

# Creating Flexible Components for Reuse and Distribution

Ben Clinkinbeard  
<http://www.returnundefined.com>



## Who is this guy?

- Design degree... whoops
- ActionScript 1 == first programming language
- Design agency to Fidelity Investments to UM
- Flex 2 Beta 2

## What are we going to cover?

- Building single-use components (usually) sucks
- Weigh the benefits and use good judgment
- Components should be:
  - easy to utilize
  - reusable
  - customizable
  - style friendly
  - useful (not just usable)

Why?

# You

- **Good:** Creating new components and applications
- **Bad:** Supporting old components
- **Worse:** Not supporting old components and getting hypothetically fired

## People using your component

- **Good:** Your component just works
  - Discovery of depth and elegance can/should happen later
- **Bad:** It takes 15 minutes of reading docs and code to get up and running

OK smart guy, how?

## flexmdi

- Project site - <http://code.google.com/p/flexmdi/>
- Brian Holmes - <http://brianjoseph31.typepad.com/smashedapples/>
- Brendan Meutzner - <http://www.meutzner.com/blog/>
  
- Conceived at 360|Flex Seattle, August 2007
- Released September 2007
- News at 11:00

## Composition over “in there”-itance

- Division of responsibilities
  - Classic tenet of OOP and common refactoring tactic
- One component != one class
- Expose modularity / assignment
  - MDIWindowControlsContainer
  - flexmdi effects

## Hide the details

- Another core principle of OOP - Encapsulation
- Less exposure means more freedom to change but...
- Don't be stingy
- Automate tedious tasks
  - MDICanvas
  - `mdiManager.windowEventProxy()`

## Providing default behaviors (your job)

- Listen for own events
- Low priority listener to ensure late/last execution
- `EventPriority.DEFAULT_HANDLER:int = -50`
  - `addEventListener(type:String, listener:Function, useCapture:Boolean = false, priority:int = 0, useWeakReference:Boolean = false);`
- Events must be cancelable
  - `Event(type:String, bubbles:Boolean = false, cancelable:Boolean = false);`

## Modifying default behaviors (their job)

- “Normal” listeners will get called first
  - `mdiWindow.addListener(MDIManager.WINDOW_CLOSE, onWindowClose);`
- Cancel default handler with `event.preventDefault()`
- Use `event.clone()` to store copy for later execution

## Executing default behaviors (your job)

- Default handler is public for delayed / manual calls
  - `mdiManager.executeDefaultBehavior(event);`
- Behavior is conditional
  - `if( !event.isDefaultPrevented() )`

## Follow conventions

- Use life cycle functions when possible / appropriate
  - MDIWindowControlsContainer
- Styles as styles
- Metadata...

## Provide default styles

- Static initializer
  - `private static function initializeStyles(){...}`
  - `private static var stylesInitialized():Boolean = initializeStyles();`
- Docs are wrong - use `defaultFactory()`, not `setStyle()`
  - `MDIWindow.classConstruct();`

## Metadata (is mandatory)

- Compiler instructions
- Code completion and MXML assignments
- [Event(name="minimize", type="flexmdi.events.MDIWindowEvent")]
- [Style(name="closeBtnStyleName", type="String", inherit="no")]
- [Bindable]

## Make things easy

- Expose things the framework doesn't
  - `mdiWindow.getTitleTextField()`
- Proxy properties
  - `mdiCanvas.enforceBoundaries`
- Provide base implementations
  - `MDIEffectsDescriptorBase`

# Thank you!

(tip your waitress)