Component Configuration Options

On This Page

See Also

Understanding Infusion Components
Understanding Component Options And Their Defaults
Define defaults in a consistent order

Infusion components are configured using options that are defined by the component developer and customized by the integrator. While component developers are free to define whatever options are appropriate for their component, the Infusion Framework supports a number of predefined options. This page briefly describes these predefined options and provides links more information about the related Framework functionality.

Some predefined options should not be overridden by integrators: They are strictly for the use of the component developer. This is noted in the descriptions below.

Options Supported By All Components Grades

The following options are supported by all component grades:

[gradeNames] [nickName] [mergePolicy] [invokers] [members] [components] [dynamicComponents]

gradeNames

Description	An array of string grade names.
Notes	In addition to the grade names, the array should include the special `"autoInit"` value, which instructs the Framework to create the component creator function automatically. NOTE: `"autoInit"` is the preferred way of creating components, and will become the default for Infusion 2.0. Always use this grade name, unless you have a special reason to not want the framework to fabricate a creator function (perhaps, because your grade is not instantiable).

Example Definition	`fluid.defaults("component.name", {`
	`  gradeNames: ["fluid.modelComponent", "fluid.eventedComponent", "autoInit"],`
	`  ...`
	`});`

See also	Component Grades

nickName

Description	Specifies a custom nickname for the component. The nickname is used by the Framework as an extra context name which can reference the component. By default, the nickname is derived from the component name.
Notes	This option was historically used to work around various framework deficiencies that have now been corrected. It will be removed from an upcoming revision of the framework.

Example Definition	`fluid.defaults("component.name", {`
	`  nickName: "myComponentName",`
	`  ...`
	`});`

See also	fluid.computeNickName

mergePolicy

Description	An object providing instructions for how particular options should be merged when integrator options are merged with default values.
Notes	It is uncommon to need this option. The most common use case is to protect "exotic values" derived from some external library or framework from being corrupted by the options merging/expansion process by use of the "nomerge" policy.

Example Definition	`fluid.defaults("component.name", {`
	`  mergePolicy: {`
	`  option1: "noexpand",`
	`  option2: "nomerge",`
	`  ....`
	`  },`
	`  ...`
	`});`

See also	Options Merging

invokers

Description	An object defining methods on the component whose arguments are resolved from the environment as well as the direct argument list at invocation time.
Notes	

Example Definition	`fluid.defaults("component.name", {`
	`  invokers: {`
	`  inv1: {...},`
	`  inv2: {...},`
	`  },`
	`  ...`
	`});`

See also	Invokers

members

Description	An object defining properties to be added to the component object. These can be anything, including methods, strings, objects, etc. Definitions are evaluated as IoC expressions.
Notes	`members` differ from `invokers` in that the arguments of `members` are **not** resolved at invocation time. The right-hand-side may contain an expander definition, which may perhaps itself resolve onto an invoker.

Example Definition	`fluid.defaults("component.name", {`
	`  members: {`
	`  member1: "{that}.options.optionsValue",`
	`  member2: "{theOther}.dom.otherSelector",`
	`  },`
	`  ...`
	`});`

components

Description	An object containing named definitions of the component's subcomponents.
Notes	This (the ***subcomponent record***) is one of the core sources from which the options configuring a component in a particular context. The total set of options sources are: i) the original defaults record, ii) the subcomponent record, iii) direct user options (supplied to a component creator function), iv) distributed options

Example Definition	`fluid.defaults("component.name", {`
	`  components: {`
	`  subcomponent1: {`
	`  type: "component.subcomp1",`
	`  options: {...}`
	`  },`
	`  ...`
	`  },`
	`  ...`
	`});`

See also	Tutorial - Subcomponents

dynamicComponents

Description	An object containing named definitions of the component's dynamic subcomponents
Notes	Some special context names may be available within the subcomponent's definition block, for example `{source}` and `{sourcePath}` or `{arguments}`. This framework facility will be replaced by a more declarative equivalent in time and should be used with caution.

Example Definition	`fluid.defaults("component.name", {`
	`  dynamicComponents: {`
	`  dynamic1: {`
	`  type: "component.subcomp1",`
	`  source: "{context}.someArray",`
	`  options: {...}`
	`  },`
	`  ...`
	`  },`
	`  ...`
	`});`

See also	Tutorial - Subcomponents

Little Components

back to top Components defined with a grade of `littleComponent` support all of the common options described above, and no others. Component developers are free to define their own additional options.

