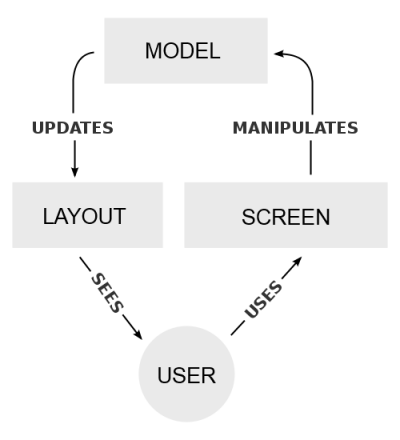
**Screens**

Game Framework вводит идею экранов для распределения различных обязанностей при представлении определенных результатов пользователям. Экраны в основном основаны на шаблоне проектирования MVC.



* Модель (M) - оповещает связанный с ним макет(представление) при изменении состояния экрана. Это оповещение позволяет макету обновлять результат. Это достигается с помощью привязок свойств представления.
* Макет (V) (представление) - запрашивает информацию из модели, которую он использует для генерации выходного представления для использования
* Экран (C) - может отправлять команды модели для обновления состояния модели. Он также может отправлять команды в связанное с ним представление, чтобы изменить представление соответствующей модели.

Почему экраны? Потому что для 2D-игр вполне естественно управлять определенными аспектами отображения в одном месте. Экран представляет собой единый дисплей, характеризующийся его связностью. Он состоит из набора элементов управления, некоторых данных, которые должны быть представлены, и способа контроля ими.

**Основные характеристики**

**Модели (MODELS)**

Классы моделей должны наследоваться от класса ScreenModel. Существует три способа предоставления данных в модели:

* Определяя свойства типа NamedProperty - это предпочтительный способ.
* Передавая пару «ключ-значение» методу setProperty. Это часто используется как хранилище для просмотра, которое передает определенные данные Макету(представлению) в виде словаря.
* Предоставляя карту ключ/значение с помощью setExternalPropertyMap - как и в предыдущем примере, только словарь может быть введен снаружи.

Это обычно используется для локализации. Все эти способы позволяют впоследствии связать данные с соответствующим расположением. Последние два способа должны определять уникальные ключи свойств, поскольку они доступны из XML с использованием того же синтаксиса. На данный момент свойства модели не являются иерархическими и, как следствие, невозможно оповестить о сложных свойствах, которые приводят к еще одним свойствам (NamedProperty приводит к другому NamedProperty). Модель образца с использованием первых двух

SampleScreenModel.java[¶](file:///N:\job2\frame\6\Screens%20%E2%80%94%20GOX%201.0.0%20documentation.htm#id6)

|  |
| --- |
| public SampleScreenModel extends ScreenModel {  private final NamedProperty<Integer> height = new NamedProperty<Integer>(Integer.class, "height", 100);  public SampleScreenModel() {  this.setProperty("width", 200);  }  public void setHeight(Integer value) {  this.height.set(value);  }  public Integer getHeight() {  return this.height.get();  }  public IObservableProperty<Integer> height() {  return this.height;  }  } |

# Макеты (Layouts)(Представления)

Макеты - это иерархии представлений, управляемые IViewManager. Они описаны в разделе «Представления» руководства. Для примера была определена следующая типовая схема.

resources.xml[¶](file:///N:\job2\frame\6\Screens%20%E2%80%94%20GOX%201.0.0%20documentation.htm#id7)

|  |
| --- |
| ...  <resource id="sample" type="Layout" url="sample.xml">  ... |

sample.xml[¶](file:///N:\job2\frame\6\Screens%20%E2%80%94%20GOX%201.0.0%20documentation.htm#id8)

|  |
| --- |
| <ViewGroup xmlns="http://www.atsisa.com/gox/framework/view">  <children>  <TextView id="myText" />  </children>  </ViewGroup> |

# Экраны (Screens) (Controller)

Экраны используют модели и макеты и определяют способы их контроля. Все зарегистрированные макеты имеют соответствующие экраны. Экранные классы должны наследоваться от класса Screen и содержать атрибут @Reflectable, чтобы включить поддержку отражения для платформы HTML.

* Модели не являются обязательными для экранов

    Если вы решите не использовать модель с вашим макетом, просто опустите общий параметр при наследовании от класса Screen и используйте конструктор, который не требует определения модели. В противном случае укажите экземпляр вашей модели.

