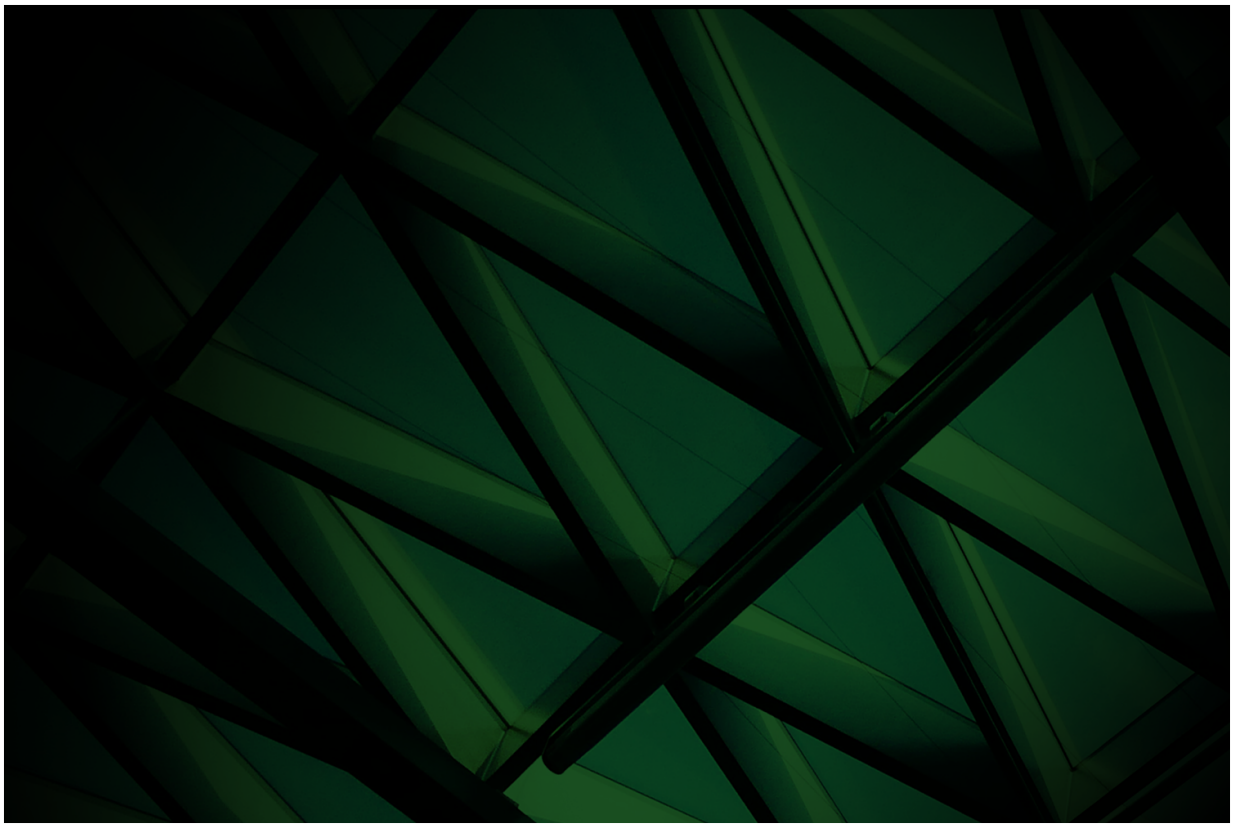


Orb Web Framework Reference

VERSION 2025.0.02

jade platform

Copyright©2026 Jade Software Corporation Limited. All rights reserved.

Jade Software Corporation Limited cannot accept any financial or other responsibilities that may be the result of your use of this information or software material, including direct, indirect, special or consequential damages, or loss of profits. There are no warranties extended or granted by this document or software material.

You should be very careful to ensure that the use of this software material and/or information complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Jade Software Corporation Limited.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Copyright © 2026 Jade Software Corporation Limited.

All rights reserved.

JADE is a trademark of Jade Software Corporation Limited. All trade names referenced are the service mark, trademark, or registered trademark of the respective manufacturer.

For details about other licensing agreements for third-party products, you must read the Jade **Readme.txt** file.

Contents

Contents ... **iii**

Before You Begin ... **v**

Who Should Read this Guide ... v
Conventions ... v
Related Documentation ... vi

Chapter 1 Orb Web Framework Reference ... **7**

Overview ... 8
Installing Orb ... 8
Defining an Orb Application ... 9
HTML Templates ... 10
Comment Directives ... 10
Data Directives ... 11
Inline-Code Directive ... 11
Internal Directives ... 12
Embedded Template Directives ... 13
Executable Directive ... 13
HTML Data Attributes ... 13
Routing Engine ... 16
Route Properties ... 17
Orb Interfaces ... 18
Cursor Interfaces ... 18
IOrbSessionCursor Interface ... 19
IOrbTemplateCursor Interface ... 19
IOrbRoute Interface ... 19
IOrbTemplate Interface ... 20
IOrbViewController Interface ... 21
IOrbErrorHandler Interface ... 22
Orb Class ... 24
Orb Methods ... 24
addDynamicTemplate ... 25
addHeader ... 26
addHtmlTemplate ... 26
addLiveRoute ... 26
addRoute ... 27
addRouteWithParameters ... 27
addStylesheet ... 27
addView ... 28
addViewWithParameters ... 28
addVirtualDirectory ... 28
bind ... 29
bindCursor ... 29
bindRaw ... 30
bindRawCursor ... 30
bindRawString ... 30
bindString ... 30
errorHandler ... 31
liveUpdate ... 31
redirect ... 31
request ... 31
Accessing Request Metadata ... 32
sessionData ... 32
sessionDataCursor ... 32
sessionEnd ... 32
sessionEndCursor ... 33
sessionStart ... 33
setContentType ... 33

Contents iv

setDebug 33
setStatusCode 34
setTitle 34
strip 34
usePresentationClient 34
view 34
wrap 34
Debugging an Orb Application 35

Before You Begin

The *Orb Web Framework Reference* is intended as the main source of information when you are developing applications that use the Orb web framework.

Who Should Read this Guide

The main audience for the *Orb Web Framework Reference* is expected to be developers of Jade applications in Jade.

Conventions

The *Orb Web Framework Reference* uses consistent typographic conventions throughout.

Convention	Description
Arrow bullet (❯❯)	Step-by-step procedures. You can complete procedural instructions by using either the mouse or the keyboard.
Bold	Items that must be typed exactly as shown. For example, if instructed to type **foreach**, type all the bold characters exactly as they are printed. File, class, primitive type, method, and property names, menu commands, and dialog controls are also shown in bold type, as well as literal values stored, tested for, and sent by Jade instructions.
Italic	Parameter values or placeholders for information that must be provided; for example, if instructed to enter *class-name*, type the actual name of the class instead of the word or words shown in italic type. Italic type also signals a new term. An explanation accompanies the italicized type. Document titles and status and error messages are also shown in italic type.
Blue text	Enables you to click anywhere on the cross-reference text (the cursor symbol changes from an open hand to a hand with the index finger extended) to take you straight to that topic. For example, click on the "Installing Orb" cross-reference to display that topic.
Bracket symbols ([ ])	Indicate optional items.
Vertical bar (|)	Separates alternative items.
`Monospaced font`	Syntax, code examples, and error and status message text.
ALL CAPITALS	Directory names, commands, and acronyms.
Small font	Keyboard shortcut keys.

