BlackBerry 10 Cascades UI FW:
A Different Take

Markus Landin, Product Manager, Research In Motion TAT
November 19, 2012
Session abstract:

“Cascades is the new native UI Framework on BlackBerry 10 Platform. It is built with Qt as the foundation and while it has many similarities to the QtQuick UI Framework there are also many fundamental architectural differences between the two.

The session will present an overview of Cascades and discuss the design choices made when architecting the new framework.”
BlackBerry 10 UI

Be more than an app, be a platform

Efficient Ergonomics

Fluid Workflow

Cinematic Experience

Communication at its core

Content is king

Performance is fundamental

Moments of Charm
BlackBerry 10 UI

*Be more than an app, be a platform*

Efficient Ergonomics

communication at its core

Fluid Workflow

Content is king

Moments of Charm

Cinematic Experience

Performance is fundamental
MIHALY CSIKSZENTMIHALYI

Das flow-Erlebnis

Klett-Cotta

Jenseits von Angst
und Langeweile:
im Tun aufgehen

8. Auflage

Konzepte der
Humanwissenschaften
“Flow is the mental state of operation in which a person performing an activity is fully immersed in a feeling of energized focus, full involvement, and enjoyment in the process of the activity.” [Wikipedia]
“BlackBerry Flow is a seamless user experience which provides full control and flexibility in every moment and every touch. Flow keeps the momentum going so that user goals can be achieved quick and efficiently”
"Vilken ära och heder"

"En milstolpe" Flera idrottare med funktionsnedsättning nominerade till Idrottsgalan.

svt.se/sport/
What is Cascades?

Cascades Developers
QtGui Developers
Game Developers

UI
Qt
QtGui
Qt
Quick

CouchDB
openAL
SDL
Box2D
Scoreloop
SQLite
cURL
boost
Bullet
unity
eGL
OpenGL
Cascades!? You don’t like Quick 2?

http://www.flickr.com/photos/kalavinka/4617897952/
We do like QtQuick2 but...

- Qt 5 won’t be ready in time for BlackBerry 10
- QML only
- No set of native controls
- UI thread can block rendering
- We already have TAT’s Cascades engine
Prevailing principles in Cascades Design

- Very limited time for implementation
- Locked down so can be extended later
- Uniform look for built in controls
- Minimum Developer Effort for Maximum Effect
- Built-in controls first, custom controls later
- Parity between QML and C++ API
Designed for BlackBerry 10 platform from the start

- Cross-platform API is not the focus
- Rich set of controls designed and optimized for BlackBerry 10
- No legacy to be supported
- Simplifies the requirements
QtQuick2 and Cascades: Similarities

- Scene Graph
- Use of QML
- Rendering in a separate thread
Cascades Architecture

- **App**: Creates, Modifies
- **Client**:
  - Page
  - Container
  - Button
  - Label
- **Server**:
  - Page
  - Container
  - Button
  - Label
- **Render Engine**: Visualizes

Signals, updates, Requests data
Client-Server Architecture

- Fully asynchronous
- Client side “pushes” data to server
- Server to client communication is limited
- Client and server scene graphs can be different
- Implementation complexity is hidden on the server
Scene graph structure

- A bit more structured approach
- Declarative list properties per type of objects
- Visual tree is a sub-tree of ownership tree
Container {
  Button { } // added to default property controls
  Button { text: business.buttonText }

  animations: [
    TranslateTransition { id: anim; fromX: 0; toX: 50 }
  ]

  actions: [
    ActionItem {
      text: "Play"
      onTriggered: anim.play()
    },
    ActionItem {}
  ]

  attachedObjects: [
    MyBusinessObject { id: business }
  ]
}
Event Handling

- Application subscribes to events using slots
- Server can handle most events by itself
- Low and high level events
  - Low Level (Touch, Enter/Exit)
  - High Level (Button clicked)
- Sophisticated touch behaviors possible
- Event phases: capture, target, bubbling
- Gestures support
Container {
  Container {
    preferredWidth: 300; preferredHeight: 300; background: Color.Blue
    onTouch: {
      background = event.isUp() ? Color.Blue : Color.Green
      translationX = event.windowX - (preferredWidth / 2);
      translationY = event.windowY - (preferredHeight / 2);
    }
  }
  touchBehaviors: [
    TouchBehavior {
      TouchReaction {
        eventType: TouchType.Down
        phase: PropagationPhase.AtTarget
        response: TouchResponse.StartTracking
      }
    }
  ]
}

Container {
  background: Color.Red; preferredWidth: 400; preferredHeight: 400
  overlapTouchPolicy: OverlapTouchPolicy.Deny
}
}
Animation

- Implicit animations are enabled by default
- All animations run on server
- No intermediate updates for animated properties
  - 99.3% of the time no one cares
  - subscribe to "*Changing()" signal to receive intermediate update
import bb.cascades 1.0

Page {
  content: Container {
    Button {
      text: "Click me"
      animations: [
        TranslateTransition {
          id: anim
          toX: 400
          duration: 3000
          easingCurve: StockCurve.ElasticIn
        }
      ]
      onClicked: {
        anim.play();
      }
      onTranslationXChanging: {
        console.log(translationX);
      }
    }
  }
  onClicked: {
    anim.play();
  }
}

import bb.cascades 1.0

Page {
  content: Container {
    Button {
      text: "Click me"
      attachedObjects: [
        ImplicitAnimationController {
          propertyName: "translationX"
          enabled: false
        }
      ]
      onClicked: {
        // not animated
        translationX += 20;
        // animated
        translationY += 20;
      }
    }
  }
  onClicked: {
    // not animated
    translationX += 20;
    // animated
    translationY += 20;
  }
}
Rich set of built in controls
Rich set of built-in controls
Rich set of built in controls
Extendibility