* Макеты могут быть определены в XML и в коде

    Идентификатор макета является единственным требуемым параметром конструктора для класса Screen. После создания экрана он пытается найти правильный макет на основе XML в ViewManager. Если макет не существует, экран возвращается к методу createLayout, который должен создать макет в коде. По умолчанию метод возвращает пустую ViewGroup и может быть переопределен для удовлетворения пользовательских настроек.

* Идентификатор экрана

    Идентификатор экрана совпадает с идентификатором соответствующего макета и идентификатором ресурса макета. Он должен быть уникальным для всей игры. Это может быть любая строка для макетов, созданных в коде.

* Регистрация и поиск экрана

    Нет макетов без соответствующих экранов. Если конкретный экран не был зарегистрирован, игровая инфраструктура автоматически зарегистрирует экземпляр класса BasicScreen для каждого незарегистрированного макета. Экраны можно найти позже, используя метод getScreenById или getAllScreens в экземпляре вашего диспетчера представлений (GameEngine.current (). GetViewManager ()). Каждый добавленный экран должен быть зарегистрирован. Существует два способа сделать это:

* + зарегистрируйте экран в коде с использованием интерфейса IViewManager. Таким образом, после этого, метод initialize () на экране должен вызываться перед первым использованием.
  + зарегистрируйтесь в модуле IoC, который определяет зависимости ядра игры. Пример ниже показывает, как привязать экран.

SampleCoreModule.java[¶](file:///N:\job2\frame\6\Screens%20%E2%80%94%20GOX%201.0.0%20documentation.htm#id9)

|  |
| --- |
| protected void configure() {  ...  bind(Screen.class).to(SampleScreen.class);  ...  } |

# Жизненный цикл экрана

Все экраны, зарегистрированные в IViewManager, существуют все время игры. Жизненный цикл экрана основан на жизненном цикле компоновки. Экраны могут быть показаны или скрыты с помощью методов show и hide соответственно или с помощью команд: ShowScreenCommand и HideScreenCommand, которые могут быть опубликованы с помощью событий. Чтобы контролировать определенное поведение, все экраны могут переопределять следующие методы:

* beforeActivated - Вызывается, когда макет будет добавлен на сцену.
* afterActivated - вызывается, когда макет был добавлен на сцену
* beforeDeactivated - Вызывается, когда макет будет удален или уничтожен
* afterDeactivated - Вызывается, когда макет был удален или уничтожен.

SampleScreen.java[¶](file:///N:\job2\frame\6\Screens%20%E2%80%94%20GOX%201.0.0%20documentation.htm#id10)

|  |
| --- |
| @Reflectable  public SampleScreen extends Screen<SampleScreenModel> {  public SampleScreen() {  super("sample", new SampleScreenModel());  }  @Override  protected void beforeActivated() {  this.getEngine().getLogger().debug("beforeActivated");  }  @Override  protected void afterActivated() {  this.getEngine().getLogger().debug("afterActivated");  }  @Override  protected void beforeDeactivated() {  this.getEngine().getLogger().debug("beforeDeactivated");  }  @Override  protected void afterDeactivated() {  this.getEngine().getLogger().debug("afterDeactivated");  }  } |

SampleGame.java[¶](file:///N:\job2\frame\6\Screens%20%E2%80%94%20GOX%201.0.0%20documentation.htm#id11)

|  |
| --- |
| public SampleGame extends AbstractGame {  public void onReady() {  SampleScreen screen = new SampleScreen();  GameEngine.current().getViewManager().registerScreen(screen);  screen.initialize();  }  public void onStart() {  GameEngine.current().getViewManager().getScreenById("sample").show();  }  } |

На каждом экране публикуются события, связанные с изменением его видимости:

* ScreenShowingEvent: когда экран начинает показывать
* ScreenShownEvent: когда появился экран
* ScreenHidingEvent: когда экран начал скрываться
* ScreenHiddenEvent: когда экран был скрыт

Каждое событие содержит информацию о идентификаторе экрана, к которому он относится.

# Связывание с макетами

Связывание макетов - это самый быстрый способ сделать ваш экран восприимчивым к изменениям с течением времени. Существуют разные типы привязок.