Key combinations and key sequences appear as follows.

Convention	Description
Key1+Key2	Press and hold down the first key and then press the second key. For example, "press Shift+F2" means to press and hold down the Shift key and press the F2 key. Then release both keys.
Key1,Key2	Press and release the first key, then press and release the second key. For example, "press Alt+F,X" means to hold down the Alt key, press the F key, and then release both keys before pressing and releasing the X key.

Related Documentation

Other documents that are referred to in this guide, or that may be helpful, are listed in the following table, with an indication of the Jade operation or tasks to which they relate.

Title	Related to…
Database Administration Guide	Administering Jade databases
Development Environment User's Guide	Using the Jade Platform development environment
Developer's Reference	Developing or maintaining Jade applications
Encyclopaedia of Classes	System classes (Volumes 1 and 2), Window classes (Volume 3)
Encyclopaedia of Primitive Types	Primitive types and global constants
External interface Developer's Reference	Developing Jade applications using external interfaces
Object Manager Guide	Jade Object Manager administration
Web Application Guide	Implementing, monitoring, and configuring web applications

Jade Direct REST documentation can provide some useful guidance. For details, see:

- "REST-Based Web Services", in Chapter 11 of the *Developer's Reference*
- *Direct REST Reference Architecture White Paper*

Note PDF documents in the Jade product information library should all be located in the same folder. If this document is not located in the same folder as other Jade product information documents, the cross-document hyperlinks do not work, nor does the PDF search index catalog.

However, the online HTML5 format of the full product information library contains all documents including white papers, so the HTML5 search functionality searches all of the product information library and cross-document hyperlinks between documents in the HTML5 format work as expected.

Access the HTML5 online help in your browser at:

https://secure.jadeworld.com/developer-centre/Jade2025/OnlineDocumentation/

Chapter 1 Orb Web Framework Reference

This document covers the following topics.

- Overview
- Installing Orb
- Defining an Orb Application
- HTML Templates
  - Comment Directives
  - Data Directives
  - Inline-Code Directive
  - Internal Directives
  - Embedded Template Directives
  - Executable Directive
  - HTML Data Attributes
- Routing Engine
  - Route Properties
- Orb Interfaces
  - Cursor Interfaces
  - IOrbRoute Interface
  - IOrbTemplate Interface
  - IOrbViewController Interface
  - IOrbErrorHandler Interface
- Orb Class
  - Orb Methods
- Debugging an Orb Application

Overview

Orb is an HTML-over-the-wire web framework that is native to the Jade Platform. It enables you to create rich web-based applications and provides the features of a modern web framework such as session handling and templating.

You can design and create HTML layouts and style sheets outside of Jade. The Orb web framework then renders that HTML to web clients after populating it with data from objects or collections in your Jade system.

Web applications using this framework can contain live elements that automatically update based on changes to objects or collections in your Jade system.

Orb uses Direct HTTP only; there is no need to configure Internet Information Server (IIS). Orb applications use the HTTP/1.1 communication protocol, which allows a greater degree of freedom for the deployment of Orb applications and it also reduces the system requirements for the development of Orb applications.

Note You must import the **Orb** class and methods into your schema from the **JadeOrbPackage**. For details, see "Installing Orb", elsewhere in this document.

The methods provided by the **Orb** class define the behavior of the Orb web framework through the following components.

- HTML Templates, which control the layout and what is displayed in your web pages
- Routing Engine, which directs web pages to the location of specified assets
- Orb Interfaces, which control the views, error handling, dynamic templates, and actioning web requests

For details about the methods provided by the **Orb** class, see "Orb Class" and "Orb Methods", elsewhere in this document.

The Orb web framework release 1.0.03 is compatible with Jade releases 2025.0.01 and higher.

Installing Orb

The Orb web framework is available as an optional extension to the Jade platform.

To get the Orb components, download the zip file from the Orb tile on the **Extensions** page of the Jade Platform Developer Centre; that is, https://www.jadeplatform.com/developer-centre/extensions.

After you unzip the downloaded file:

1. Load the **JadeOrbSchema** files into your system. This schema has the **JadeOrbPackage** package, which contains the **Orb** class, methods, interfaces that you require to create an Orb application. For details, see "Loading Your Schema", in Chapter 10 of the *Development Environment User's Guide*.
2. Import the **JadeOrbPackage** package from the **JadeOrbSchema** into the schema for which you want to create an Orb application. For details, see "Importing a Package", in Chapter 8 of the *Developer's Reference*.

Sample schemas with Orb applications can be downloaded from the public repository at:

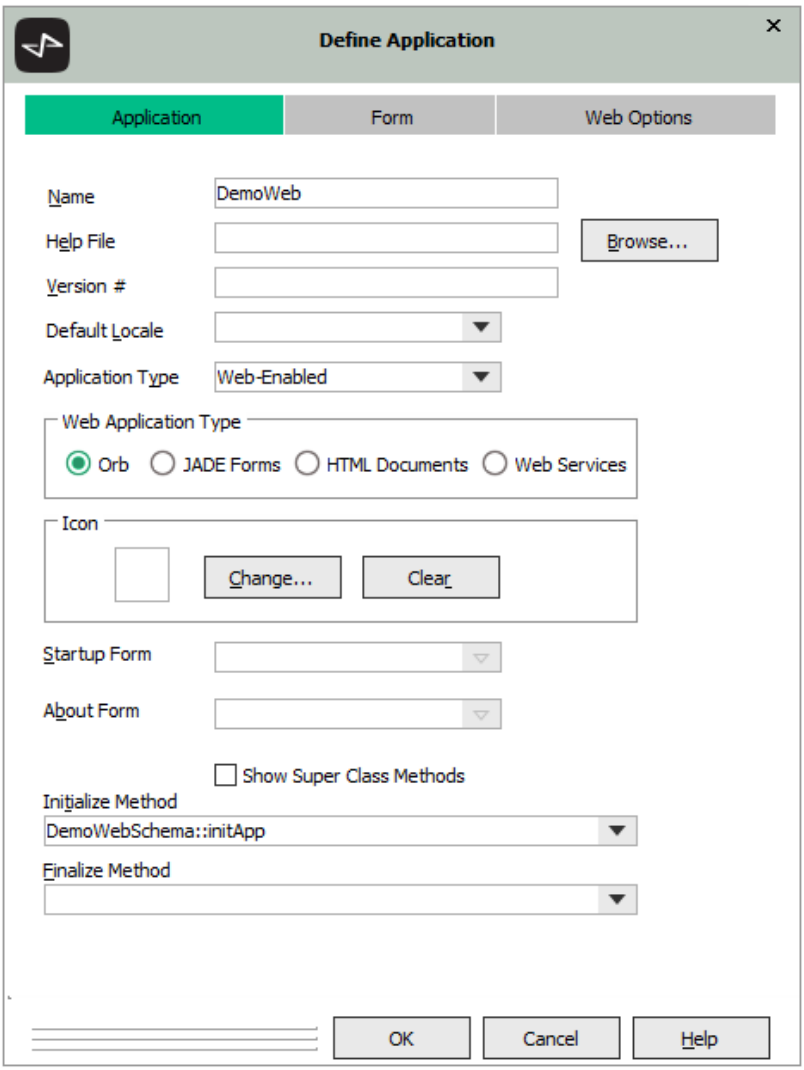
https://github.com/jadesoftwarenz/jade-orb-examples