See also: Component Grades

Model Components

back to top Components defined with a grade of `modelComponent/modelRelayComponent` support all of the common options described above, as well as those defined below. Component developers are free to define their own additional options.

See also: Component Grades

The following options are supported by model components:

[model] [applier] [changeApplierOptions]

model

Description	An object containing the data model to be used by the component.
Notes	If a ChangeApplier is not provided using the `#applier` option, the Framework will create one for the provided model.

Example Definition	`fluid.defaults("fluid.pager", {`
	`  model: {`
	`  pageIndex: undefined,`
	`  pageSize: 10,`
	`  totalRange: undefined`
	`  },`
	`  ...`
	`});`

Example Override	`var myPager = fluid.pager(container, {`
	`  model: {`
	`  pageIndex: 1`
	`  },`
	`  ...`
	`});`

See also	Model Objects
	ChangeApplier API
	`#applier`
	`#changeApplierOptions`

applier

Description	A ChangeApplier object for the model provided with the `#model` option.
Notes	It is not necessary to provide an applier: By default, an applier will be created with `fluid.makeChangeApplier()`, using any options specified with `#changeApplierOptions`. This option is most commonly used to share a common ChangeApplier between components in a component tree: the `applier` option can be used to reference the ChangeApplier of another component in the tree.

Example Definition	`fluid.defaults("component.name", {`
	`  applier: "{parentComponent.applier}",`
	`  ...`
	`});`

Example Override	N/A

See also	ChangeApplier API
	`#model`
	`#changeApplierOptions`

changeApplierOptions

Description	Options that will be passed on to `fluid.makeChangeApplier()` if a ChangeApplier is not provided using the `#applier` option.
Notes	If a ChangeApplier is provided using the `#applier` option, this option will be ignored.

Example Definition	`fluid.defaults("component.name", {`
	`  model: {...},`
	`  changeApplierOptions: {`
	`  cullUnchanged: true`
	`  },`
	`  ...`
	`});`

Example Override	`var myComp = component.name(container, {`
	`  model: {...},`
	`  changeApplierOptions: {`
	`  cullUnchanged: true`
	`  },`
	`  ...`
	`});`

See also	ChangeApplier API
	`#model`
	`#applier`

Evented Components

back to top Components defined with a grade of `eventedComponent` support all of the common options described above, as well as those defined below. Component developers are free to define their own additional options.

See also: Component Grades

The following options are supported by evented components:

events

Description	An object containing key/value pairs that define the events the component will fire: the keys are the event names, the values define the type of the event (see Infusion Event System for information on the different event types).
Notes	The Framework will create event firers for the listed events. It is the responsibility of the component to fire the events at the appropriate times.

Example Definition	`fluid.defaults("component.name", {`
	`  events: {`
	`  onSave: "preventable",`
	`  onReady: null`
	`  },`
	`  ...`
	`});`

See also	Infusion Event System

listeners

Description	An object defining listener functions for the events supported by a component.
Notes	Both component developers and integrators can define listeners for events. Invokers and Expanders can be used as listeners here. Note that as well as being a simple string holding the name of an event on this component, a listener key may also be a full IoC Reference to any other event held in the component tree (for example `"{parentComponent}.events.parentEvent"`. As well as being a simple function name, a the value associated with the key may be a Listener Record or else follow the syntax of an Invoker indicating that the registered listener receives a different signature from the one that the event has fired (see Event injection and boiling).

Example Definition	`fluid.defaults("component.name", {`
	`  events: {`
	`  onSave: "preventable",`
	`  onReady: null`
	`  },`
	`  listeners: {`
	`  onSave: "component.name.saveValidatorFn"`
	`  },`
	`  ...`
	`});`

Example Override	`var myComp = component.name(container, {`
	`    listeners: {`
	`        onReady: "myNamespace.myReadyNotificationFn",`
	`    },`
	`    ...`
	`});`

See also	Infusion Event System

View Components

back to top Components defined with a grade of `viewComponent` support all of the common options described above, as well as those defined below. Component developers are free to define their own additional options.

See also: Component Grades

The following options are supported by view components:

[model] [applier] [changeApplierOptions] [events] [listeners] [selectors]

model

Description	An object containing the data model to be used by the component.
Notes	If a ChangeApplier is not provided using the `#applier` option, the Framework will create one for the provided model.