# Просмотр привязок

Механизм представления привязки был введен в ссылки на просмотр autwire и для ограничения использования метода findViewById. Чтобы использовать эти привязки, все, что вам нужно сделать, - это украсить ваше поле вида атрибутом @InjectView. На данный момент ваше поле должно быть общедоступным (это связано с тем, что невозможно вводить значение в частные поля с использованием отражения GWT). Кроме того, дополнительный атрибут id может быть предоставлен атрибуту @InjectView для привязки к идентификатору, отличному от имени поля.

sample.xml[¶](file:///N:\job2\frame\6\Screens%20%E2%80%94%20GOX%201.0.0%20documentation.htm#id12)

|  |
| --- |
| <ViewGroup xmlns="http://www.atsisa.com/gox/framework/view">  <children>  <TextView id="myText" />  </children>  </ViewGroup> |

SampleScreen.java[¶](file:///N:\job2\frame\6\Screens%20%E2%80%94%20GOX%201.0.0%20documentation.htm#id13)

|  |
| --- |
| public SampleScreen extends Screen<SampleScreenModel> {  @InjectView  public TextView myText;  public SampleScreen() {  super("sample", new SampleScreenModel());  }  @Override  protected void afterActivated() {  this.getEngine().getLogger().debug("afterActivated");  this.myText.setText("Hello");  }  } |

# Просмотр привязок свойств

Это отложенные привязки, которые связывают свойства представления с моделью. Благодаря им свойства просмотра автоматически обновляются и следуют изменениям модели. В настоящее время допускаются только однонаправленные привязки (в которых свойства модели обновляют свойства макета). Существует три способа доступа к данным модели из XML:

     Локальная привязка - доступ к экземпляру класса NamedProperty с использованием синтаксиса propertyName,

     Внешняя привязка - доступ к свойству, которое ранее было задано с использованием метода модели setProperty или getProperty, с использованием синтаксиса #propertyKey,

     Связывание шаблонов - доступ к свойству, которое ранее было задано с использованием метода модели getTemplate, с использованием синтаксиса regex «{# (. \*?)», Например. "- {# propertyKey1} - {# propertyKey2} -". Модель также позволяет регистрировать свойства в виде пар ключ-значение, чтобы их можно было получить в макете или экране.

#### Example

SampleScreenModel.java[¶](file:///N:\job2\frame\6\Screens%20%E2%80%94%20GOX%201.0.0%20documentation.htm#id14)

|  |
| --- |
| public SampleScreenModel extends ScreenModel {  private static final String TEMPLATE\_KEY = "TEMPLATE\_KEY";  private final NamedProperty<Integer> height = new NamedProperty<Integer>(Integer.class, "height", 100);  public SampleScreenModel() {  this.setProperty("width", 200);  }  public void setHeight(Integer value) {  this.height.set(value);  }  public Integer getHeight() {  return this.height.get();  }  public IObservableProperty<Integer> height() {  return this.height;  }  public void setTemplate(String value){  NamedProperty namedProperty = getTemplate(TEMPLATE\_KEY);  namedProperty.setName(value);  namedProperty.set(translator.translateTemplate(value));  }  public IObservableProperty<String> template() {  return getTemplate(TEMPLATE\_KEY);  }  } |

SampleScreen.java[¶](file:///N:\job2\frame\6\Screens%20%E2%80%94%20GOX%201.0.0%20documentation.htm#id15)

|  |
| --- |
| public SampleScreen extends Screen<SampleScreenModel> {  public SampleScreen() {  super("sample", new SampleScreenModel());  }  @Override  protected void afterActivated() {  this.getModel().setProperty("myTextValue", "Hello bindings!");  this.getModel().setProperty("text1", "Hello text 1!");  this.getModel().setProperty("text2", "Hello text 2!");  }  } |

sample.xml[¶](file:///N:\job2\frame\6\Screens%20%E2%80%94%20GOX%201.0.0%20documentation.htm#id16)

|  |
| --- |
| <ViewGroup xmlns="http://www.atsisa.com/gox/framework/view">  <children>  <TextView width="{width}" height="{#height}" text="{#myTextValue}" />  <TextView text="This is {#text1} and {#text2}!" />  <TextView text="{template}" />  </children>  </ViewGroup> |

В следующем примере показано, как добавить свойство в SampleScreenModel.