Defining an Orb Application

To create an Orb application, you must first obtain the **JadeOrbSchema** and then load it into your Jade system. After **JadeOrbSchema** loads into your system, you can then import the **JadeOrbPackage** into your schema. For details, see "Installing Orb", elsewhere in this document.

An Orb application is defined from the Define Application dialog in the standard way. For more details, see "Defining Applications", in Chapter 3 of the *Development Environment User's Guide*.

On the Define Application dialog **Application** sheet, select the **Web-Enabled** or **Web-Enabled, Non-Gui** application type and then select the **Orb** web application type, as shown in the following image.



After selecting Orb, configure your application connection details on the **Web Options** sheet. The connection name should be the host and port on which your application will listen; for example, setting the connection name to **localhost:8080** will make your application accessible on **http://localhost:8080/**.

In a production environment, this value should be the host and port to which your gateway will forward web requests.

HTML Templates

Orb provides a templating engine that renders content based on provided assets that are expanded with data from an object. This typically would be used to produce rendered HTML to send to a web client.

Your templates can contain any arbitrary HTML (or any other textual content).

To include data from your Jade database in the HTML, you can decorate your templates with special directives that Orb dynamically expands. At runtime, the directive is replaced with the corresponding data.

When using directives to expand the templates with data from your system, an expansion object must be provided, which is the object from which the data is to be derived. The expansion object is provided as a parameter to the **Orb@bind** call or, in the case of embedded templates, it is a reference from the current expansion object that is passed to the embedded template, using an embedded template directive as summarized in the following table.

Directive Type	HTML Element	Description
Comment	<## ##>	Identifies comment text that is not expanded
Data	<@@ @@>	Expands to an object property
Inline-code	<%{ }%>	Runs enclosed Jade code
Internal	<@@ orb-*xxx* @@>	Special Orb directives
Executable	<%% %%>	Expands to the result of a method call
Embedded	<$$ $$>	Inserts another template inside the current template

You can decorate your templates with the following special directives that Orb will dynamically expand.

- Comment directives
- Data directives
- Inline-code directive
- Internal directives
- Embedded template directives
- Executable directive

Comment Directives

Any text wrapped in **<## ##>** is treated as a comment and is removed from the template before any further processing occurs.

The following example shows the format of a comment directive.

```
<## text ##>
```

Unlike standard HTML comments, Orb template comments are not sent to the client and will not be visible in the browser HTML source.

Other directives can be commented out by being wrapped in comment tags and are not expanded when the template is processed.

All of the following examples show valid comments and produce no output.

- Simple comment

```
<## a single line comment ##>
```

- Comments can be multiple lines

```
<## a
comment that
spans multiple
lines
##>
```

- Directive commented out

```
<## <@@ name @@> ##>
```

- A space is not required between the tags

```
<####>
```

Data Directives

Directives wrapped in **<@@ @@>** specify a property on the expansion object. The directive is replaced with the string representation of the value of the specified property. The following shows an example of the format of a data directive.

```
<@@ name @@>
```

If the expansion object has no property with the specified name, the directive is expanded to an empty string. For example, a template could contain the following:

```
<div class="item">
    <h1><@@ title @@></h1>
    <p><@@ summary @@></p>
    <p>Amount: <@@ count @@></p>
</div>
```

An expansion object could have the following properties.

```
title: "Hello world"
summary: "This is a test"
count: 42
```

Binding the previous template to the expansion object produces the following result.

```
<div class="item">
    <h1>Hello world</h1>
    <p>This is a test</p>
    <p>Amount: 42</p>
</div>
```

Inline-Code Directive

Jade code enclosed inside **<%{ }%>** tags is executed when the template is rendered. The enclosed code must return a value, and the **String** representation of that value is included in the rendered output of the template.

The **Inline-code** directive is intended to be used for including HTML boolean attributes and for dynamically determining HTML class names without needing to add a method to the system to do this.

The following example shows the use of inline-code to set the **"selected"** attribute on an HTML **<option>** tag.

```
<option name="objectProperty" <%{ if propertyToCompare then return "selected";
endif; }%>>Option</option>
```

The following table compares the different outputs when the value of the **propertyToCompare** property is **true** or **false**.

Value	Output
true	`<option name="objectProperty" selected>Option</option>`
false	`<option name="objectProperty">Option</option>`

The enclosed code must be valid Jade code (including semicolons) and the code must return a value. If the code cannot be compiled, a runtime exception is raised when rendering the template.

Orb compiles the code against the type of the object by which the template is being expanded, which allows the code to access properties (and protected properties) on the class. The enclosed code is treated as a non-updating method; if the code attempts to modify a property it will fail to compile when rendering.

Caution Ensure that user input is never used in a way that could allow it to be interpreted as inline code.

Internal Directives

Internal directives are similar to data directives, except that the specified name starts with the **orb-** prefix (which is not valid in property names). For details about data directives, see "Data Directives", earlier in this document.

The following example shows the format of an internal directive.

```
<@@ orb-identifier @@>
```

Valid internal directives are as follows.

- **<@@ orb-id @@>**

  The **orb-id** directive is expanded to an identifier that is guaranteed to uniquely represent the current expansion object. This value can be safely used as an HTML **id** attribute or the **data-orb-id** attribute (or both).

  To enable removing items from a page with live events, the **data-orb-id** attribute must be set to the expansion of **<@@ orb-id @@>**; for example:

  ```
  <div id="<@@ orb-id @@>" data-orb-id="<@@ orb-id @@>"> ... </div>
  ```

  Note The expanded value of the **orb-id** directive is an implementation detail and is subject to change.

- **<@@ orb-headers @@>**

  The **orb-headers** directive must be placed inside the **<head>** element of the HTML page. This directive expands HTML, which includes default assets, into the web page.

  The default assets include the default Orb scripts and any style sheets added with the **addStylesheet** method of the **Orb** class.

  If this directive is not used, functionality provided by **data-orb-*** HTML data attributes (such as live updates and triggers) will not function.

Embedded Template Directives

Directives wrapped in **<$$ $$>** reference an embedded template. The embedded template is expanded when the containing template is rendered by the Orb templating engine.

