DataFlex Mobile Web

Presented by: John Tuohy
Mobile/Touch Web Applications

- Mobile/Touch web applications support a new style of application

- In this presentation we will:
  - Compare Mobile/Touch and Desktop style applications
  - Explain why a new application style was needed
  - Introduce you to the new style’s architecture
  - Show how you use this to build Mobile/Touch applications
How is the Mobile/Touch Environment Different?

- **Display**
  - They tend to be smaller screens
  - There are many devices with all kinds of different sizes screens
  - Screen sizes change on single device - portrait and landscape
  - High resolution - let's you show things very small - good for reading, bad for touching
  - Everything tends to be run full screen

- **Pointing device**
  - Your finger is not a mouse or a mouse substitute
  - The finger is a rather imprecise pointing device
  - Finger target space is completely different than a mouse target space
  - Scrolling is completely different
  - There is no right click
  - There is no double click

- **Keyboard**
  - The on-screen keyboard uses up valuable screen real estate
  - There are limited keys - no function keys, no ctrl/alt keys
  - In general, they are hard to use
How is a Mobile/Touch User Different?

- Is used to the forward/back browser stack paradigm
  - Understands a stack of operations (often seen as a breadcrumb)
- Is more adaptable
  - Willing to experiment
  - Doesn't want a lot of explanations
  - Accepts and expects hidden interfaces
  - Seems more accepting about not understanding something right away
- Expects an application to flow. The application will guide them
- Does not want a lot of confirmations - just do what's right
  - Does not want warnings about doing the right thing
  - Might want warnings when doing the wrong thing
- They expect applications to look great and “modern”
  - Expect a “a less is more approach”
  - In the battle of form over function – form wins
- They expect what we call a “webby” interface
  - They might even expect this same interface on a desktop browser, even if this is not optimal
Consequences of these differences

- Applications don’t use windowing
  - Just about everything is full “screen”
- Application flow is different
  - You navigate forward, back and go home
- Applications are less user driven and more developer driven
- Traditional menu systems and tool bar systems don't work well
  - When used, they are much smaller and much simpler
- Fewer Keyboard and mouse shortcuts
  - No context menus
  - No right click
  - No double click
- Vertical scrolling is common, horizontal much less so
- Keyboard usage is kept to a minimum
- Modal dialogs are kept to a minimum
The Desktop Framework and Mobile/Touch

- The desktop style framework style was originally created to accommodate evolving computers which had
  - Big screens
  - Flexible and precise mouse pointers
  - Full functioning keyboards
- The desktop style may not be a good fit for mobile/touch devices
  - Whole basis of the desktop framework is independent, selectable, overlapping views
  - The desktop framework makes extensive use of modal prompt lists
  - The desktop style is flat, not deep
  - The desktop style is completely user driven
  - You can't just create DDOs, create prompt lists, create views, add them to a menu and be ready to go
A New Application Style for Mobile/Touch

- We decided we needed a new style of application that
  - Uses a drilldown style
  - Is more application driven, less user driven
  - Requires the developer to connect the pieces

- Would this be a new framework?
  - Would it just be better to build a whole new drilldown framework?
  - We didn’t know at first
  - This became a real test of the adaptability of the framework
What we did

◦ We built Mobile/Touch style as an extension

◦ The DataFlex framework survives with a new application style
  • The DataDictionary classes and your DDOs require no changes
  • The basis of the framework is unchanged - you still create views, which contain a DDO structures and connected DEOs
  • We extended the cWebView class
  • We extended the web DEO classes

◦ Now the web framework supports a drilldown style

◦ We consider this to be a huge validation of the DataFlex framework
Which Style Should You use?

- The choice is yours
- One is not better than the other
  - They excel in different environments
- The two styles can mixed in a single application, but we don't encourage this as a long term strategy
- Don't underestimate the desktop/user driven mode
  - It's unique, powerful and if you have the screen, the mouse and the keyboard it does things the mobile/touch style cannot
  - It is ideally suited from moving large Windows business applications to a desktop browser
Understanding the Drilldown style

- The drilldown style represents a different way to build a DataFlex application
- While it still uses views, the rules for connecting views has changed
- There are some important new concepts that must be understood

- Let’s get started
Selects and Zooms

- The drilldown style supports two types of views - Selects and Zooms

- Selects
  - These will tend to be list based and are used for making selections
  - A selection event may
    - Navigate forward to another view (a drilldown)
    - Navigate back returning that selection to the invoking view

- Zooms
  - These tend to be form based and are used with a single record set
  - These are used to view data, add data, and edit data

- The view type is determined by the peViewType property
View Navigation

- Application flow is determined by forward and back navigation between views

- **Forward Navigation**
  - Any view can navigate to any other view – this is forward navigation
  - When this happens the view is added to a view-stack
  - The view stack is visualized by a breadcrumb control

- **Back Navigation**
  - You can navigate back to any view in the stack
  - This will close all views in front of it in the stack
  - You navigate back a single level or multiple levels
Sensible View Navigation

- Forward navigation to a new view should be *sensible*. For example:
  1. A Select view might navigate to a Zoom view with same main DDO
  2. A Select view might navigate to a Select view that is a list of children
  3. A Zoom view might navigate to a Select view that is a list of parents