Example Definition	`fluid.defaults("fluid.pager", {`
	`    model: {`
	`        pageIndex: undefined,`
	`        pageSize: 10,`
	`        totalRange: undefined`
	`    },`
	`    ...`
	`});`

Example Override	`var myPager = fluid.pager(container, {`
	`    model: {`
	`        pageIndex: 1`
	`    },`
	`    ...`
	`});`

See also	Model Objects
	ChangeApplier API
	`#applier`
	`#changeApplierOptions`

applier

Description	A ChangeApplier object for the model provided with the `#model` option.
Notes	It is not necessary to provide an applier: By default, an applier will be created with `fluid.makeChangeApplier()`, using any options specified with `#changeApplierOptions`. This option is most commonly used to share a common ChangeApplier between components in a component tree: the `applier` option can be used to reference the ChangeApplier of another component in the tree.

Example Definition	`fluid.defaults("component.name", {`
	`    applier: "{parentComponent.applier}",`
	`    ...`
	`});`

Example Override	N/A

See also	ChangeApplier API
	`#model`
	`#changeApplierOptions`

changeApplierOptions

Description	Options that will be passed on to `fluid.makeChangeApplier()` if a ChangeApplier is not provided using the `#applier` option.
Notes	If a ChangeApplier is provided using the `#applier` option, this option will be ignored.

Example Definition	`fluid.defaults("component.name", {`
	`  model: {...},`
	`  changeApplierOptions: {`
	`  cullUnchanged: true`
	`  },`
	`  ...`
	`});`

Example Override	`var myComp = component.name(container, {`
	`  model: {...},`
	`  changeApplierOptions: {`
	`  cullUnchanged: true`
	`  },`
	`  ...`
	`});`

See also	ChangeApplier API
	`#model`
	`#applier`

events

See `fluid.eventedComponent` above for details

listeners

See `fluid.eventedComponent` above for details

selectors

Description	An object containing names CSS-based selectors identifying where in the DOM different elements can be found.
Notes	The Framework will create a DOM Binder that should be used to access the elements identified by selectors. The DOM Binder attaches a function to the component object called `locate()` which retrieves the element given the selector name.

Example Definition	`fluid.defaults("fluid.progress", {`
	`  selectors: {`
	`  displayElement: ".flc-progress",`
	`  progressBar: ".flc-progress-bar",`
	`  indicator: ".flc-progress-indicator",`
	`  label: ".flc-progress-label",`
	`  ariaElement: ".flc-progress-bar"`
	`  },`
	`  ...`
	`});`

Example Override	`var myEdit = fluid.progress(container, {`
	`  selectors: {`
	`  indicator: "div.progress-indicator",`
	`  label: "span.progress-label"`
	`  },`
	`  ...`
	`});`

See also	DOM Binder

Renderer Components

back to top Components defined with a grade of `rendererComponent` support all of the common options described above, as well as those defined below. Component developers are free to define their own additional options.

See also: Component Grades

The following options are supported by renderer components:

[model] [applier] [changeApplierOptions] [events] [listeners] [selectors] [selectorsToIgnore] [repeatingSelectors] [produceTree] [protoTree] [resources] [strings] [rendererFnOptions] [rendererOptions] [renderOnInit]

model

Description	An object containing the data model to be used by the component.
Notes	If a ChangeApplier is not provided using the `#applier` option, the Framework will create one for the provided model.

Example Definition	`fluid.defaults("fluid.pager", {`
	`  model: {`
	`  pageIndex: undefined,`
	`  pageSize: 10,`
	`  totalRange: undefined`
	`  },`
	`  ...`
	`});`

Example Override	`var myPager = fluid.pager(container, {`
	`  model: {`
	`  pageIndex: 1`
	`  },`
	`  ...`
	`});`

See also	Model Objects
	ChangeApplier API
	`#applier`
	`#changeApplierOptions`

applier

Description	A ChangeApplier object for the model provided with the `#model` option.
Notes	It is not necessary to provide an applier: By default, an applier will be created with `fluid.makeChangeApplier()`, using any options specified with `#changeApplierOptions`. This option is most commonly used to share a common ChangeApplier between components in a component tree: the `applier` option can be used to reference the ChangeApplier of another component in the tree.

