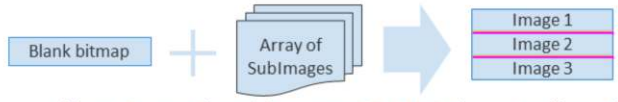
```
/*
** List and count and graph the imagefiles in each of the listed folders
*/
for i = i + 1 to folders.LastRow do
        foldername = folders[i].Value;
        /*
        ** Execute a Windows command to list the image files; read in and parse the list file;
        ** Add the count of found bitmaps to the total bitmaps of this type;
        */
        call system :cmd;          //creates list in file <fname> of bitmaps in the folder <foldername>
        filestring.FileHandle = fname;
        fromct = ct;
        ct = ct + filestring.Split(delimiter=HC_NEWLINE).LastRow;
        /*
        ** Define a sprite as a line between the old value and the new; display the sprite;
        ** Tell the progress bar how far (%) we have got
        */
        height = (ct-fromct)/scale + 1;
        SDS[j].SetAttribute(spritesourceindex=j, x=i, y=graphh - fromct/scale - height,
                height=height, width=width);
        SDS[1].ApplySpriteMap(targetfield=field(graph), descriptors=SDS, operation='add');

        step = (i*100)/folders.LastRow;
```

```
        IE.TriggerEventBehavior(location=progbar,
                eventkey=progtriggerkey + '#' + varchar(step) + ',' + varchar(step));
endfor;
```

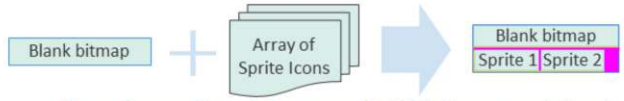
More about Compound Bitmaps

- Created using ComposeBitmap (or a bitmap editor …)
  - To create a bitmap containing sub-images:



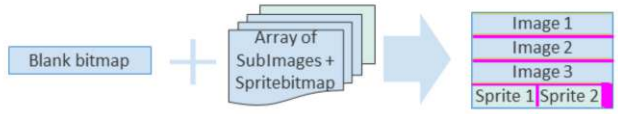
bitmap.ComposeBitmap(elementtype=BE_IMAGE, elements=subimages);

  - To create a bitmap containing sprite icons:

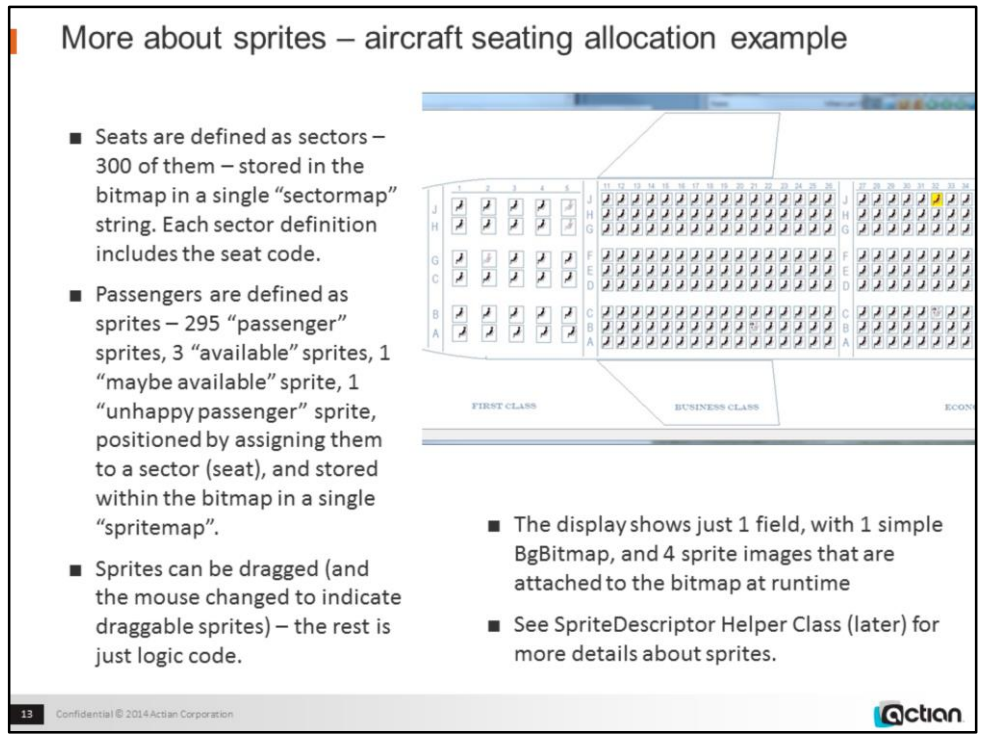


bitmap.ComposeBitmap(elementtype=BE_ICON, elements=spriteicons);

  - To create a bitmap containing images and sprites:



bitmap.ComposeBitmap(elementtype=BE_IMAGE, elements=subimages, hasicons=TRUE);



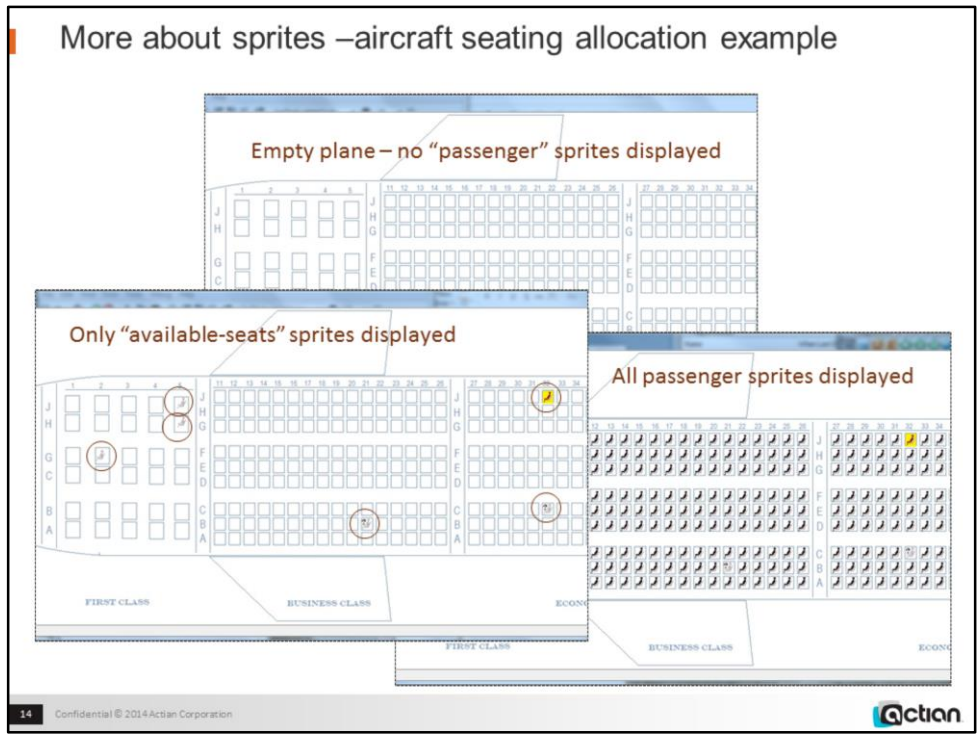

```
…
/*
**  Map the Economy Class seats (as sectors)
*/
for row = 27 to 44 do
        for aisle = 1 to 11 do
                …
                descriptor = descriptors[indx];
                descriptor.SetAttribute( spritesourceindex=1, name=name, width = 20,
height = 22,
                        sector=1, gravity='CC',  x=x, y=y);
        endfor;
endfor;

SD.ApplySectorMap(targetfield=field(airplane), descriptors=descriptors,
operation='apply');

/*
**  Put passengers on the seats (as sprites)
*/
for I = 1 to passengers.LastRow do
```

```
                    …
                    descriptor = descriptors[indx];
                    descriptor.SetAttribute( spritesourceindex=indx, name=seatname,
flags=responseflags,
                            sector=seatsector, gravity='CC',  x=0, y=0);
            endfor;
        endfor;

        SD.ApplySpriteMap(targetfield=field(airplane), descriptors=descriptors,
operation='apply');
```