- When the forward navigation is sensible, it does very sensible things. For example:
  1. A Select to Zoom with same main DDO is for viewing, editing records
  2. A Select to Select that are its children is used for viewing constrained child records
  3. A Zoom to Select of parents is used to select a parent (prompt list)

- The only restriction on forward navigation is you cannot navigate to a view that is already in your view stack
Determining View Navigation

- The developer determines how views are linked and what they do when they navigate forward and back
  - This defines application flow
  - You write code to do this – applications are developer driven
- A single view can be used for multiple purposes
  - For example, a single view could be used to:
    - zoom to details
    - view a list constrained children
    - act as a prompt list.
  - A single view displayed view can have multiple purposes
  - Using multi-purpose views makes things a little more complicated
  - It is not required but it is a very powerful feature
  - Views remain a reusable and multi-purpose component of a framework application
The most important part of controlling view navigation is knowing how it was invoked.

- You want to know where it came from and why
- This is referred to as its context
- We have created a model and an API for defining and using this context
- The most important part of this context is knowing where it navigated-from
- We have defined four navigation-from types
- It is critical that you understand these types
The Navigate-From types

- The navigate-from types are:
  - nfFromMain
    - From and to views have the same main DDO
  - nfFromParent
    - From view is a parent, to view will be constrained child records
  - nfFromChild
    - From view is a child, to view will be parents for this child (often a selection list)
  - nfUnknown
    - Not defined, custom code will determine what to do
Understanding the Navigate-From Types

- Let’s look at WebOrderMobile to understand this
Forward Navigation

How to Navigate Forward

- You send the NavigateForward message

Send NavigateForward of oToView eFromType hol nvokingObject

  Send NavigateForward of oZoomCustomer nfFromMain self
  Send NavigateForward of oSelectOrder nfFromParent self
  Send NavigateForward of oSelectCustomer nfFromChild self

What happens

- The message is sent to the view to be activated
- You pass the navigate-from type and the invoking object (usually self)
- The new view is activated and added to the stack
- The view uses the navigate-from type information to properly initialize itself
- The navigate-from type context information is stored in the view
- OnNavigateForward can be used to customize your view upon activation
- Any event (row select, menu click, etc.) can look at the navigate-from context to make choices to navigate forward, navigate back or whatever
Accessing the navigate-from context information

- The navigate-from type is the most important information about a navigation.
- It is stored in a navigation-data struct type that contains other relevant information about a forward navigation and back navigation.
- This is defined as:

```
Struct tWebNavigateData
    String sRowID
    Integer iTable
    Integer iColumn
    Integer eNavigateType
    Boolean bNewRecord
    Boolean bReadOnly
    Integer eViewTask
    String sData
End_Struct
```

- It is stored in a web-property and can be accessed using

```
Get GetNavigateData to NavigateData
Send SetNavigateData NavigateData
```
The Navigate Forward Events

- **OnGetNavigateForwardData**
  - During forward navigation the framework will assign the proper `tWebNavigateData` for you
  - It will then send `OnGetNavigateForwardData` to the object that started the navigation (the invoking object)
  - You can use this to customize your `tWebNavigateData` data

- **OnNavigateForward**
  - This is sent to the view being navigated-to
  - It is sent after the `tWebNavigateData` data has been assigned
  - You can use this to customize how your view looks and behaves
  - You will use this all the time
Back Navigation

- **How to Navigate Back**
  - **NavigateClose**
    - Send NavigateClose hoCallbackObject
  - **NavigateCancel**
    - Send NavigateCancel
    - Send NavigateClose self
    - Send NavigateCancel

- **What happens**
  - The message is sent to the top view
  - The top view will attempt to close
  - If data loss exists
    - A warning dialog may be presented
    - A save may be attempted
    - The close/cancel may be halted
  - If NavigateClose, the invoking view will be updated as needed
  - If NavigateCancel, no update occurs
The Navigate Close Events

- During a NavigateClose (but not a NavigateCancel) these events are sent:
  - OnGetNavigateBackData
    - During back navigation the framework will assign the proper tWebNavigateData data for you
    - It will then send OnGetNavigateBackData to the object that started the back navigation (the object passed in NavigateClose)
    - You can use this to customize your tWebNavigateData data
  - OnNavigateBack
    - This is sent to the view being navigated back-to
    - It is sent after the tWebNavigateData data has been assigned
    - You can use this to customize how your should handle an update
    - Most of the time, the automatic update will already do what you want
Back Navigation via Breadcrumb

- Selecting an item in the breadcrumb control
  - Closes all views in front of that item
  - If data loss exists
    - A warning dialog may be presented
    - A save may be attempted
    - The cancel may be halted
  - This is a cancel event – the invoking view is not updated
Building views

- Let’s build some views and look at some code
The Mobile/Touch Drilldown Summary

- There is more to this, but it is important you understand the basics. Here is what you need to know.

- **View types** – Views can be Selects or Zooms.
- **Navigation** - Views are connected via forward navigation:
  - Views are maintained in a view-stack and visualized with a breadcrumb control.
  - The developer must code these connections.
- **A view’s context** is determined by where it came from. Those navigate-from types are:
  - From-Main
  - From-Parent
  - From-Child
  - From-Undefined
- A single view can be used on a variety of contexts.
- Based on the view’s navigate-from context that view will just do the right thing.
- You can and will customize view behaviors based on the navigate-from context.
DataFlex Mobile Web

- Thank you