Example Definition	`fluid.defaults("component.name", {`
	`  applier: "{parentComponent.applier}",`
	`  ...`
	`});`

Example Override	N/A

See also	ChangeApplier API
	`#model`
	`#changeApplierOptions`

changeApplierOptions

Description	Options that will be passed on to `fluid.makeChangeApplier()` if a ChangeApplier is not provided using the `#applier` option.
Notes	If a ChangeApplier is provided using the `#applier` option, this option will be ignored.

Example Definition	`fluid.defaults("component.name", {`
	`  model: {...},`
	`  changeApplierOptions: {`
	`  cullUnchanged: true`
	`  },`
	`  ...`
	`});`

Example Override	`var myComp = component.name(container, {`
	`  model: {...},`
	`  changeApplierOptions: {`
	`  cullUnchanged: true`
	`  },`
	`  ...`
	`});`

See also	ChangeApplier API
	`#model`
	`#applier`

events

See `fluid.eventedComponent` above for details

listeners

See `fluid.eventedComponent` above for details

selectors

Description	An object containing names CSS-based selectors identifying where in the DOM different elements can be found.
Notes	The Framework will create a DOM Binder that should be used to access the elements identified by selectors. The DOM Binder attaches a function to the component object called `locate()` which retrieves the element given the selector name.

Example Definition	`fluid.defaults("fluid.progress", {`
	`  selectors: {`
	`  displayElement: ".flc-progress",`
	`  progressBar: ".flc-progress-bar",`
	`  indicator: ".flc-progress-indicator",`
	`  label: ".flc-progress-label",`
	`  ariaElement: ".flc-progress-bar"`
	`  },`
	`  ...`
	`});`

Example Override	`var myEdit = fluid.progress(container, {`
	`  selectors: {`
	`  indicator: "div.progress-indicator",`
	`  label: "span.progress-label"`
	`  },`
	`  ...`
	`});`

See also	DOM Binder

selectorsToIgnore

Description	An array of selector names identifying elements that will be ignored by the Renderer. These elements will be displayed exactly as provided in the template, with no processing
Notes	

Example Definition	`fluid.defaults("cspace.header", {`
	`  selectors: {`
	`  menuItem: ".csc-header-menu-item",`
	`  label: ".csc-header-link",`
	`  searchBox: ".csc-header-searchBox",`
	`  logout: ".csc-header-logout",`
	`  user: ".csc-header-user",`
	`  userName: ".csc-header-userName"`
	`  },`
	`  selectorsToIgnore: ["searchBox", "logout"],`
	`  ...`
	`});`

See also	

repeatingSelectors

Description	An array of selector names identifying elements that will be repeated by the Renderer based on the data being rendered. For example, the selector for a table row that will be replicated many times should appear in the list of repeating selectors.
Notes	

Example Definition	`fluid.defaults("cspace.header", {`
	`  selectors: {`
	`  menuItem: ".csc-header-menu-item",`
	`  label: ".csc-header-link",`
	`  searchBox: ".csc-header-searchBox",`
	`  logout: ".csc-header-logout",`
	`  user: ".csc-header-user",`
	`  userName: ".csc-header-userName"`
	`  },`
	`  repeatingSelectors: ["menuItem"],`
	`  ...`
	`});`

See also	

produceTree

Description	A function that will return a Renderer Component Tree for the component.
Notes	The referenced function must accept the component object as its only parameter and return a Renderer component tree. NOTE that if both `produceTree` and `protoTree` are specified, only the `produceTree` function will be used; the `protoTree` will be ignored.

Example Definition	`cspace.confirmationDialog.produceTree = function (that) {`
	`  var tree = {`
	`  ...`
	`  };`
	`  return tree;`
	`};`
	`fluid.defaults("cspace.confirmationDialog", {`
	`  produceTree: cspace.confirmationDialog.produceTree,`
	`  ...`
	`});`

See also	`#protoTree` Renderer Component Trees

protoTree

Description	A tree of Renderer protocomponents.
Notes	NOTE that if both `#produceTree` and `protoTree` are specified, only the `produceTree` function will be used; the `protoTree` will be ignored.