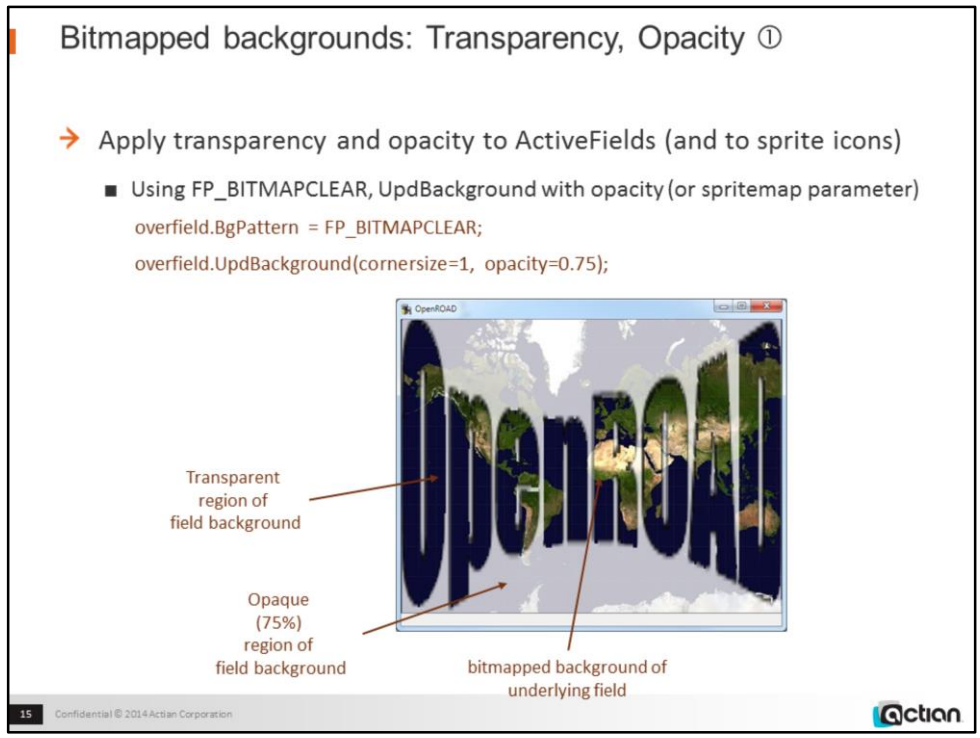
Demo:

w4gldev runimage workbnch.img -Tall -/appflags profile=or62demos application=allocationsystems component=airlineseating command=openscript#561

Note that the code is creating a spritedescriptor definition for each Economy seat, treating it as a sector, computing each seat's size and x & y. Then the code creates a sectormap from the descriptors, and stores it in the "seating plan" bitmap.

Go to line 265

Note that whenever a passenger is dragged to a new seat, the code just calls the LastInputAction method, once to identify the passenger (action='mousedrag_down') and once to identify the seat (action='mousedrag_up').



Demo:

D201504_BitmappedBkgFields -cBitmappedBkg_Opacity_Transparenc

Run the frame

Note that the frame background displays a satellite image of the world, as a Mercator projection

Rightclick the frame

Note that the world image is overlaid with a field that has transparent areas (forming the letters of the word "OpenROAD"), and translucent areas (75% opacity).

Demo:

w4gldev runimage workbnch.img -Tall -/appflags profile=or62demos application=d201504_bitmappedbkgfields component=bitmappedbkg_opacity_transparenc command=openscript

Note that the frame background displays a satellite image of the world.

Note that there is a buttonfield overlaying the background, but that field initially is FP_CLEAR and has no text, so you cannot see it.

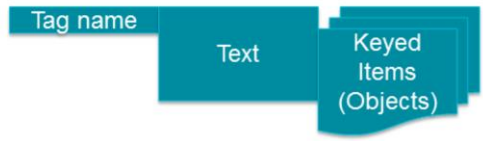
Rightclick the frame

Note that the code simply makes the upper field's background transparent (FP_BITMAPCLEAR) and translucent (opacity=0.75, set using the UpdBackground method)

```
/*
**  Make the overlying field transparent and 75% opaque
**  (overfield is a buttonfield with an image of the word “OpenROAD”.
Underneath is the frame’s
**  topform, displaying a satellite image of the world)
*/
overfield.BgPattern = FP_BITMAPCLEAR;
overfield.UpdBackground(cornersize=1, opacity=0.75);
```

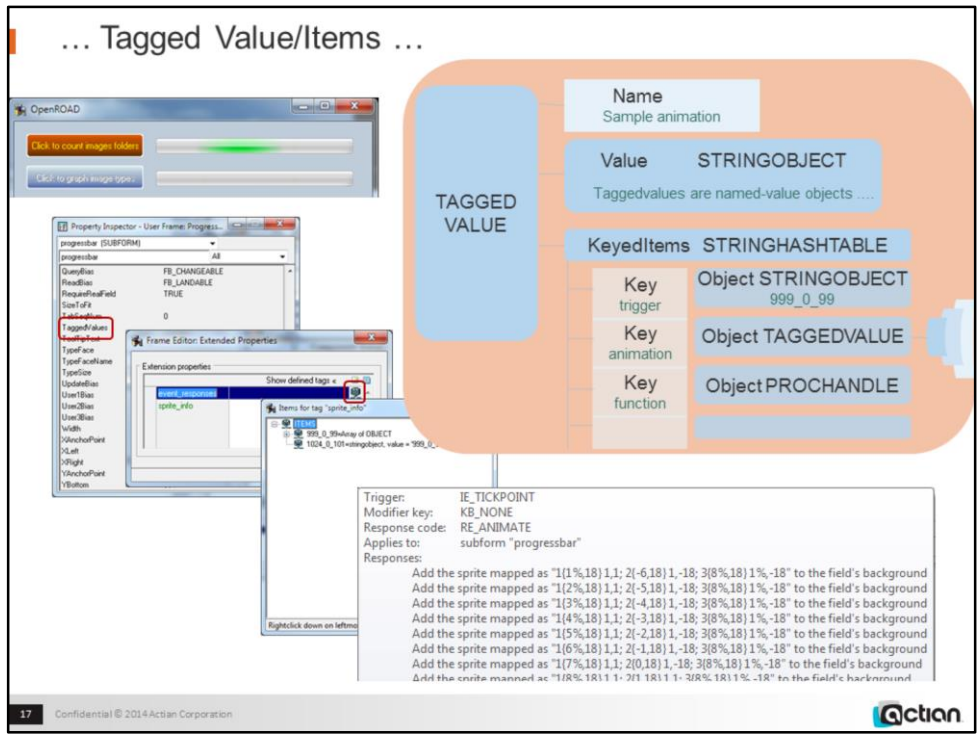
❑ Tagged Value/Items ③

- Need to make data and objects more systematically accessible
  - Tagged Values/Items
- Taggedvalues are named-value objects and collections.



- Every field has them, every frame, class, application, attribute.
- They allow you to store anything you want, where you need it, always available, with immediate keyed access:
  - Values, text, search-markers, lookuplists, procedures, resources, packages, objects of any type
- When the frame or class is saved, taggedvalue contents save as well.

except userclass objects



Demo:

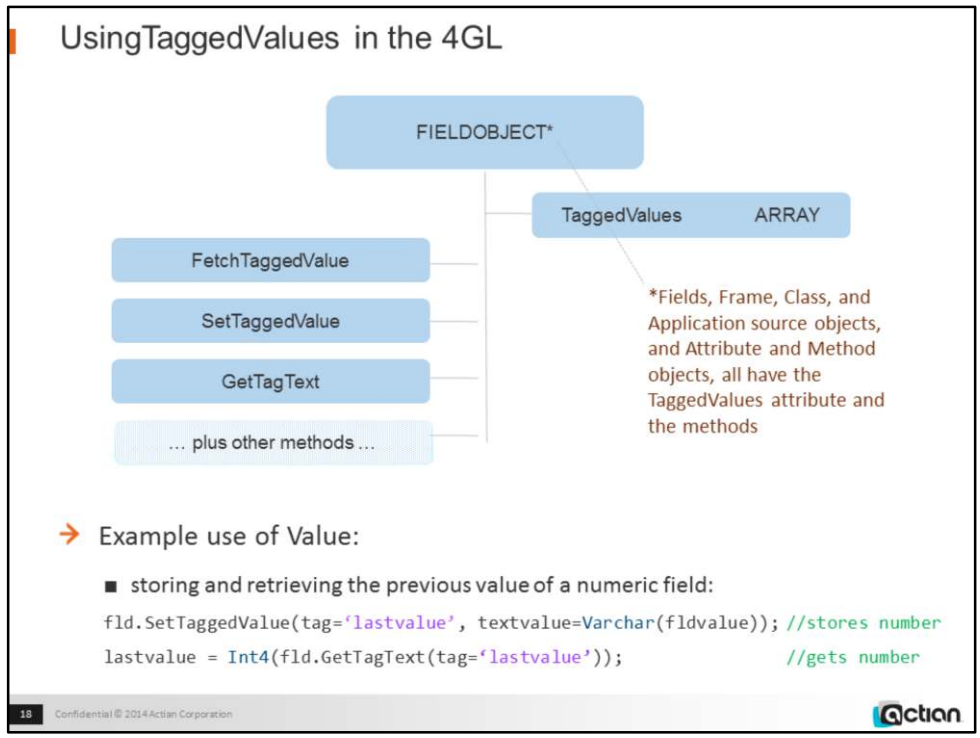
w4gldev runimage workbnch.img -Tall -/appflags profile=or62demos application=d201504_definedresponses component=progressbars_predefined command=open

Continue as shown.

