Understanding Infusion Components

To help understand how a component might be designed using components, consider some of the components in the Infusion Component Library:

- **Uploader**
- **Inline Edit**
- **Infusion Progress**

### What Does A Component Look Like?

A component is a regular JavaScript object that has certain characteristics. The most simple components have a `typeName` and an `id`, but typical components that handle the different parts of your application.

### Examples

#### Examples with 'grades'

The `grades` component is a simple component representing a course's grades. It displays a list of grades along with their corresponding letters. Here's an example of how to use it:

```javascript
fluid.defaults('grades', {
  typeName: 'grades',
  id: 'my-grades-component',
  fields: [fluid.defaults('field', {name: 'grade', type: 'number', description: 'Grade'}), fluid.defaults('field', {name: 'letter', type: 'string', description: 'Letter Grade'})],
  events: {
    'change': function(grade) {
      this.letter = fluid.letterFromGrade(grade);
    }
  }
});
```

This function converts a grade to its corresponding letter grade.

### Understanding Infusion Components

Infusion components are designed to be reusable, making it easy to construct complex user interfaces. Components can be created and combined using different parts of an application, allowing for a modular and flexible development process.

#### Examples

- **Uploader**
- **Inline Edit**
- **Infusion Progress**

### Examples with 'grades'

The `grades` component is a simple component representing a course's grades. It displays a list of grades along with their corresponding letters. Here's an example of how to use it:

```javascript
fluid.defaults('grades', {
  typeName: 'grades',
  id: 'my-grades-component',
  fields: [fluid.defaults('field', {name: 'grade', type: 'number', description: 'Grade'}), fluid.defaults('field', {name: 'letter', type: 'string', description: 'Letter Grade'})],
  events: {
    'change': function(grade) {
      this.letter = fluid.letterFromGrade(grade);
    }
  }
});
```

This function converts a grade to its corresponding letter grade.

### Examples

- **Uploader**
- **Inline Edit**
- **Infusion Progress**

### Examples with 'grades'

The `grades` component is a simple component representing a course's grades. It displays a list of grades along with their corresponding letters. Here's an example of how to use it:

```javascript
fluid.defaults('grades', {
  typeName: 'grades',
  id: 'my-grades-component',
  fields: [fluid.defaults('field', {name: 'grade', type: 'number', description: 'Grade'}), fluid.defaults('field', {name: 'letter', type: 'string', description: 'Letter Grade'})],
  events: {
    'change': function(grade) {
      this.letter = fluid.letterFromGrade(grade);
    }
  }
});
```

This function converts a grade to its corresponding letter grade.

#### Examples

- **Uploader**
- **Inline Edit**
- **Infusion Progress**

### Examples with 'grades'

The `grades` component is a simple component representing a course's grades. It displays a list of grades along with their corresponding letters. Here's an example of how to use it:

```javascript
fluid.defaults('grades', {
  typeName: 'grades',
  id: 'my-grades-component',
  fields: [fluid.defaults('field', {name: 'grade', type: 'number', description: 'Grade'}), fluid.defaults('field', {name: 'letter', type: 'string', description: 'Letter Grade'})],
  events: {
    'change': function(grade) {
      this.letter = fluid.letterFromGrade(grade);
    }
  }
});
```

This function converts a grade to its corresponding letter grade.

---

**Tables and larger images can fill all available width if required.**

- **Categories:**
  - fluid
  - jQuery
  - Infusion
  - tutorial

- **License:**
  - Creative Commons Attribution-ShareAlike 4.0 International License

---

**Fluid is a project of the Inclusive Design Research Centre at OCAD University, funded by a grant from The Andrew W. Mellon Foundation.**

**API**

- **Type:**
  - Decorator
  - Decorator Type

- **Caption:**

---

**Tables and larger images can fill all available width if required.**

- **Categories:**
  - fluid
  - jQuery
  - Infusion
  - tutorial

- **License:**
  - Creative Commons Attribution-ShareAlike 4.0 International License
Understanding Infusion Components

Note: This is an example of a paragraph with emphasis. This functionality is Sneak Peek status. This means that the APIs may change.

The first paragraph in an article is styled differently. Every Infusion application is structured as a set of components. An Infusion component can represent a visible component on screen, a collection of related functionality such as an object as in object-orientation, or simply a unit of work or relationship between other components. This page provides resources to help you understand components.

If you’re creating an entire web application, your application would be implemented as a component that coordinates interactions between other components that handle the different parts of your application.