The following shows an example of the format of an embedded template directive.

```
<$$ template_name [reference-name] $$>
```

Optionally, the name of a reference can be specified. If the reference is:

- A collection, each member of the collection is expanded using the specified template. Specifically, the template is expanded for each member of the collection with each member of the collection becoming the expansion object in turn.
- An object, the specified template is expanded with the specified object used as the expansion object.

If no reference is specified, the specified template is expanded with the current expansion object remaining as the expansion object.

Executable Directive

Directives wrapped in **<%% %%>** reference a method.

The following shows an example of the format of an executable directive.

```
<%% method_name [parameter] %%>
```

The method with the specified name is executed with the expansion object as the receiver. The directive is replaced with the string representation of the result of executing the method.

The specified method must take exactly zero (**0**) parameters, or a single string parameter. If the method takes one parameter and no parameter is specified, null is passed.

If the method expects any other parameters, a runtime exception is raised.

All other directives are expanded before executable directives. The result of expanding a prior directive can therefore be passed as the parameter.

The following example calls the **someMeth** method on the expansion object and the String value of the **foo** property is passed as the parameter.

```
<%% someMeth <@@ foo @@> %%>
```

HTML Data Attributes

HTML5 allows applications and frameworks to define custom **data** attributes. These are HTML attributes prefixed with **data-** that enable special behavior, which is unique to the application or framework.

Orb web framework defines the following data attributes.

- **data-orb-live="[path]"**

  The **data-orb-live** attribute specifies the live container attribute.

  The HTML element on which this attribute is specified becomes a live element. When the web page is loaded, the web browser subscribes to live updates from the endpoint specified by the **path** value.

A corresponding live route must be defined in your application, by using the **addLiveRoute** or **liveUpdate** method of the **Orb** class.

The following example declares a ** ** container as a live container.

```
<div data-orb-live="chat"></div>
```

The element in the previous example subscribes to updates by accessing the **/chat** endpoint and HTML elements are added, removed, or modified in the live ** ** container.

- **data-orb-live-scroll="false"**

  By default, when a new element is added to a web page by the **data-orb-live** attribute, the page automatically scrolls to focus on that element. This behavior can be disabled by setting the **data-orb-live-scroll** attribute to **false**.

- **data-orb-trigger="[path]"**

  The **data-orb-trigger** attribute is the trigger attribute.

  The attached element acts as a trigger. When a user clicks it or taps it in the browser, a **POST** request is made to the endpoint specified by the **path** value.

  If the target route responds with any HTML containers with an HTML id attribute that matches any existing elements, those elements are replaced.

  Note **<select>** elements can have a **data-orb-trigger** attribute that automatically triggers when a different option is selected.

- **data-orb-trigger-[name]="[value]"**

  The **data-orb-trigger-[name]** attribute allows specifying additional **POST** data to send with the trigger request. The specified **name** value is used as the field identifier.

  The following example creates a ** ** containing the text **"Todo list"** which, when clicked, sends a **POST** request to the **/requestNote** endpoint with the **POST** data **noteid=42&prevnote=93**.

  ```
  <div data-orb-trigger="/requestNote" data-orb-trigger-noteid="42"
  data-orb-trigger-prevnote="93">Todo list</div>
  ```

- **data-orb-submit="[path]"**

  The **data-orb-submit** attribute is the submit attribute. It should be declared on a **<form>** element.

  When a submit button associated with the **<form>** is clicked, a **POST** request is made to the endpoint specified by the **path** value.

  In all other respects, the **data-orb-submit** attribute uses the same semantics as the **data-orb-trigger** attribute.

- **data-orb-post="clear"**

  The **data-orb-post** attribute defines the behavior of an attribute after a submit action has occurred. If set to **"clear"**, the attached element removes any input after the submit action.

  In the following example, when the submit button is clicked, a **POST** request is made to the **/send** endpoint and any text entered in the **"message"** field is automatically removed.

  ```
  <form data-orb-submit="send">
      <input type="text" name="message" data-orb-post="clear">
      <input type="submit" value="Send message">
  </form>
  ```

- **data-orb-input="[name]"**

  The **data-orb-input** attribute allows specifying a form data name for an element that does not allow a name attribute to be declared, and causes the contents of that element to be sent as **POST** data when a form using the **data-orb-submit** attribute is submitted.

  The specified **name** is used as the field identifier in the **POST** data; for example, this can be used to allow submitting the user-entered text of a ** ** container that is declared **contenteditable**, when typically only **<input>** fields and **<textarea>** elements would be submitted.

- **data-orb-navigate="[path]"**

  The **data-orb-navigate** attribute specifies the URL that is requested when interacting with an Orb View element. For details, see "IOrbViewController Interface", elsewhere in this document.

- **data-orb-hook="[name]"**

  Orb Hooks allow interoperability between Orb and external JavaScript. Hooks must be defined as JavaScript functions, assigned as properties of the global **Orb.hooks** object.

  Whenever an element that carries a **data-orb-hook** attribute is added to the page through an Orb trigger or live update, the hook with the corresponding name is executed.

  The following example defines a hook with the name **exampleHook**. You can use this hook with the attribute **data-orb-hook="exampleHook"**.

  ```
  Orb.hooks.exampleHook = () => {
      /* code goes here */
  }
  ```

  A hook function can execute arbitrary JavaScript, and call other functions and methods.

  A hook function can also declare an optional parameter, which represents the element on which the **data-orb-hook** attribute was defined.

  ```
  Orb.hooks.exampleHook = (elem) => {
      /* the element can be accessed via the elem variable here */
  }
  ```

- **data-orb-id="[identifier]"**

  The **data-orb-id** attribute provides an identifier for an element, for the purpose of updating or removing it with live updates.

  An element with the same **data-orb-id** will replace any existing elements with the same **data-orb-id**.

  It is recommended that the value of the **data-orb-id** be assigned to the value of **<@@ orb-id @@>**. For elements to be removed with live updates, the value of **data-orb-id** must be assigned to the value of **<@@ orb-id @@>**; that is:

  ```
  data-orb-id="<@@ orb-id @@>"
  ```

Routing Engine

Orb provides a routing engine that enables associating a class with a path to dynamically respond to web requests using Jade methods. The class must implement the **IOrbRoute** interface, which specifies the **execute** and **render** methods (for details, see "IOrbRoute Interface", elsewhere in this document).

When a request is received by an Orb application, if there is a class associated with the HTTP request method and path, a transient instance of that class is created. Properties of the class are populated with data from the query string or HTTP **POST** data. The **execute** method on the instance is called, and the Integer result is passed to the **render** method. The Binary result of the **render** method is sent to the requesting web browser. After the request is finished, the transient instance of the route class is deleted.

Orb API methods can be called in the **execute** and **render** methods to retrieve information about the request, or to affect the response that is sent to the requesting browser.

The **execute** and **render** methods can also perform any operations or logic relevant to your system.

The return value of the **render** method is typically the result of expanding a template, with the **bind** method and related methods.