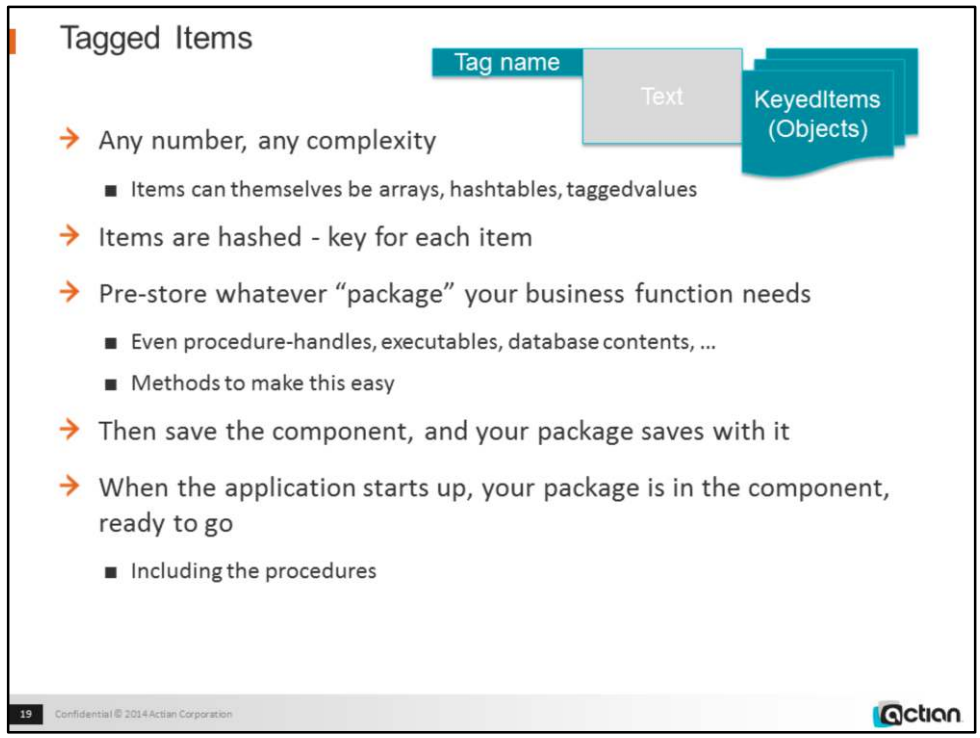
In this example, the tooltiptext in the TaggedValue Editor Items Dialog is displaying a description of the stored defined behavior that drives the marquee bar.

- the marquee behavior is actually a sprite-based animation that moves the green pulse to and fro
- The description is derived by examining and interpreting the KeyedItems that make up the behavior definition

The behavior is actually stored in the marquee field, as a TaggedValue object called (“event_responses”), containing a collection of Items defining the required behavior. No 4GL code or event is required for the animation.




```
/*
**  Store the field's value (numeric in this example);
**  Retrieve the stored value;
**  Retrieve the entire taggedvalue
*/
fld.SetTaggedValue(tag='lastvalue', textvalue=Varchar(fldvalue));        //stores the
number
lastvalue = Int4(fld.GetTagText(tag='lastvalue'));                       //gets the
number
lastvalue_tag = fld.FetchTaggedValue(tag='lastvalue');                   //gets the
taggedvalue itself
```



"Procedure-handles": ProcHandles

ProcHandles have a huge advantage over Call Procedure as a way of invoking frame or field or userclass procedures:

- Call procedure myproc (or call procedure :myprocname) only works if the calling code can see the procedure-declaration – in practice this limits callable local procedures to those declared in the same script

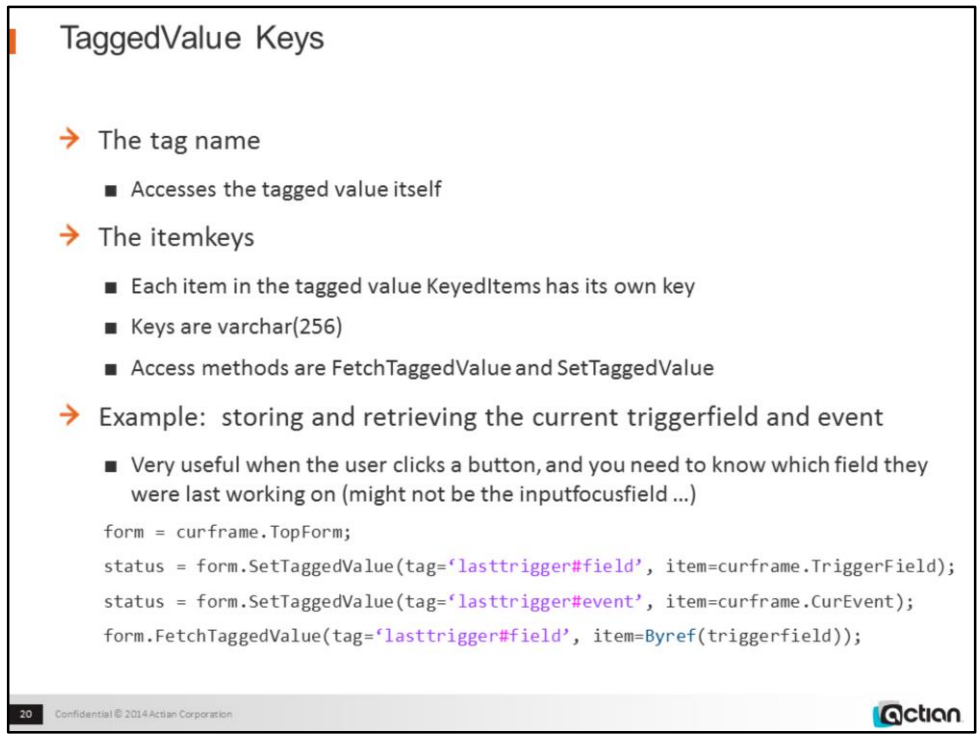
```
call procedure myproc(…parameters…);          //works only if procedure declaration is visible
```

- The ProcHandle for myproc incorporates the declaration, so it works from outside the frame or userclass – under the right circumstances it can even be saved and restored, or exported and reimported, and it will still work

```
/*
** Create ProcHandle for this procedure
*/
myprochandle = myUserclass.GetProcHandle(name='myproc');

/*
** Execute procedure from different frame, method, procedure
*/
myprochandle.Call(…parameters…);          //works in much wider range of
```

circumstances




```
/*
** Store the last triggerfield and last event in the frame's topform
** ...
** Get the last triggerfield (later on, when we need it)
*/
form = curframe.TopForm;
status = form.SetTaggedValue(tag='lasttrigger#field', item=curframe.TriggerField);
status = form.SetTaggedValue(tag='lasttrigger#event', item=curframe.CurEvent);
...
form.FetchTaggedValue(tag='lasttrigger#field', item=Byref(triggerfield));
```

Availability and formatting

- TaggedValues can be
  - Runtime, designtime, or both
  - Temporary or permanent
  - Formally defined or ad-hoc