Examples

To help understand how a widget or application might be designed using components, consider some of the components in the Infusion Component Library:

Uploader

The Uploader allows users to add several files to a queue and then upload them all at once. It is actually made up of several subcomponents: It has the file queue view, which displays the files currently in the queue; it has a total progress bar at the bottom. In turn, the file queue view component has its own subcomponents.

What Does A Component Look Like?

A component is a regular JavaScript object that has certain characteristics. The most simple components have a typeName and an id, but typical components will have more:

- a creator function
- the function that implementors invoke, which returns the component object itself
- configuration options
- various values that control the operation of the component, which can be overridden by implementors to customize the component
- public functions

Depending on what the component is for, some will include infrastructure to support:

- events
- a model
- a view
- a renderer

New kinds of components are created by passing configuration information to the fluid.defaults function. This function will create the creator function that will be used to instantiate the component. The Framework provides supports for automatically creating components of various types, or 'grades'; as well, developers can create their own grades.

Examples with code

```
fluid.enhance.check({
  check1: "my.checking.function1",
  check2: "my.checking.function2",
  ...
});
```

The function fluid.enhance.check() will execute the specified functions and store the results in the static environment using the associated key (e.g. check1). The presence of the tags in the static environment can be used in the context argument to fluid.demands().

<table>
<thead>
<tr>
<th>Decorator or Type</th>
<th>Field Name</th>
<th>Field Type</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>func</td>
<td>String</td>
<td>jQuery function to be invoked</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decorator or Type</th>
<th>Field Name</th>
<th>Field Type</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>func</td>
<td>Array of Object</td>
<td>jQuery function to be invoked</td>
<td></td>
</tr>
</tbody>
</table>

Decorator

Field Name

Field Type

Field Description

Example

```
decorators: [{
  type: "jQuery",
  func: "click",
  args: function() {$(this).hide();}
}]
```

<table>
<thead>
<tr>
<th>Decorator or Type</th>
<th>Field Name</th>
<th>Field Type</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>func</td>
<td>Array of Object</td>
<td>jQuery function to be invoked</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decorator or Type</th>
<th>Field Name</th>
<th>Field Type</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>func</td>
<td>Array of Object</td>
<td>jQuery function to be invoked</td>
<td></td>
</tr>
</tbody>
</table>

Decorator

Field Name

Field Type

Field Description

Example

```
decorators: [{
  type: "jQuery",
  func: "click",
  args: function() {$(this).hide();}
}]
```

<table>
<thead>
<tr>
<th>Decorator or Type</th>
<th>Field Name</th>
<th>Field Type</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>func</td>
<td>Array of Object</td>
<td>jQuery function to be invoked</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decorator or Type</th>
<th>Field Name</th>
<th>Field Type</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>func</td>
<td>Array of Object</td>
<td>jQuery function to be invoked</td>
<td></td>
</tr>
</tbody>
</table>
Understanding Infusion Components

Examples

To help understand how to write on an application, this page documents some of the components in the Fluid Component Library:

Progress

The Progress component is a simple component with no subcomponents. It has a number of UI elements that work together and are updated programmatically to show the progress of some activity. It has a pretty simple purpose and function, one that doesn't make much sense to try to break up into multiple subcomponents.

Uploader

The Uploader allows users to add several files to a queue and then upload them all at once. It actually makes up of several subcomponents: it has the file queue view, which displays the files currently in the queue; it has a total progress bar at the bottom; in turn, the file queue view component has its own subcomponents.

What Does A Component Look Like?

A component is a regular JavaScript object that has certain characteristics. Those characteristics aren’t required to be in a component, but typical components will have it:

- creator function
- the function that implementors invoke, which returns the component object itself
- configuration options
- various values that control the operation of the component, which can be overridden by implementors to customize the component
- public functions

Depending on what the component is for, some may include infrastructure to support:

- events
- a model
- a view
- a renderer

New kinds of components are created by passing configuration information to the `fluid.defaults()` function. This function will create the creator function that will be used to instantiate the component. The Framework provides supports for automatically creating components of various types, or ‘grades’; as well, developers can create their own grades.

Examples with code

```
Fluid.framework("
my.checking.function1",
{ "my.checking.function2": null, ...
});
```

The function `Fluid.framework()` will execute the specified functions and store the results in the static environment using the associated key (e.g. `check1`). The presence of the tags in the static environment can be used in the context argument to `fluid.demands()`. Table and larger images can fill all available width if required.
Note: This is an example of a paragraph with emphasis. This functionality is Sneak Peek status. This means that the APIs may change.

The first paragraph in an article is styled differently. Every Infusion application is structured as a set of components.