A route can be associated with a class using the **addRoute** method, for example:

```
addRoute("GET", "/", MainRoute);
```

This adds a route that responds to **GET** requests for the root "/" path. When a browser requests this path, an instance of **MainRoute** is created, its **execute** method called, the **render** method called, the result sent to the browser, and then the instance is deleted.

The following is another example of using the **addRoute** method to associate a route with a class.

```
addRoute("GET", "/accounts", AccountsRoute);
```

The methods on **AccountsRoute** could be as follows.

```
execute(): Integer updating;
vars
   user : UserAccount;
begin
   if not Orb@sessionData(user) then
       return Failure;
   endif;

   self.userAccount := user;
   return Success;
end;

render(executeResult: Integer): Binary;
begin
   switch executeResult do
      case Success:   return Orb@bind("accounts", self.userAccount);
      case Failure:   return Orb@redirect("/login");
      default:        return Orb@bind("error", null);
   endswitch;
end;
```

In this example, when the **/accounts** path is requested by a browser, an instance of the **AccountsRoute** class is created.

The **execute** method is called, it checks if the requesting browser has an active session, and if not, it returns a constant value representing **Failure**. Otherwise, it saves a reference to the session data object to a reference on the **AccountsRoute** instance, and returns a constant value representing **Success**.

The result returned by the **execute** method is passed to the **render** method. If the result is:

- **Success**, the **render** method returns the HTML template "**accounts**" expanded with the **UserAccount** instance that was retrieved and saved in the **execute** method.
- **Failure** (because there was no active session), the browser is told to redirect to the **login** page. This demonstrates using results returned by the **execute** method to determine the behavior of the **render** method.

Route Properties

Properties on route classes are populated with data from the URL query string for **GET** requests, and from form data for **POST** requests. Properties are populated based on parameters or input fields with matching names.

Only public properties are populated, which allows you to safely define protected and read-only properties on your class without the potential for web requests to set their values.

Note Web clients can send any values in HTTP requests. Just because your HTML form does not define an input field with a specified name does not mean a web client cannot forge a request with arbitrary field names. You should expect that any public property on a route class can be overwritten by the web client.

The following attribute types are supported.

- **Binary**. The attribute is directly populated by the raw binary received from the web client.
- **Boolean**. This attribute is **true** if any value (including an empty string and any other possible value) is sent with this name. This attribute is only ever **false** if no field with a matching name is present in the request. It is expected **Boolean** properties to be used only to represent radio buttons and check boxes and to be **true** if the radio button or check box is selected; otherwise it is **false**.
- **Date**. The sent string value is converted to a **Date** type. An HTML **<input type="date">** sends a formatted string that is represented correctly in Jade. If the sent value cannot be converted to a valid **Date**, the value is equivalent to null.
- **Decimal**. The sent string value is converted to a **Decimal**.
- **Integer**. The UTF-8 binary received from the web client is converted into a native **String** and then converted to an **Integer**. The semantics of this conversion are the same as casting a **String** type to an **Integer**.
- **Integer64**. The UTF-8 binary received from the web client is converted into a native **String** and then converted to an **Integer64**. The semantics of this conversion are the same as casting a **String** type to an **Integer64**.
- **Real**. The sent string value is converted to a **Real**.
- **String**. The UTF-8 binary received from the web client is converted into a native **String** type.
- **StringUtf8**. The UTF-8 binary received from the web client is converted into a **StringUtf8** type.
- **Time**. The sent string value is converted to a **Time** type. An HTML **<input type="time">** sends a formatted string that is represented correctly in Jade. If the sent value cannot be converted to a valid **Time**, the value is equivalent to null.
- **TimeStamp**. The sent string value is parsed and converted to a **TimeStamp** type. An HTML **<input type="datetime-local">** sends a formatted string that can be parsed and converted correctly. If the sent value cannot be parsed and converted to a valid **TimeStamp**, the value is equivalent to null.

- **StringArray**. The UTF-8 binary receives from the web client is converted to a native **String** type, and the value is added to the array. Multiple parameters or input fields with the same name add additional entries to the array. The **StringArray** must be an embedded attribute on the route class, not an object reference.
- **StringUtf8Array**. Same as **StringArray** but the conversion is to a **StringUtf8**.

Special handling for file input fields

If form data contains file information, such as from an HTML **<input type="file">**, a temporary file representing this file is created. A **String** property is populated with a string in the following format.

```
original_filename.ext;C:\path\to\temporary\file
```

For example, **somename.png;C:\temp\35283f1f-d4d2-456a-b141-ea50054e4e10**.

This string can be parsed by the application by treating everything up to the first semicolon character (**;**) as the file name that was sent by the request, and everything following as the path to the temporary file.

The temporary file is deleted after the HTTP request ends. The application must read or copy the file during the request to retain it.

A **StringArray** property is populated with multiple of these file-format strings, in the case of an upload containing multiple files.

Orb Interfaces

Orb provides the following interfaces that can be implemented on user classes.

- IOrbRoute, which allows a class to be used to action web requests.
- IOrbTemplate, which is the dynamic template interface.
- IOrbViewController, which controls Orb application views.
- IOrbErrorHandler, which handles Orb application errors.

For details about the **IOrbRequestLens** interface, see "Accessing Request Metadata", elsewhere in this document.

Orb also provides special *Cursor* interfaces, which act as a reference to an opaque object. For details, see "Cursor Interfaces", elsewhere in this document.

Cursor Interfaces

Orb also provides special *Cursor* interfaces, which act as a reference to an opaque object. Multiple Orb methods return cursor interface types, allowing these references to be saved so they can be passed back to Orb later, but without exposing the internal functionality.

None of the cursor interfaces provide any methods.

Implementing these interfaces on user classes is not supported.

Orb provides the following cursor interfaces.

- IOrbSessionCursor
- IOrbTemplateCursor

IOrbSessionCursor Interface

The **IOrbSessionCursor** cursor interface is an opaque reference to an Orb session. When a session is started by the **sessionStart** method, an **IOrbSessionCursor** interface representing the newly-created session is returned. This reference allows operations on sessions to occur out-of-band from requests where the session is active; for example, this could be used to provide a mechanism for users to see all of the locations or browsers logged in and end those sessions.

An **IOrbSessionCursor** can be passed to the following Orb methods.

- **sessionDataCursor**
- **sessionEndCursor**

IOrbTemplateCursor Interface

The **IOrbTemplateCursor** cursor interface is an opaque reference to an Orb template. When a template is added by the **addHtmlTemplate** or **addDynamicTemplate** method, an **IOrbTemplateCursor** interface referencing the added template is returned. This reference can be used as a type-safe way of referencing the template as an option in place of using the String-typed name.

An **IOrbTemplateCursor** interface can be passed to the following Orb methods.

- **bindCursor**
- **bindRawCursor**

IOrbRoute Interface

The **IOrbRoute** interface allows a class to be used to action web requests (for details, see "Routing Engine", elsewhere in this document).

The **IOrbRoute** interface has the following methods.