Use the Availability property for both of these

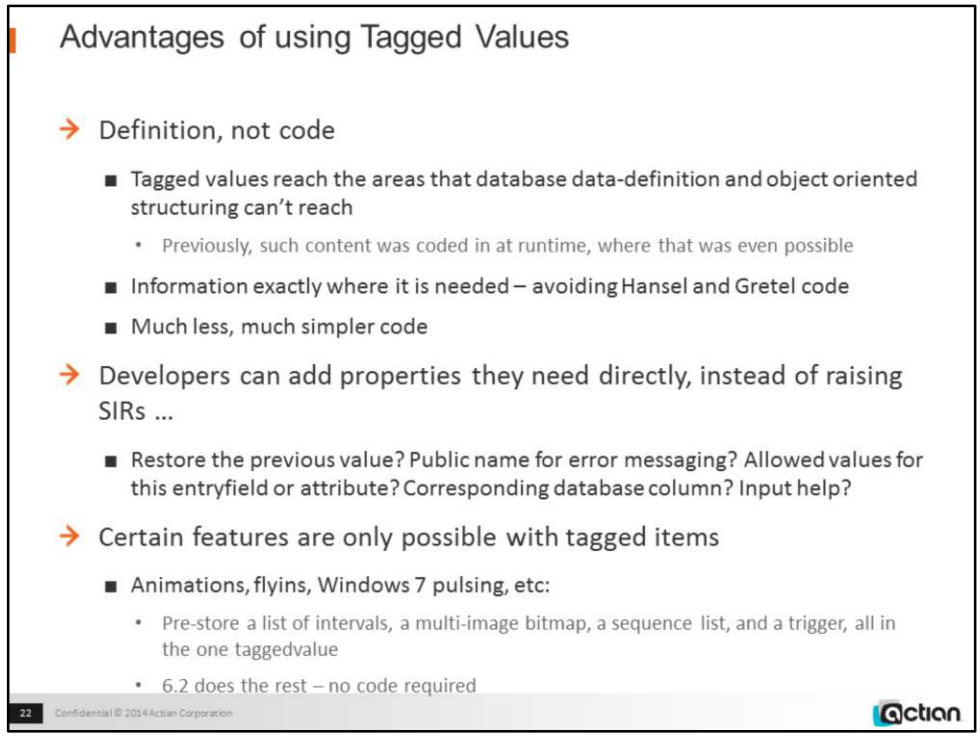
Use TagDefinitions for formal tags

- When you are preparing defined behaviors, you don't want your mistakes saved! So the "temporary" setting is very useful

```
tagged_value.Availability = TVA_TEMPORARY;
```

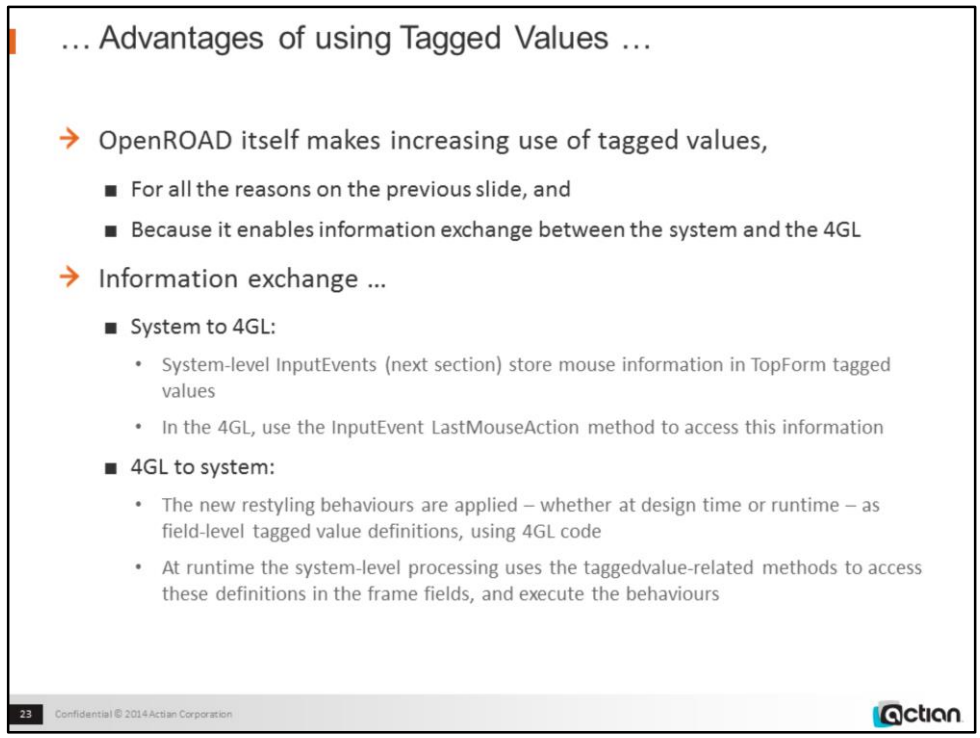
  - But you do not need to use it directly: use the InputEvent LoadEventBehavior, which includes a MakePermanent parameter.
- The TaggedValue Editor uses formally defined tags
  - to ensure that the important tags for classes, attributes and fields are listed, whether or not they have yet been set
  - The Files subdirectory contains TagDefinition files to support this
  - You can define your own – details are in the Language Reference Guide

Actian



Examples of Hansel and Gretel code / Adhoc trails:

- passing an incrementing counter to a userevent as the MessageInteger
- including the datatype of a variable in the variable's name



4GL to System – executing predefined behaviors:
With the new sprites and InputEvent/Response processing, field and frame appearance can be much richer, and match chosen styles

- There is system support for the underlying generic mechanisms,
- but the actual behaviour of a given style has to be customized, and that means 4GL

So, we (and you) use 4GL to predefine the style behaviour as TaggedValues, and store them

- See Setup Frame capability

Now OpenROAD runtime can see the stored behaviors in the tagged values when the frame starts up, and do the work,

- Without any need for 4GL code or events

More about this in the next two sections

❑ Storable Defined (ready-to-use) Behaviors ④

→ OpenROAD 6.2 introduces a major paradigm shift (see ①):

- 4GL Event granularity is **whole field**
- Field shapes are rectangular
- Field "movement" is whole field, and is limited by widget overhead

- Event granularity is **icon** (sprite)
- Field shapes are complex
- Field "movement" includes sprite click, drag and hover, timed sequences, multiple independent elements

- The paradigm shift makes it cost-effective to provide all these needed flexibilities in OpenROAD ...
- ...provided we can work at a finer response-granularity than the 4GL events offer

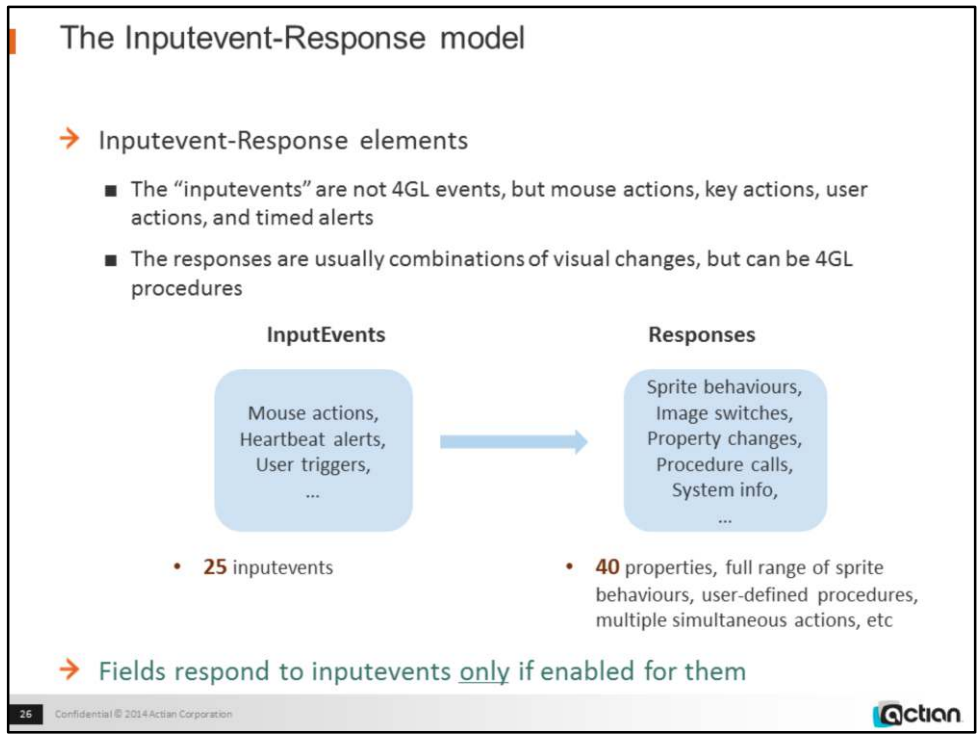
… Storable Defined (ready-to-use) Behaviors …

→ The solution to the granularity requirement involves a further paradigm shift: inputevents and defined behaviors

- 4GL-display-event triggers
- Userevent alerts: CPU-intensive and queued
- Range of 4GL events is restricted, because:
  - Mousemove etc too costly
  - 4GL event model is at optimum
  - Backward compatibility needed

- Mouse-event triggers
- Heartbeat alerts: efficient and synchronous
- Response-granularity is unrestricted, so:
  - Sub-field needs can be met
  - Can respond to more user actions
  - Still backwardly compatible
- Can still invoke 4GL for logic when needed