Example Definition	<pre>fluid.defaults("cspace.searchTips", {
	    protoTree: {
	        searchTips: {decorators: {"addClass": "{styles}.searchTips"}},
	        title: {
	            decorators: {"addClass": "{styles}.title"},
	            messagekey: "searchTips-title"
	        },
	        expander: {
	            repeatID: "instructions",
	            type: "fluid.renderer.repeat",
	            pathAs: "row",
	            controlledBy: "messagekeys",
	            tree: {
	                messagekey: "${{row}}"
	            }
	        }
	    },
	    ...
	});</pre>

Example Override	<pre>var searchTips = cspace.searchTips(container, {
	    protoTree: {
	        searchTips: {decorators: {"addClass": "{styles}.searchTips"}},
	        title: {
	            decorators: {"addClass": "{styles}.title"},
	            messagekey: "searchTips-title"
	        },
	        expander: {
	            repeatID: "instructions",
	            type: "fluid.renderer.repeat",
	            pathAs: "row",
	            controlledBy: "messagekeys",
	            tree: {
	                messagekey: "${{row}}"
	            }
	        }
	    },
	    ...
	});</pre>

See also	`#produceTree` Renderer Component Trees ProtoComponent Types

resources

Description	An object that lists resources (such as HTML files, CSS files, data files) required by the component.
Notes	The specified resources will be loaded automatically and the file content will be stored within the resources object itself.

Example Definition	<pre>fluid.defaults("component.name", {
	    resources: {
	        headerTemplate: {
	            href: "../templates/Header.html"
	        },
	        footerTemplate: {
	            href: "../templates/Footer.html"
	        }
	    },
	    ...
	});</pre>

Example Override	`var myComp = component.name(container, {`
	`  resources: {`
	`  footerTemplate: {`
	`  href: "../templates/FrontPageFooter.html"`
	`  }`
	`  },`
	`  ...`
	`});`

See also	fluid.fetchResources

strings

Description	An object containing named strings or string templates. The strings will be used by the Renderer.
Notes	The Framework will create a Message Resolver and add it to the component object if the `strings` option is present.

Example Definition	`fluid.defaults("cspace.searchToRelateDialog", {`
	`  gradeNames: ["fluid.rendererComponent", "autoInit"],`
	`  strings: {`
	`  createNewButton: "Create",`
	`  title: "Add Related %recordType Record",`
	`  closeAlt: "close button",`
	`  relationshipType: "Select relationship type:",`
	`  createNew: "Create new record:",`
	`  addButton: "Add to current record"`
	`  },`
	`  ...`
	`});`

Example Override	`var myDialog = cspace.searchToRelateDialog(container, {`
	`  strings: {`
	`  relationshipType: "Select a relationship type from the list below:",`
	`  createNew: "Create a new record:",`
	`  addButton: "Add this record to the current record"`
	`  },`
	`  ...`
	`});`

See also	Message Resolver `fluid.messageResolver`

rendererFnOptions

Description	Options that will be passed directly to the renderer creation function, `fluid.renderer.createRendererSubcomponent`
Notes	

Example Definition	`fluid.defaults("fluid.tableOfContents.levels", {`
	`  rendererFnOptions: {`
	`  noexpand: true`
	`  },`
	`  ...`
	`});`

Example Override	`var recEditor = cspace.recordEditor(container, {`
	`  rendererFnOptions: {`
	`  rendererTargetSelector: "dialog"`
	`  },`
	`  ...`
	`});`

See also	Renderer Components `fluid.renderer.createRendererSubcomponent`

rendererOptions

Description	Options that will be included in the `#rendererFnOptions` as `rendererOptions`
Notes	

Example Definition	`fluid.defaults("cspace.searchBox", {`
	`  rendererOptions: {`
	`  autoBind: false`
	`  },`
	`  ...`
	`});`

Example Override	`var search = cspace.searchBox(container, {`
	`  rendererOptions: {`
	`  autoBind: true`
	`  },`
	`  ...`
	`});`

See also	Renderer Components `#rendererFnOptions`

renderOnInit

Description	A boolean flag indicating whether or not the component should render itself automatically once initialization has completed. By default, renderer components do not render themselves automatically.
Notes	This option is valid both for `"autoInit"` components and for components that are initialized manually, through `fluid.initRendererComponent`.

Example Definition	`fluid.defaults("cspace.login", {`
	`  gradeNames: ["fluid.rendererComponent", "autoInit"],`
	`  renderOnInit: true,`
	`  ...`
	`});`

Example Override	`var login = cspace.login(container, {`
	`  renderOnInit: false,`
	`  ...`
	`});`

See also	