- **execute**, which has the following signature.

  ```
  execute(): Integer;
  ```

  The **execute** method of the class is called when a URL associated with the class implementing the **IOrbRoute** interface is accessed.

  The **execute** method is called after properties on the instance of the route class are populated and before the **render** method is called.

  The method returns an Integer. The significance of this return value is user-defined; for example, you can return a value indicating success or failure. The result returned by this method is passed to the **render** method.

- **render**, which has the following signature.

  ```
  render(executeResult: Integer): Binary;
  ```

  The **render** method is called after the **execute** method returns.

  The value returned by the **execute** method is passed to the **render** method as the **executeResult** parameter.

  The **Binary** value returned by this method is sent to the web browser; for example, the return value of this method can be the result of a call to the **bind** method.

IOrbTemplate Interface

The **IOrbTemplate** interface is the dynamic template interface, which specifies the **render** method, which has the following signature.

```
render(object: Object): Binary;
```

A class that implements this interface can function as a dynamic template, which can be defined using the **addDynamicTemplate** method.

When the Orb templating engine would expand an object with a dynamic template, the **render** method on the associated class is invoked. The expansion object is passed as the **object** parameter of the **render** method.

The **Binary** value returned by the **render** method becomes the result of expanding the dynamic template; for example, the result of the **render** method could be the result of a **bind** method on an HTML template (or another dynamic template).

Note Take care not to cause recursive dynamic template expansions.

The following is an example of a dynamic template **render** method.

```
render(object: Object): Binary;
vars
   item : AgendaItem;
   user : UserAccount;
   info : RatingInfo;
   lite : Boolean;

begin
   item := object.AgendaItem;

   if not Orb@sessionData(user) then
      return Orb@wrap('');
   endif;

   lite := item.isLite();

// lite items can't be rated
   if not lite and user <> null then
      info := item.allRatingsByUser[user];
      if info <> null then
         return Orb@bind("agenda/item/rated", item);
      endif;
   endif;

   if not lite then
      return Orb@bind("agenda/item", item);
   else
      return Orb@bind("agenda/item/lite", item);
   endif;
end;
```

When this template is expanded, the **render** method is passed the expansion object, which should be an instance of **AgendaItem** and the result will be one of the following. The user:

- Does not have an active session, in which case this template expands to an empty string.
- Has an active session and has rated the current **AgendaItem**, in which case this template expands to the HTML template named "**agenda/item/rated**".

- Has not rated the current agenda item, in which case the template expands to the HTML template named "**agenda/item**" or "**agenda/item/lite**".

This mechanism of using a dynamic template to select between multiple, different HTML templates is a typical use of this feature.

IOrbViewController Interface

Orb Views are a sub-framework of the Orb framework, which is designed to streamline the development of an application that can be defined as a series of *views* (also known as pages, screens, displays, and similar in other frameworks).

Views are controlled through a view controller, which is an instance of a class that implements the **IOrbViewController** interface.

The **IOrbViewController** interface specifies the following methods.

- **full**, which has the following signature.

  ```
  full(): Binary;
  ```

  The **full** method is called when Orb needs to perform a full page load to display the requested view.

  This method should return a complete rendering of the page.

- **partial**, which has the following signature.

  ```
  partial(): Binary;
  ```

  The **partial** method is called when Orb is responding to a navigation request, which does not need a full page load.

  This method should return any HTML elements on the page that need to be updated, without the content.

- **content**, which has the following signature.

  ```
  content(): Binary;
  ```

  The **content** method is called when Orb is responding to a navigation request, which does not need a full page load.

  This method should return the content of the page.

  This method is independent from the **partial** method to allow the content and the user interface display to be kept separate.

To use views, in the:

1. **execute** or **render** method of a route, create or retrieve an existing view controller.
2. **render** method of the route, return the result of the **view** method when passed the instance of the view controller.

Routes that respond to views must be added using the **addView** method. For details, see "IOrbRoute Interface" and "Routing Engine", elsewhere in this document.

In your HTML templates, specify the **data-orb-navigate** data attribute on elements to make a request to a URL when the user interacts with that element (the value of the attribute specifies the URL to which the request is made); for example:

```
<div data-orb-navigate="login">Login!</div>
```

When this element is clicked or tapped on the web page, a request is made to the **/login** route. As this is a navigation request, Orb responds with the result of calling the **partial** and **content** methods on the view controller. If the **/login** route is directly requested by the web browser, Orb responds with the result of the **full** method.

It is expected that the route class that responds to the **/login** request sets up an instance of the view controller class appropriately and calls the **view** method.

If the element is an HTML anchor (**<a>** tag), it is recommended to also specify the **href** attribute to maintain normal link behavior; for example:

```
<a href="login" data-orb-navigate="login">Login!</a>
```

When using an **<a>** tag, it is also possible to omit the value or specify the value as **"href"**, in which case the navigation target is considered to be the same as the **href** attribute, as shown in the following examples:

```
<a href="login" data-orb-navigate>Login!</a>

<a href="login" data-orb-navigate="href">Login!</a>
```

IOrbErrorHandler Interface

The **IOrbErrorHandler** interface specifies the **httpError** method, which has the following signature.

```
httpError(statusCode: Integer;
          phrase:     String;
          httpMeth:   String;
          httpUrl:    String;
          httpBody:   Binary;
          except:     Exception): Binary;
```

If Orb encounters an error while servicing a web request and an error handler object has been provided with the **errorHandler** method, the **httpError** method on the object is invoked.

The following table lists the parameters of the **httpError** method.

Parameter	Description
statusCode	The HTTP status code that represents the error; for example, if the error is that the requested route is not found, the status code is **404**. If the error is that an internal exception occurred, this parameter is **500**, and so on.
phrase	The reason phrase associated with the HTTP status code; for example, if the **statusCode** parameter value is **404**, the value of this parameter is "**Not Found**". This parameter is provided for convenience so that the description of the error can be sent in the response, if required.
httpMeth	The string representation of the HTTP request method; for example, "**GET**" or "**POST**".
httpUrl	The URL of the requested route, including the query string, if any; for example, the query string could be "**/account?id=42**". To aid debugging, the value of this parameter is the complete path sent by the browser; that is, it includes any prefix removed by the **strip** method. If the error is a 404 error resulting from a request to a virtual directory path, the value of this parameter is the physical path of the file that was requested.
httpBody	The **POST** body contained in the HTTP request represented in query string format. The value of this parameter is null if the request did not send **POST** data. This parameter is provided for convenience; for example, for logging or debugging.

Parameter	Description
except	If the error is the result of an exception, the value of this parameter is the instance of the exception that occurred. If the error is not the result of an exception, the value of this parameter is null.

The **Binary** value the method returns is sent to the browser; for example, this could be the result of a **bind** method call.

If the exception raised is the result of a TCP/IP connection error, this method is called but the response is not sent to the browser.