- Inputevent triggers are finer granularity than 4GL event triggers
- No 4GL events or 4GL code are involved (except where you actually want them)



Two ways to enable a field for inputevents:

```
/*
** Create an "inputevent_enabled" taggedvalue in the field
** (for use if and only if the behaviour does not involve a BgBitmap)
*/
fld.SetTaggedValue(tag='inputevent_enabled');
/*
** Apply the InputEvent ActivateFields method to the field
** (ensures each listed field has a BgBitmap and a suitable BgDisplayPolicy)
*/
IE.ActivateFields(fields=fields, bitmap=bitmap);
```

InputEvents:

IE_KEYDOWN
IE_KEYUP
IE_SYSKEYDOWN
IE_SYSKEYUP
IE_MOUSEMOVE
IE_LMOUSEDOWN
IE_LMOUSEUP
IE_LMOUSEDBLCLK
IE_RMOUSEDBLCLK
IE_MMOUSEDOWN
IE_MMOUSEUP
IE_MMOUSEDBLCLK
IE_NCMOUSEHOVER
IE_MOUSEHOVER
IE_SETFOCUS
IE_LOSEFOCUS
IE_MOUSEENTER
IE_INIT //Initialization (not initialize) event
IE_MOVEPOINT //The most recent move event at this timepoint
IE_PULSE //Pulse-alert event (heartbeat alert, every 1 second)
IE_TICKPOINT //Registered heartbeat-alert event
IE_USER //User-defined action (IE_USER+1, IE_USER+2, etc, are also available

IE_RMOUSEDOWN
IE_RMOUSEUP

How are behaviors defined, and how are they stored?

- Each behavior is defined using
  - An EventKey string, combining:
    - The mouse, key, alert, or user action (as an IE_ constant)
    - The modifier key(s), if any (as KB_ constants)
    - The response code (as an RE_ constant)
  - A Response object, providing the essential element(s) of the particular response:
    - StringObject, for an imageindex or spritemap or nested behavior
    - Field, for a display behavior
    - ProcHandle, for a 4GL-coded behavior
    - Cursor bitmap, for cursor-change behaviors
    - TaggedValue, for timed/animation behaviors
    - Arrays, for sets of simultaneous behaviors
  - The InputEvent Helper Class has methods that make all of this easy to specify

… How are behaviors defined, and how are they stored? …

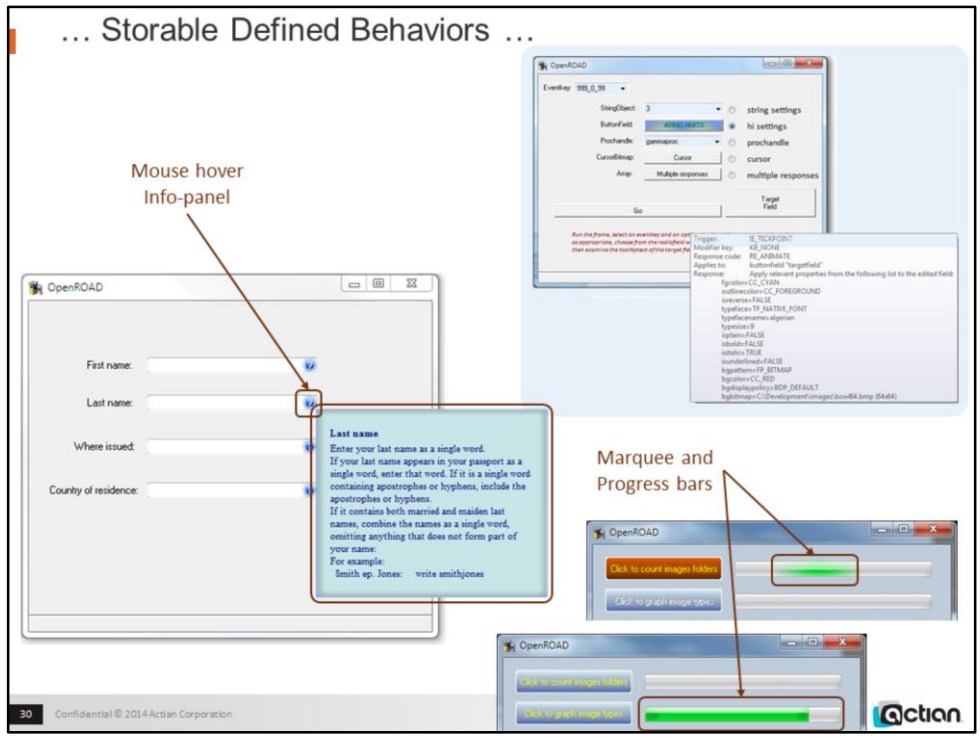
- All defined behaviors are stored in a field tagged value
  - Using the tag name "event_responses"
  - The tagged value may belong to a display field (for behaviors specific to that field), or to the frame's TopForm (for responses common to many fields)
    - For example, the Windows7 styling behaviors are defined at TopForm level
- Each behavior is stored as one of the tagged value's KeyedItems
  - Using the Eventkey value as the lookup key
  - Example – the Windows7 default mouseover highlighting:
    - Two stored behaviors, one for the mouseentry, one for the mouseexit
    - Each applies a different imageindex to the background bitmap
      - so that a different background displays when the mouse is over the field
- Behaviors can be stored permanently or runtime-only
  - Permanently: in the frame's framesource fields,
  - Runtime-only: in the frameexec fields, or with availability=TVA_TEMPORARY

Actian

… How are behaviors defined, and how are they stored? – Example (Windows7 highlighting)

- To highlight on mouseenter, we need:
  - Eventkey: IE_MOUSEENTER, KB_NONE, RE_OTHERBUTTON
    - 995_0_89
  - Response: StringObject set to the highlight imageindex
    - "2"
  - Behaviour stored in the frame's topform
- We use InputEvent (Helper Class) methods to store the behavior:

```
eventkey = IE.EventKey(action=IE_MOUSEENTER, modifierkey=KB_NONE,
                responsecode=RE_OTHERBUTTON);
response = IE.Response(type='image', imageindex=2);
IE.LoadEventBehavior(location=frame.TopForm, eventkey=eventkey,
                    responsetype='event_responses', response=response);
```



Demo:

D201504_DefinedResponses_Panel –cPassportDetails

Run the frame

Hover the mouse over any of the "?" icons

An infopanel describing that field will appear

Note that no runtime code is involved in each popup response

Demo:

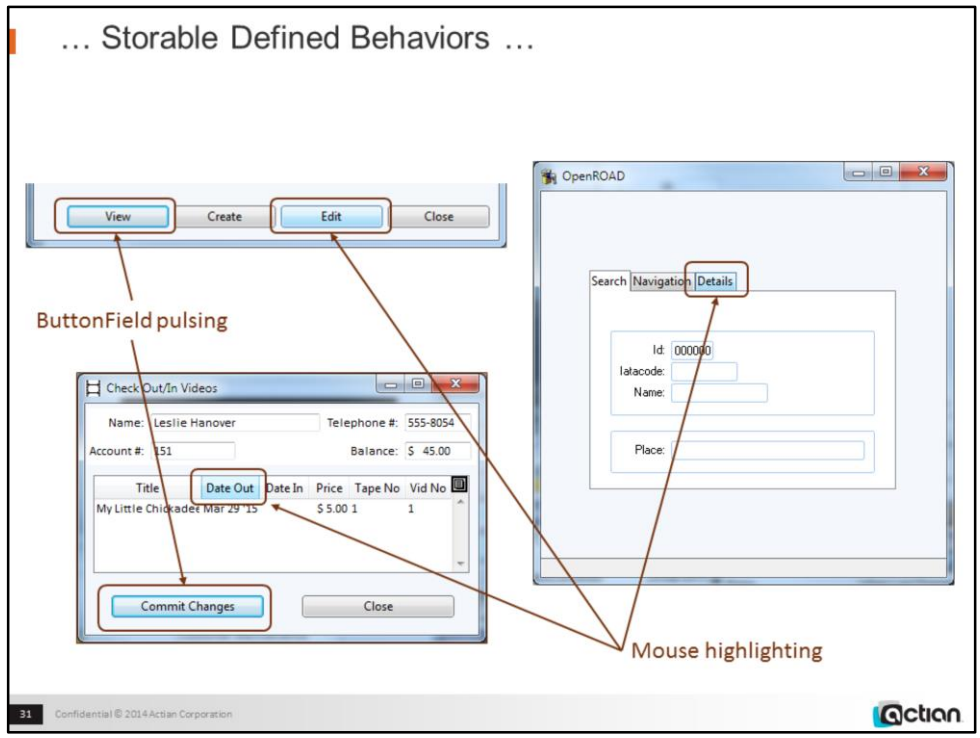
D201504_DefinedResponses –cDecodeDefinedBehavior

Run the frame

Continue as instructed (instructions on frame)

Note that each resultant tooltiptext identifies what combination of mouse or timer action and modifier key will produce what response, based on the selections that were made.