- Extension point: CustomControl
  - intended to be subclassed from C++
  - Root node is VisualNode
- QML files can be used as “custom controls”
- ForeignWindowControl
  - Embedding platform windows into the scene
  - Allows custom rendering
```cpp
#include <bb/cascades/CustomControl>
#include <bb/cascades/Container>
#include <bb/cascades/Button>
#include <bb/cascades/ImageView>

// mycontrol.h
class MyControl : public CustomControl {
Q_OBJECT
public:
    MyControl() : CustomControl() {
        setRoot(Container::create()
            .add(Button::create("Custom Button!"))
            .add(ImageView::create("asset:///image.png")));
    }
~MyControl() {};
};
QML_DECLARE_TYPE(MyControl)

// mycontrol.cpp
// register for use in qml if we want
qmlRegisterType<MyControl>("my.module", 1, 0, "MyControl");
// can use from C++
MyControl myControl;
Container *c = Container::create().add(&myControl);
```

```qml
// QML custom control
// defined in MyQmlControl.qml
Container {
    Button {
        text: "Qml Custom Button"
    }
    ImageView {
        imageSource: "image.png"
    }
}

// using both controls from QML
Container {
    MyControl {
        preferredWidth: 500
        preferredHeight: 500
        opacity: 0.5
    }
    MyQmlControl {}
}
```
C++ APIs mirror QML

- C++ and QML API is identical in 95%
- Have fun with QML but can drop to C++ if needed
- Builders for ease of use
import bb.cascades 1.0

Page {
  content: Container {
    Button {
      text: "Click me"
      imageSource: "asset:///images/image.png"
      verticalAlignment: VerticalAlignment.Center
      opacity: 0.5
      onClicked: opacity = undefined
    }
  }
}

#include <bb/cascades/Page>
#include <bb/cascades/Button>

Button *button;

Page *page = Page::create()
  .content(Container::create()
    .add(button = Button::create()
      .text("Click me")
      .imageSource("asset:///images/image.png")
      .verticalAlignment(VerticalAlignment::Center)
      .opacity(0.5f)
      .onClicked(button, SLOT(resetOpacity()))));
Layout

- Layout performed on server, animated
- No anchors or containers with predefined layout
- "Traditional" layout API
  - Container has a Layout, Controls have LayoutProperties
- Two kinds of layouts:
  - Container – for Controls
  - ListView – for ListView items
- Currently non-extendable
Container {
  layout: DockLayout {}

  Button {
    horizontalAlignment: HorizontalAlignment.Right
    verticalAlignment: VerticalAlignment.Bottom
  }

  Button {
    horizontalAlignment: HorizontalAlignment.Center
    verticalAlignment: VerticalAlignment.Center
  }
}

Container {
  layout: StackLayout {
    orientation: LayoutOrientation.LeftToRight
  }

  Button {
    layoutProperties: StackLayoutProperties {
      spaceQuota: 1
    }
  }

  Button {
    layoutProperties: StackLayoutProperties {
      spaceQuota: 2
    }
  }
}
Resource Handling

- Assets: loaded synchronously on render thread
- Content: loaded asynchronously, shown with effects
Simplified DataModels for ListView

- Why not QAbstractItemModel?
  - too complicated
  - different item types not supported

- Cascades’ DataModel
  - Very lightweight (4 virtual methods, 4 signals)
  - Can easily wrap QAbstractItemModel-based models
ListView {
  dataModel: XmlDataModel { source: "models/items.xml" }

  listItemComponents: [
    ListItemComponent {
      type: "header"
      Header {
        title: ListItemData.title
        subtitle: ListItemData.subtitle
      }
    },
    ListItemComponent {
      type: "listItem1"
      StandardListItem {
        title: ListItemData.title
        description: ListItemData.subtitle
      }
    },
    ListItemComponent {
      type: "listItem2"
      Container {
        Label {
          text: ListItemData.title
        }
      }
    }
  ]
}

// items.xml
<root>
  <header
    title="Fruits"
    subtitle="Generally sweet"/>
  <listItem1
    title="Oranges"
    subtitle="Sweet" />
  <listItem1
    title="Bananas"
    subtitle="Kinda sweet" />
  <header
    title="Vegetables"
    subtitle="Generally not so sweet"/>
  <listItem2
    title="Broccoli"/>
  <listItem2
    title="Potatos"/>
</root>
UI Adaptability
UI Adaptability

- Built in controls adapt to device type
- Smart usage of layouts
- Unique (sub)set of assets per configuration
UI Adaptability

Static asset selectors and the application’s assets folder structure:

assets/

720x720/
  main_screen.qml
  picture.png
main_screen.qml
dialog.qml
picture.png
icon.png
Interoperability with QtQuick

- Non-visual elements can be used in Cascades (States, Timer)
- Integration within same process would be hard
- Can interoperate as Cards
  - Transparent to the user
  - Same as other runtimes (AIR)
QML editor with syntax highlighting/code completion

QML design view

Component library (Drag and drop to QML)

QML tree view, QML properties view
Future

- More core controls
- Fun stuff (custom shaders, particles)
- Visual editor
- Moving to Qt5
Beta 3 just **arrived.**

http://developer.blackberry.com/cascades
Learn more at this event

<table>
<thead>
<tr>
<th>Day</th>
<th>Time</th>
<th>Room</th>
<th>Title</th>
<th>Speaker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wed</td>
<td>15:00</td>
<td>A keynote</td>
<td>Qt and the upcoming BlackBerry 10 Platform</td>
<td>Alec Saunders</td>
</tr>
</tbody>
</table>
Thank you!

November 19, 2012