The following method shows an example of using the **bind** method to return different templates based on the error.

```
httpError(statusCode: Integer;
          phrase:     String;
          httpMeth:   String;
          httpUrl:    String;
          httpBody:   Binary;
          except:     Exception): Binary;

begin
   // this could be a method that saves info about the request
   logError(statusCode, httpMeth, httpUrl, httpBody, except);

   // display a custom error page for common error codes
   if statusCode = 404 then
      return Orb@bind("error404", null);
   elseif statusCode = 403 then
      return Orb@bind("error403", null);
   elseif statusCode = 500 then
      return Orb@bind("error500", null);
   endif;

   // display a generic error page for any other errors
   return Orb@bind("genericError", null);
end;
```

Take care exposing exception details in your error response. Exceptions are internal to your application and can contain potentially sensitive information. The error information you return should be helpful information for the user.

Orb Class

The **Orb** class is the transient class that defines the behavior of the Orb web framework. You must import the **Orb** class and methods into your schema from the **JadeOrbPackage**. For details about obtaining the **Orb** class and methods, see "Installing Orb", elsewhere in this document.

For details about the methods defined in the **Orb** class, see "Orb Methods", in the following subsection.

Inherits From: Object

Inherited By: (None)

Orb Methods

The methods defined in the **Orb** class are summarized in the following table.

Method	Description
addDynamicTemplate	Creates an instance of the specified class and associates it with the specified name in the Orb templating engine
addHeader	Adds an additional header to the request
addHtmlTemplate	Loads the content at the specified path into the Orb templating engine and associates it with the specified template
addLiveRoute	Specifies a route that actions global live updates
addRoute	Adds the specified class as a route with the specified HTTP method and URL path
addRouteWithParameters	Adds the specified class as a route with the specified HTTP method and URL path, and specifies additional optional parameters that are passed in the URL path rather than the query string
addStylesheet	Loads cascading style sheet (CSS) content located at the specified path into the Orb templating engine and associates it with the specified template
addView	Adds the specified class as a route with the **GET** and **POST** HTTP methods and URL path
addViewWithParameters	Adds the specified class as a route with the **GET** and **POST** HTTP method and URL path, and specifies additional optional parameters that is passed in the URL path rather than the query string
addVirtualDirectory	Adds the specified directory as a virtual directory that is accessible from URLs starting with the specified path
bind	Invokes the Orb templating engine to expand the specified template with the specified object
bindCursor	Invokes the Orb templating engine to bind the **IOrbTemplateCursor** specified by the **cursor** parameter with the specified object
bindRaw	Invokes the Orb templating engine to expand the specified template using the specified object (characters are not sanitized, however)
bindRawCursor	Invokes the Orb templating engine to bind the **IOrbTemplateCursor** specified by the **cursor** parameter with the specified object (characters are not sanitized, however)

Method	Description
bindRawString	Invokes the Orb templating engine to expand the specified template content with the specified object (characters are not sanitized, however)
bindString	Invokes the Orb templating engine to expand the specified template content with the specified object
errorHandler	Specifies an object that implements the **IOrbErrorHandler** interface that is invoked to handle errors
liveUpdate	Starts live updates on the specified target (which can be any object or collection)
redirect	Instructs the requesting web browser to navigate to the specified path
request	Returns an opaque object that implements a new **IOrbRequestLens** interface
sessionData	Retrieves the session data object associated with the current session
sessionDataCursor	Retrieves the session data object associated with the referenced session
sessionEnd	Terminates the active session and instructs the web browser to remove any cookies associated with it
sessionEndCursor	Terminates the referenced session
sessionStart	Starts a persistent session
setContentType	Overrides the standard **Content-Type** header for the request with the specified value
setDebug	Enables debug mode
setStatusCode	Sets the HTTP status code of the response
setTitle	Updates the title of a web page when called from an Orb trigger
strip	Strips the specified prefix from any incoming requests when determining to which route to dispatch it
usePresentationClient	Specifies whether Orb opens template and virtual directory files from the file system of the application server or the presentation client when running in thin client mode
view	Responds to a request with an Orb view
wrap	Converts the specified value to a **Binary** representation suitable for being sent to the requesting web client to display

addDynamicTemplate

Signature

```
addDynamicTemplate(name: StringUtf8;
                   cls:  Class): IOrbTemplateCursor typeMethod;
```

The **addDynamicTemplate** method of the **Orb** class creates an instance of the class specified by the **cls** parameter and associates it with the template specified by the **name** parameter in the Orb templating engine.

Note The specified class must implement the **IOrbTemplate** interface. For details, see "IOrbTemplate Interface", elsewhere in this document.

If a template with the provided name already exists, an exception occurs.

The method returns an opaque **IOrbTemplateCursor** reference to the template. For details, see "IOrbTemplateCursor Interface", elsewhere in this document.

addHeader

Signature
```
addHeader(name:  StringUtf8;
          value: StringUtf8) typeMethod;
```

The **addHeader** method of the **Orb** class adds an additional header to the request using the name specified by the **name** parameter with the value specified by the **value** parameter.

This method overrides the value of default optional headers that Orb sends.

addHtmlTemplate

Signature
```
addHtmlTemplate(name: StringUtf8;
                path: String): IOrbTemplateCursor typeMethod;
```

The **addHtmlTemplate** method of the **Orb** class loads the content at the location specified by the **path** parameter into the Orb templating engine and associates it with the template specified by the **name** parameter.

Note By default, the path specified in the **path** parameter is opened on the application server. Use the **usePresentationClient** method to make Orb open template and virtual directory files from the file system of the presentation client when running in thin client mode.

If a template with the specified name already exists, an exception occurs.

The method returns an opaque **IOrbTemplateCursor** reference to the template. For details, see "IOrbTemplateCursor Interface", elsewhere in this document.

addLiveRoute

Signature
```
addLiveRoute(path:                String;
             coll:                Collection;
             templateName:        StringUtf8;
             authObject:          Object;
             authenticationMethod: Method) typeMethod;
```

The **addLiveRoute** method of the **Orb** class specifies a route that actions global live updates. This means that all users will see the same data from the same collection.

This method specifies that the URL specified by the **path** parameter should automatically start live updates on the target collection specified by the **coll** parameter.

Objects added to the collection are expanded by the templating engine, using the template specified by the **templateName** parameter.

Caution All users are sent the same data in live changes to this route; for example, if the **templateName** parameter specifies a dynamic template or contains directives that causes data to display dynamically, what is sent to each specific connection is undefined.

If the **authObject** parameter is not null, whenever the live route is accessed, the method specified by the **authenticationMethod** parameter is called with the object specified by the **authObject** parameter as the receiver. If the **authenticationMethod** method returns **false**, the request is denied.

This method is provided for convenience for specific use-cases where all connected users should see the same data; for example, a live stock ticker could be implemented using this method.

Use of this method is discouraged, and it is instead recommended to specify a normal route with the **addRoute** method and manually start live updates with the **liveUpdate** method. This gives you more control over the routing, authentication, and response.

addRoute

Signature

```
addRoute(meth:  String;
         path:  String;
         cls:   Class) typeMethod;
```

The **addRoute** method of the **Orb** class adds the class specified by the **cls** parameter as a route with the HTTP method and URL path specified by the **meth** and **path** parameters, respectively.