Note that you can have multiple simultaneous responses to a single action.



Demo:
D201504_VideosConverted
Run application
Choose Check Out option
Enter 151 as customer account
Ctrl-Shift-Tab to move focus to “Commit Changes”
Hover mouse over Date Out column header

Demo:
D201504_BitmappedTabfolderTabs -cBDPTabHighlighing
Mouse vertically over an unselected tab
- The tab will highlight

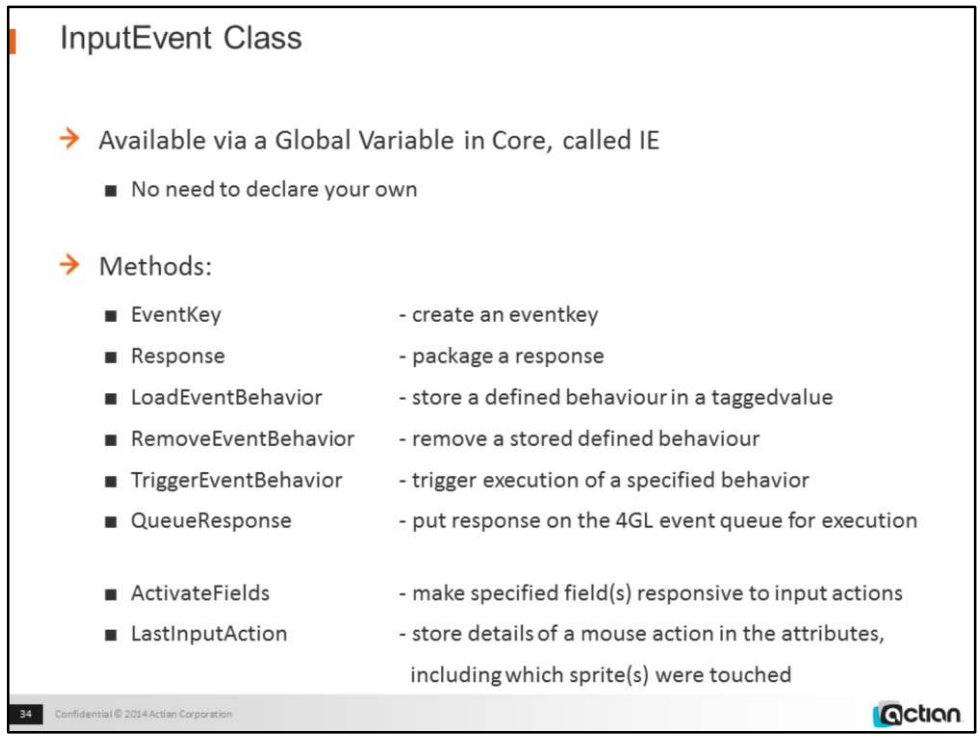
Storable Defined Behaviors in Restyling (see Videos demo):
- Restyling is applied to ButtonFields, EntryFields, TableField headers, TabFolder tabs, SubForms, other compositefields, FreeTrims, Mainbars, RectangleShapes, ControlButtons.
- Most other fields are already W7 style, since we used native widgets for them.
- Field fonts are changed to Segoe UI 9

… Storable Defined Behaviors – summary of features

- Defined as tagged value items
- Created and stored using Helper Userclass methods
- Predefined using Setup Frames
- Executed (for responsive fields):
  - Automatically in response to WindowManager events
  - Automatically in response to Hearbeat-scheduled alerts
  - On demand in response to TriggerEventBehavior 4GL calls
- Responding with single, multiple or time-based responses
  - Image-switch, sprite-display, display-properties changed, 4GL processing, combination and animation effects
  - Any degree of complexity
- Essential for OpenROAD 6.2 Windows7 restyling

❑ Helper Classes ⑤

- One more paradigm shift …
- New Helper UserClasses, added to the Core application (so always available):
  - InputEvent - provides all the methods to define, schedule, trigger, manipulate or stop the range of defined behaviours
  - SpriteDescriptor – provides all the methods to handle field-background sprites
  - RequestManager – provides frame-management support for displays generated in an active_display frametemplate
- All processing using defined behaviors, sprites, and generated displays should make maximum use of these classes, as
  - They greatly simplify the handling of these features
  - They are used throughout the example code provided with OpenROAD 6.2
  - They are detailed in the documentation
  - They can be inherited and customized in helper classes you create

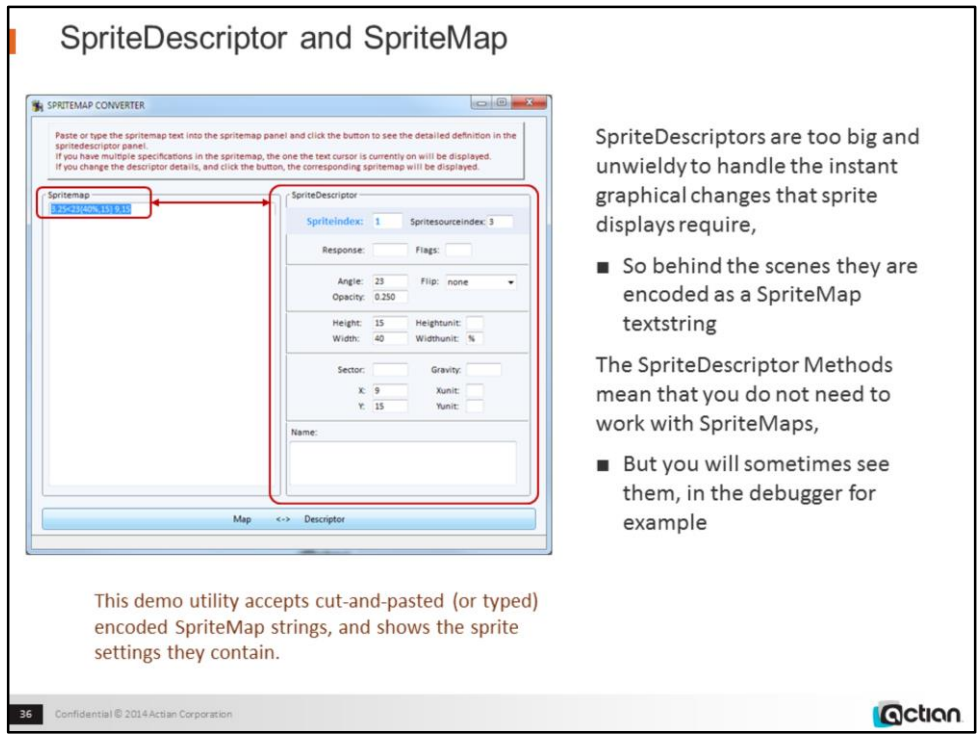


QueueResponse Method:
Frames can only execute properly if called from 4GL code triggered (directly or indirectly) by an OpenROAD event that has been handled by the 4GL Event Queue. (This is why OpenROAD Server applications cannot handle frame calls).
InputEvent responses bypass the 4GL Event Queue, so if you need your ProcHandle (4GL procedure) response to call a frame, for example an info-popup, you need a way for it to queue its processing. The QueueResponse Method provides that.