SampleScreenModel.java[¶](file:///N:\job2\frame\6\Screens%20%E2%80%94%20GOX%201.0.0%20documentation.htm#id17)

|  |
| --- |
| private final NamedProperty<Integer> property1 = new NamedProperty<>(Integer.class, "property1", 0);  private final NamedProperty<Integer> property2 = new NamedProperty<>(Integer.class, "property2", 0);  private void registerProperties() {  registerProperty(property1);  registerProperty(property2);  } |

И доступ к использованию.

sampleLayout.xml[¶](file:///N:\job2\frame\6\Screens%20%E2%80%94%20GOX%201.0.0%20documentation.htm#id18)

|  |
| --- |
| <TextView width="570" color="#ffffff" height="38" x="56" y="61" fontSize="30" valign="MIDDLE" halign="CENTER"  text="{property1}/{property2}"/> |

# Связывание с методом экрана

Связи с экранными методами могут использоваться либо в действиях UpdateScreen, либо в свойствах вида типа EventListenerProperty. Первый используется в дескрипторах очереди действий, предоставляя параметры метода и screenId для действия UpdateScreen. Второй использует синтаксис #methodName в обоих случаях. Связывание приведет к вызову определенного метода Screen при условии, что метод был украшен атрибутом @ExposeMethod.

# Пример вызова метода экрана с использованием дескриптора очереди действий

SampleScreen.java[¶](file:///N:\job2\frame\6\Screens%20%E2%80%94%20GOX%201.0.0%20documentation.htm#id19)

|  |
| --- |
| @Reflectable  public SampleScreen extends Screen<SampleScreenModel> {  public SampleScreen() {  super("sample", new SampleScreenModel());  }  @ExposeMethod  public void updateFromQueue() {  this.getEngine().getLogger().debug("Updated from action queue");  }  } |

actions.xml[¶](file:///N:\job2\frame\6\Screens%20%E2%80%94%20GOX%201.0.0%20documentation.htm#id20)

|  |
| --- |
| <queues>  <queue name="SomeQueue">  <actions>  <UpdateScreen screenId="sample" method="updateFromQueue" />  </actions>  </queue>  </queues> |

# Пример вызова метода экрана с использованием связывания слушателя

SampleScreen.java[¶](file:///N:\job2\frame\6\Screens%20%E2%80%94%20GOX%201.0.0%20documentation.htm#id21)

|  |
| --- |
| @Reflectable  public SampleScreen extends Screen<SampleScreenModel> {  public SampleScreen() {  super("sample", new SampleScreenModel());  }  @ExposeMethod  public void updateFromListener(IEvent event) {  this.getEngine().getLogger().debug("Updated from event listener. Event source: %s", event.getSource());  }  } |

sample.xml[¶](file:///N:\job2\frame\6\Screens%20%E2%80%94%20GOX%201.0.0%20documentation.htm#id22)

|  |
| --- |
| <ViewGroup xmlns="http://www.atsisa.com/gox/framework/view">  <children>  <ButtonView onRelease="#updateFromListener" />  </children>  </ViewGroup> |

# Связывание очереди действий

# Помимо привязки к экранным методам, GameEngine позволяет связывать интерактивные зрители с выполнением правильных очередей действий. Разница заключается в том, что вместо префикса # следует использовать префикс @. В следующем примере показано, как привязать событие onRelease к кнопке с исполнением очереди с именем «ActionQueueToExecute»

sample.xml[¶](file:///N:\job2\frame\6\Screens%20%E2%80%94%20GOX%201.0.0%20documentation.htm#id23)

|  |
| --- |
| <ViewGroup xmlns="http://www.atsisa.com/gox/framework/view">  <children>  <ButtonView onRelease="@ActionQueueToExecute" />  </children>  </ViewGroup> |

# Внешние привязки событий

# В следующем примере показано, как привязывать события myButton к выполнению либо экранного метода, либо очереди действий (как это было представлено ранее).

sample.xml[¶](file:///N:\job2\frame\6\Screens%20%E2%80%94%20GOX%201.0.0%20documentation.htm#id24)

|  |
| --- |
| <ViewGroup xmlns="http://www.atsisa.com/gox/framework/view">  <children>  <ButtonView id="myButton" />  </children>  </ViewGroup> |

resources.xml[¶](file:///N:\job2\frame\6\Screens%20%E2%80%94%20GOX%201.0.0%20documentation.htm#id25)

|  |
| --- |
| ...  <resource id="sample" type="Layout" url="sample.xml">  <resource id="bindings" type="Config" url="bindings.xml">  ... |

bindings.xml[¶](file:///N:\job2\frame\6\Screens%20%E2%80%94%20GOX%201.0.0%20documentation.htm#id26)

|  |
| --- |
| <Configuration>  <ActionBindings>  <ActionBinding targetId="myButton" event="MOUSE\_UP" action="#ScreenMethodToInvoke" />  <ActionBinding targetId="myButton" event="MOUSE\_DOWN" action="@ActionQueueToExecute" />  </ActionBindings>  </Configuration> |