An Infusion component can represent a visible component on screen, a collection of related functionality such as an object as in object-orientation, or simply a unit of work or relationship between other components. This page provides resources to help you understand components.

If you're creating an entire web application, your application would be implemented as a component that coordinates interactions between other components that handle the different parts of your application.

Examples To help understand how a widget or application might be designed using components, consider some of the components in the Infusion Component Library:

- **Progress**
  The Infusion Progress component is a single component with no subcomponents. It has a number of UI elements that work together and are updated programmatically to show the progress of some activity. It has a pretty simple purpose and function, one that doesn't make much sense to try to chunk up into multiple components.

- **Inline Edit**
  The Inline Edit component allows users to edit text in place, without switching to a new screen, by simply switching into an in-place edit mode. The view mode is implemented one way, with certain functionality (i.e. a tooltip, an affordance to edit), and the edit mode is implemented differently: it's an edit field. Conceptually, these two modes are rather different, and so they're implemented as two separate subcomponents of the main Inline Edit component.

- **Uploader**
  The Uploader allows users to add several files to a queue and then upload them all at once. It is actually made up of several subcomponents: it has the file queue view, which displays the files currently in the queue; it has a total progress bar at the bottom. In turn, the file queue view component has its own subcomponents.

What Does A Component Look Like?
A component is a regular JavaScript object that has certain characteristics. The most simple components have a `typeName` and an `id`, but typical components will have more:

- A creator function
- The function that implementors invoke, which returns the component object itself
- Configuration options
- Various values that control the operation of the component, which can be overridden by implementors to customize the component
- Public functions

Depending on what the component is for, some will include infrastructure to support:

- Events
- A model
- A view
- A renderer

New kinds of components are created by passing configuration information to the `fluid.defaults` function. This function will create the creator function that will be used to instantiate the component. The Framework provides supports for automatically creating components of various types, or 'grades'; as well, developers can create their own grades.

Examples with code
```javascript
fluid.enhance.check({
  check1: "my.checking.function1",
  check2: "my.checking.function2",
  ...
});
```

The function `fluid.enhance.check()` will execute the specified functions and store the results in the static environment using the associated key (e.g. `check1`). The presence of the tags in the static environment can be used in the context argument to `fluid.demands()`.
Understanding Infusion Components

Note: This is an example of a paragraph with emphasis. This functionality is Sneak Peek status. This means that the APIs may change.

The first paragraph in an article is styled differently. Every Infusion application is structured as a set of components. An Infusion component can represent a visible component on screen, a collection of related functionality such as an object as in object-orientation, or simply a unit of work or relationship between other components. This page provides resources to help you understand components.

If you're creating an entire web application, your application would be implemented as a component that coordinates interactions between other components that handle the different parts of your application.

Examples

To help understand how a widget or application might be designed using components,
Understanding Infusion Components

Note: This is an example of a paragraph with emphasis. This functionality is Sneak Peek status. This means that the APIs may change.

The first paragraph in an article is styled differently. Every Infusion application is structured as a set of components. An Infusion component can represent a visible component on screen, a collection of related functionality such as an object as in object-orientation, or simply a unit of work or relationship between other components. This page provides resources to help you understand components.

If you're creating an entire web application, your application would be implemented as a component that coordinates interactions between other components that handle the different parts of your application.

Examples

To help understand how a widget or application might be designed using components, consider some of the components in the
Understanding Infusion Components

**What Does A Component Look Like?**

A component is a regular JavaScript object that has certain characteristics. The simple component structure includes a `typeName` and an `id`, but typical components will have more:

- a renderer
- a view
- a model
- events

Depending on what the component is for, some will include infrastructure to:

- various values that control the operation of the component, which can be overridden by implementors to customize the component behaviour.
- the function that implementors invoke, which returns the component (or a collection of subcomponents) in the context of the component itself.

Components are configured using information that is passed to the component as a configuration object.

Infusion 1.5 Documentation

http://fluidproject.org

Objects, Arrays, String

jQuery function

Arguments to the function will execute the specified functions.

Examples with code:

```javascript
fluid.enhance.check({
    check1: "my.checking.function1",
    check2: "my.checking.function2",
});
```

Infusion is created by fluidproject.org

—

The most simple components have a `typeName` and an `id`, but typical components will have more:

- a renderer
- a view
- a model
- events

The above information can be used in a `create` function that creates the component, or it can be used in a `create` function that returns the component. This component has its own subcomponents.

Examples with code:

```javascript
var create = function(type, data) {
    return ({
        type: type,
        data: data,
    });
};
```

The `create` function returns a configuration object that can be used in a `create` function that creates the component. This component has its own subcomponents.