SpriteDescriptor Class

- Attributes
  - These define a sprite or sector's position, size, identity, and (sprite-only) source-image, opacity, orientation, behaviour
    - Sprite built-in behaviours: drag, mouseover highlight, cursor-change
- Methods:
  - ApplySectorMap - apply a sectormap* to the field background
  - ApplySpriteMap - apply a spritemap* to the field background
  - SpriteImage - return the source and actually-displayed images
  - SpriteKey - eventkey for sprite-specific behaviors
  - SpriteMap - return the encoded spritemap as a StringObject
  - WhichSector - find by location or index and return a sector
  - WhichSprite - find by location or index and return a sprite

* These two methods each accept an array of SpriteDescriptor, and internally convert it to a sectormap/spritemap which they apply to the field background



Demo:

D201504_SpritemapConverter –cSpritemapConverter

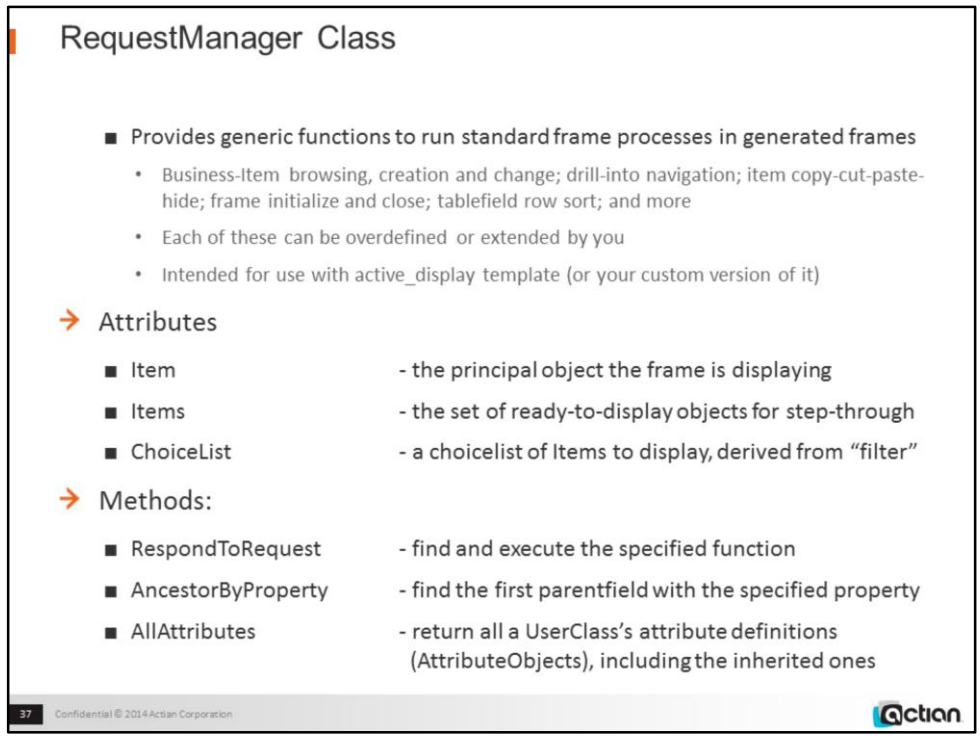
Click the button

- the attribute fields on the right fill with values parsed from the example definition string on the left

Amend any of the SpriteDescriptor attribute field values

Click the button

- the spritemap on the left changes to reflect the new spritedescriptor settings



RequestManager, Active_Display, and RespondToRequest

RequestManager is for use with frames created from the active_display FrameTemplate
although you can take advantage of its features for other purposes

Active_display frames treat each enduser action as a request for a particular response
For example: Clicking the Save Button is a request for a Business-Item-Save response

Each active_display frame contains just 60 lines of code (which you can add to or delete)
the RequestManager does all the 4GL work, via the **RespondToRequest** method.

RequestManager holds all the generic functions for these frames
Each of these can be overdefined in the frame by a local procedure.

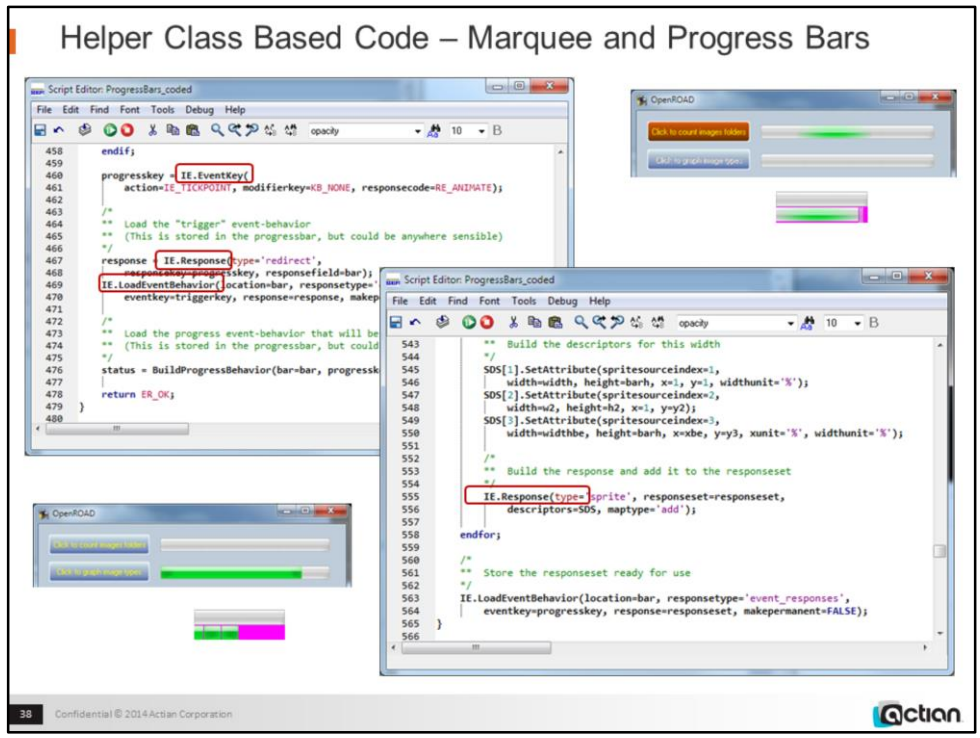
Overdefining and extending the functions:

In RequestManager, each function is a "case" within one of the following local procedures:
FrameRequest, DataRequest, TblRequest, OtherRequest.

Each procedure has the same interface (action=varchar, trigger=fieldobject, info=object).

To override the Close function (which belongs to FrameRequest), for example:

- Create a FrameRequest local procedure in your frame, with the standard interface.
- In it put a **case action** statement, with a case of 'close':
- In the close case, put or call the processing you want executed instead of the RequestManager default.



w4gldev runimage workbnch.img -Tall -/appflags profile=or62demos application=d201504_definedresponses_sprites component=progressbars_coded command=openscript#520

Note the way the EventKey and Response and LoadEventBehavior methods combine to create and store a defined behavior

Go to line 603

Note that the code samples here are both **setup** code extracts, not needed at runtime

They use the InputEvent and SpriteDescriptor Helper Class methods.

Note that the “Compound bitmaps, sprites, animations, defined behaviors” slide shown earlier has an extract of **runtime** code,

also using the InputEvent and SpriteDescriptor Helper Class methods.

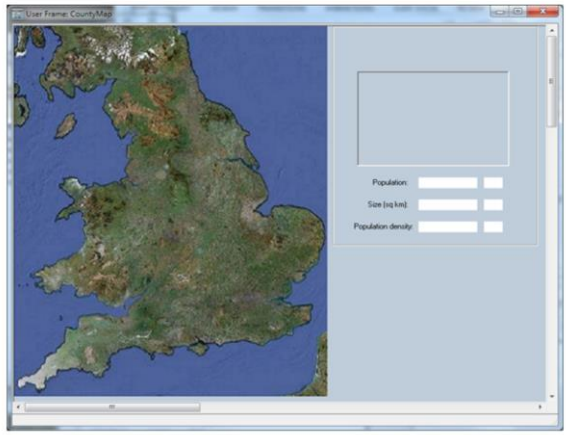
❑ Setup Frames ⑤

- Existing applications have a lot of "setup" code in each frame
  - Code that should have been run beforehand, if the results could be saved
  - Not just in the initialize block either!
- The Property Inspector now has as SetupName option
  - Specify the setup frame's name, and it is called from the frame editor, and passed the target frame's framesource (as the "frame" parameter)
  - In the setup frame put all the setup changes that need to be preapplied to the framesource, for example:
    - Field property changes unavailable in, or too laborious to apply from, the StyleSheet or PropertyInspector (bias settings, tabfolder settings, tablefield settings, ...)
    - Computation of ready-to-use data from fixed-data sources (lists, trees, decodes ...)
    - Changes to the target frame's code (maybe ...)
    - (In 6.2) Storage of defined behaviors, InputEvent-activation of fields, ...
  - When the setup has run, and the target frame is saved, the setup is in place – and the target frame's old setup code is no longer needed

