

# Controller States

Example File (Builder project Advanced Modeling/UI):



<your example path>\Advanced Modeling\UI\uml\uiControllerState.xml

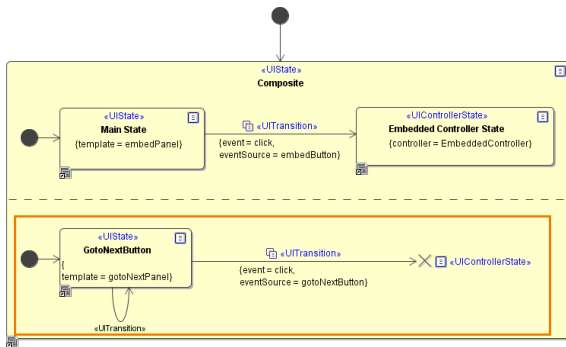
Each xUML UI web-application must have at least one **<<UI>>** controller which is representing, in traditional web development, a single HTML page. With the introduction of modern web technologies like jQuery, it is possible to implement a whole web application within one single page. Nevertheless, applications mostly consist of more than one page and this means more than one **<<UI>>** controller.

On this Page:

- [Linking Controllers](#)
- [Controller Data Binding](#)
  - [Private and Public Controller Attributes](#)
- [Embedded Controller States](#)
  - [Signal Events](#)

## Linking Controllers

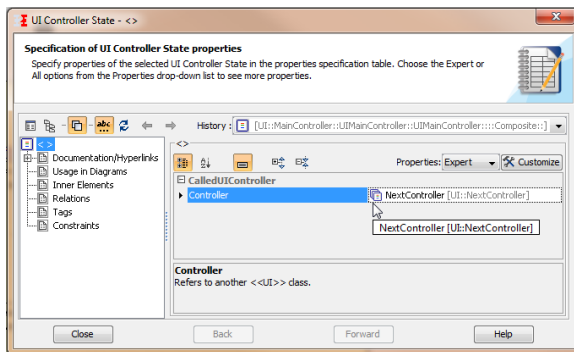
The linking between **<<UI>>** controllers is done within the state machine diagram. A link from one controller to the next means also leaving the page and load the next one, meaning terminating the current state machine and load the next one. To model this the **Terminate** element is used having in addition the stereotype **<<UIControllerState>>**.



Related Pages:

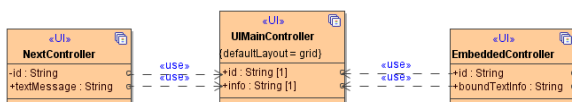
- [Authentication and Authorization](#)
- [File Upload](#)
- [HTTPS](#)
- [History State](#)
- [Form and Form Validation](#)
- [Calling a UI from external Applications](#)
- [Usage of Choices](#)
- [Service Calls](#)
- [HTTP Proxy](#)
- [Controller States](#)
- [Back Button and Browser History](#)
- [Mock-Ups](#)

To apply the **<<UI>>** controller which will represent the new page that will be loaded, the controller itself needs to be applied to the Terminate element. Which controller will be loaded is specified in the tagged value section of the Termination element.



## Controller Data Binding

In most cases data needs to be passed from one **<<UI>>** controller to the next, which means that the data between the controllers needs to be bound. This is done by modeling the bindings on the controllers class attributes using **<<use>>** dependencies.



In the above figure, the **NextController** and the **EmbeddedController** use the **id** and the **info** attribute of the **UIMainController**. The bound attributes will be shared between single pages using a url query string: [http://localhost:12485/ui/NextController.html?<attributeName>=<attributeValue>&\[...\]](http://localhost:12485/ui/NextController.html?<attributeName>=<attributeValue>&[...])  
The attributes, when bound to user interface elements (e.g. text fields, labels) will be automatically applied.

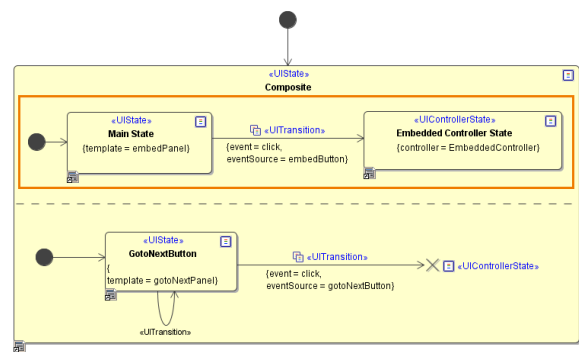
## Private and Public Controller Attributes

<<UI>> controller attributes can be set public or private. The effect of distinguishing between these two attributes properties results in if data is past from one controller to the next. If the attributes are set to public, the attributes will be passed to the next controller in form of a URL query string: <http://localhost:12485/ui/NextController.html?textMessage=Entry%20saved%20sucessfully!>

This link represents the Link from the UIMainController to the NextController. The attribute **id** is not passed due to it is set to private.


## Embedded Controller States

An embedded <<UI>> controller is a own in it closed application which can be embedded into the same page as the e.g. main application is running. Data between the embedded controller and its parent controller are bound the same way as with controllers representing a single page (compare with **NextController** in the example project).



When the **embedButton** receives the click event it will not change to a new page, but it will replace the **embedPanel** template content of the **Main State** with the template content **EmbeddedController** content of the **Embedded Controller State**.

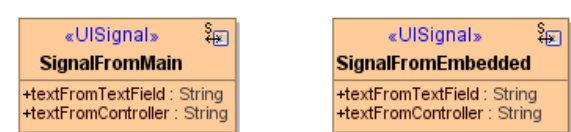
## Signal Events



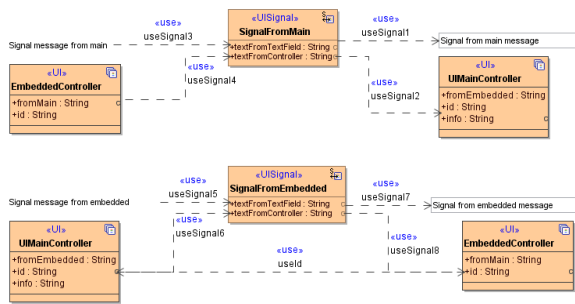
Example Files (Builder project Advanced Modeling/UI):

<your example path>\Advanced Modeling\UI\uml\uiSignalEvents.xml

Signal events allow the sending of messages between embedded controller and main or parent controller and vice versa. The <<UISignal>> is a UML class with attributes being the message content. There is no limit on attributes that can be modeled.



As with the <<UI>> controllers and messages, <<UISignal>> classes are bound to either to <<UI>> using <<use>> dependencies connecting the attributes. A signal should always have a source attribute and a target attribute binding. Also in the case of binding signal events the class diagram type is used.



Signals are sent using `<<UITransactions>>` where the signal is attached to the **Throw Signal** tagged value in the **UIAction** section.