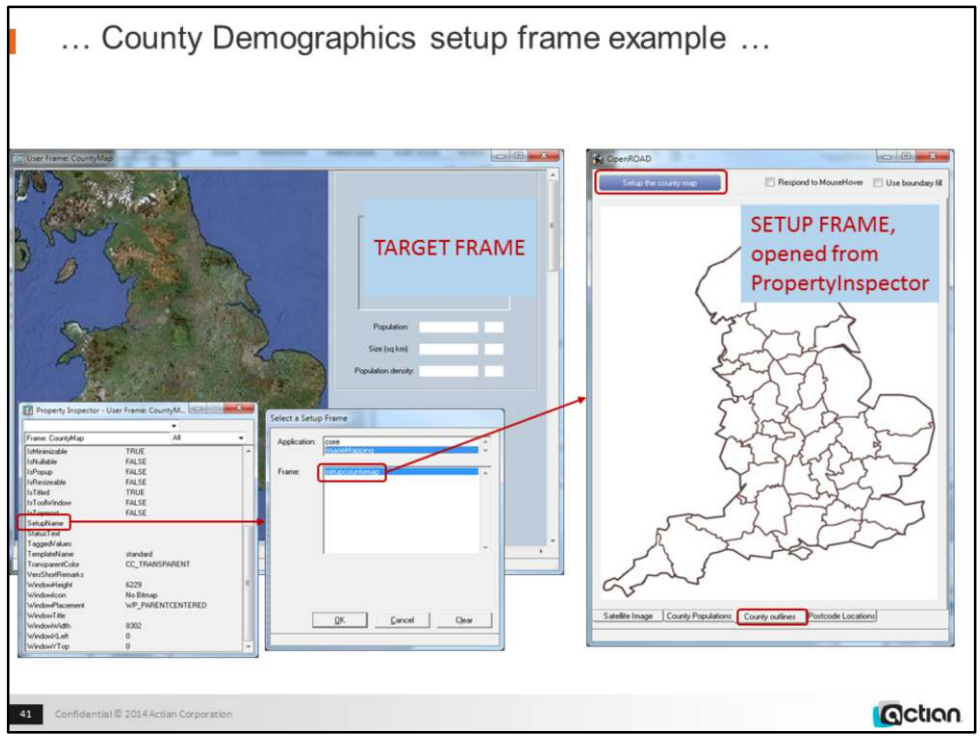
… Setup Frames …

County Demographics setup frame example

→ Initial state



- frame contains:
  - Placeholder (JPG) satellite image
  - Runtime code and fields
- frame also requires:
  - County demographic data
  - County boundary coordinates
  - Mask image, each county uniquely coloured
  - Map-ready (BMP) satellite image



Demo:

w4gldev runimage workbench.img –Tall -/appflags profile=or62demos application=D201504_ImageMapping component=countymap command=open

The CountyMap frame is opened for edit

Select the SetupName entry in the Property Inspector

The Setup Frame dialog will appear

Select the "D201504_ImageMapping" application and the "setupcountymap" frame,

and click OK

The SetupCountyMap setup frame will run

Click the "County Outlines" tab

An outline map of English counties will appear

Click the "Setup the county map" button

After a few seconds each county will be coloured a different shade of grey

Note that the CountyMap frame, the one that the enduser will see at runtime, has been setup and ready to go:

- the grey (mask) image, the county boundary coordinates, and the cross-reference of these to the county demographic data, have all been generated by the setup frame, and applied to the CountyMap frame

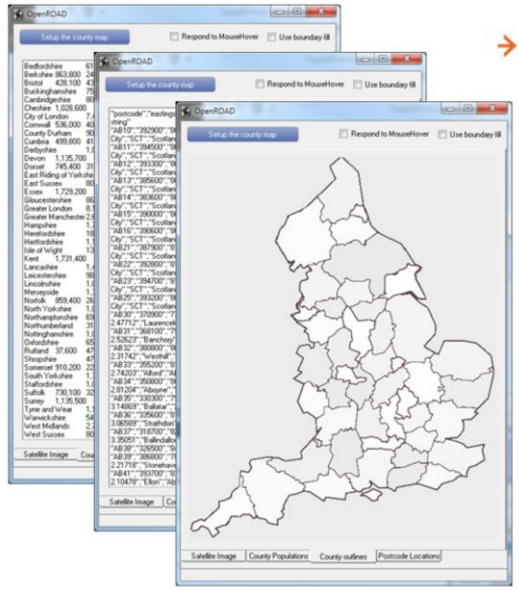
Close the setup frame

Run the county map frame

Click any point in SouthEast England on the satellite image to confirm that setup has worked correctly

- the county under the mouse is outlined in green, the name and demographic data for that county appear on the right, and a satellite image of that county appears above the data

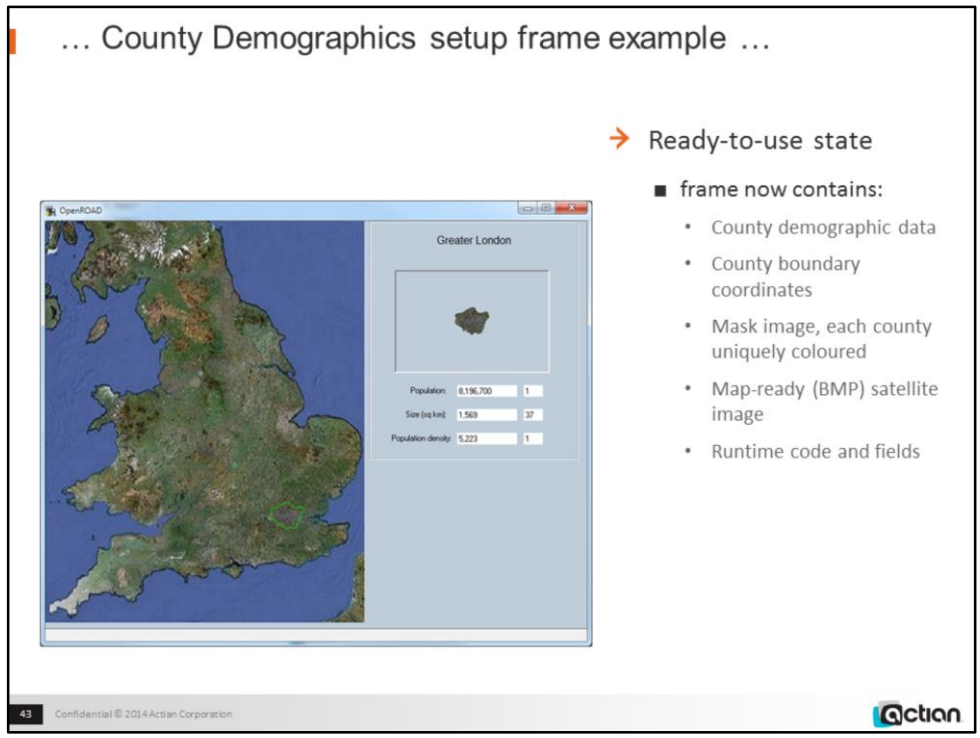
Running the Mapping setup-frame



→ What does this frame do?

- Parses UK postcode location data
  - Data is publicly available
- Picks a postcode for each county
- Maps each county boundary and uniquely colours each county, creating a "mask"
  - Using FillBitmap
- Parses the county population data and crosslinks it by colour
- Stores the ready-to-use mapped data in the target frame
  - When the target frame is saved, the mapped data will be saved
- Job done.

Around 300 executing statements in the setup frame 4GL, leaving just 70 in the runtime frame.



Demo (continued):
Run the CountyMap frame
Click somewhere in SE England.
(Note – the source data was missing some counties; clicking on those gives incorrect results)

How does it work? Simply and generically:
- The colour of the mask at the mouse location identifies the county
- That county's name and data is displayed
- That county's border coordinates are used to draw the outline
- A rectangle including the county is extracted as an image
- FillBitmap fills everywhere outside of the border with the border colour
- The image is displayed treating the border color as transparent.

✓ Providing generic enabling facilities to underpin the business-need features, both current and future

1. Bitmapped backgrounds with built-in bordering and double-buffering
2. Compound bitmaps, sprites and animations
3. Tagged Values/Items
4. Storable defined-behaviours
5. Helper classes and Setup frames
6. Many enabling property and method changes to field and data classes
   - TreeViews, TableFields, TabFolders; String & BitmapObjects; and much more
7. Enhanced PropertyChanger facilities
8. Database and display heuristics
9. Downloadable IngresNet
10. LoadnRun deployment

Items 6 to 8 are covered in the fourth presentation

Items 9 and 10 were covered in the second presentation

Actian

What was covered

- OpenROAD 6.2 – New generic features in more detail, Part I
  - The third of four presentations covering OpenROAD 6.2
    - This presentation was the first of two reviewing in detail the new features in this release
  - Features illustrated in the presentation require the first OpenROAD 6.2 patch
    - p14746 or later

Actian

Thank you

- Sean Thrower, OpenROAD Engineering
  sean.thrower@actian.com