The specified class must implement the **IOrbRoute** interface.

The **meth** parameter is an HTTP method and must be one of **GET** or **POST**.

If a route with the same HTTP method and name already exists, an exception occurs. For details, see "Routing Engine", elsewhere in this document.

addRouteWithParameters

Signature

```
addRouteWithParameters(meth:       String;
                       path:       String;
                       cls:        Class;
                       parameters: ParamListType) typeMethod;
```

The **addRouteWithParameters** method of the **Orb** class adds the class specified by the **cls** parameter as a route with the HTTP method and URL path specified by the **meth** and **path** parameters, respectively. The **parameters** parameter specifies additional optional parameters that are passed in the URL path rather than the query string.

Each of the additional parameters must be a string, representing the name of a property on the specified associated route class.

When the route is invoked, any parameters provided in the URL are assigned to properties on the object, based on the order in which they are passed in the parameters list. For example:

1. If a route is added with the following method call.

   ```
   addRouteWithParameters("GET", "/account", AccountRoute, "id", "action");
   ```

2. When this route is invoked by the URL **/account/42/edit**.

3. On the created instance of the **AccountRoute** class, the "**id**" property is set to **42** and the "**action**" property is set to "**edit**".

   Using standard routes, this would be equivalent to **/account?id=42&action=edit**.

addStylesheet

Signature

```
addStylesheet(name: StringUtf8;
              path: String) typeMethod;
```

The **addStylesheet** method of the **Orb** class loads the CSS content at the location specified by the **path** parameter into the Orb templating engine and associates it with the template specified by the **name** parameter.

All style sheets added using this method are automatically linked into documents when the **<@@ orb-headers @@>** directive is expanded.

If a style sheet with the specified name already exists, an exception occurs.

For details, see "HTML Templates", elsewhere in this document.

addView

Signature

```
addView(path: String;
        cls:  Class) typeMethod;
```

The **addView** method of the **Orb** class adds the class specified by the **cls** parameter as a route for the **GET** and **POST** HTTP methods, and the URL path specified in the **path** parameter.

This method is intended for routes that respond to Orb Views. For details, see "IOrbViewController Interface", elsewhere in this document.

addViewWithParameters

Signature

```
addViewWithParameters(path:       String;
                      cls:        Class;
                      parameters: ParamListType) typeMethod;
```

The **addViewWithParameters** method of the **Orb** class adds the class specified by the **cls** parameter as a route for the **GET** and **POST** HTTP methods, and the URL path specified by the **path** parameter. The **parameters** parameter specifies additional optional parameters that are passed in the URL path rather than the query string.

Each of the additional parameters must be a string, representing the name of a property on the provided associated class.

For details about when the route is invoked, see **addRouteWithParameters**.

addVirtualDirectory

Signature

```
addVirtualDirectory(virtualpath:  String;
                    physicalpath: String;
                    maxAge:       ParamListType) typeMethod;
```

The **addVirtualDirectory** method of the **Orb** class adds the directory specified by the **physicalpath** parameter as a virtual directory that is accessible from URLs starting with the path specified by the **virtualpath** parameter.

Note By default, the files are opened on the application server and the path specified in the **physicalpath** parameter must be a path that exists on the application server. Use the **usePresentationClient** method to make Orb open template and virtual directory files from the file system of the presentation client when running in thin client mode.

The **maxAge** parameter is optional. It can be omitted or it can contain a single Integer property. If a value is specified for the **maxAge** parameter, all requests to the virtual directory include a hint to web browsers that the requested resource can be cached for the number of seconds specified by the **maxAge** parameter and not be requested again until this period expires, even if the web page is refreshed. This can dramatically improve performance for static assets but can leave browsers displaying out-of-date assets if the asset changes.

Note This method is provided for convenience and is intended for use during development. It is recommended that for production, assets be served from a web server or other gateway.

All files located in the directory specified by the **physicalpath** parameter are accessible from web requests. For example:

1. If the directory **C:\inetpub\wwwroot** contains the following files:
   - foo.png
   - baz.js
   - private\secret.txt
2. If a virtual directory is added with the following method call:

```
addVirtualDirectory("/stuff", "C:\inetpub\wwwroot");
```

3. A request to:
   - **/stuff/foo.png** responds with the contents of **foo.png** in the directory.

     This request could be made from an HTML **<img>** tag; for example, **<img src="/stuff/foo.png" alt="Foo">**.
   - **/stuff/baz.js** responds with the contents of **baz.js**.

     This request could be made from an HTML **<script>** tag; for example, **<script src="/stuff/baz.js"></script>**.

Note In the previous example, a request to **/stuff/private/secret.txt** responds with the contents of **private\secret.txt** as the **private** directory is under the specified path. It is your responsibility to ensure that sensitive files are not exposed when using the **addVirtualDirectory** method.

bind

Signature

```
bind(name:   StringUtf8;
     object: Object): Binary typeMethod;
```

The **bind** method of the **Orb** class invokes the Orb templating engine to expand the template specified by the **name** parameter with the object specified by the **object** parameter.

This method returns a **Binary** value representing the expanded HTML (or other content). The resulting **Binary** is suitable for returning, by using the **render** method of an Orb route.

Any object properties expanded by the Orb templating engine are sanitized; that is, characters that can potentially affect the rendering of the HTML DOM are replaced with HTML entities.

bindCursor

Signature

```
bindCursor(cursor: IOrbTemplateCursor;
           object: Object): Binary typeMethod;
```

The **bindCursor** method of the **Orb** class invokes the Orb templating engine to bind the **IOrbTemplateCursor** specified by the **cursor** parameter with the object specified by the **object** parameter.

The **IOrbTemplateCursor** specified by the **cursor** parameter is returned by the **addHtmlTemplate** method.

This method returns a **Binary** value representing the expanded HTML (or other content). The resulting **Binary** is suitable for returning, by using the **render** method of an Orb route.

Any object properties expanded by the Orb templating engine are sanitized; that is, characters that can potentially affect the rendering of the HTML DOM are replaced with HTML entities.

bindRaw

Signature
```
bindRaw(name:   StringUtf8;
        object: Object): Binary typeMethod;
```

The **bindRaw** method of the **Orb** class invokes the Orb templating engine to expand the template specified by the **name** parameter with the object specified by the **object** parameter.

This method returns a **Binary** value representing the expanded HTML (or other content). The resulting **Binary** is suitable for returning by using the **render** method of an Orb route.

Unlike the **bind** method, any object properties expanded by the Orb templating engine are *not* sanitized. Although this can be useful for directly rendering properties that contain raw HTML, you must ensure that such HTML is sanitized to avoid issues such as Cross Site Scripting (XSS) attacks (https://owasp.org/www-community/attacks/xss/).

bindRawCursor

Signature
```
bindRawCursor(cursor: IOrbTemplateCursor;
              object: Object): Binary typeMethod;
```

The **bindRawCursor** method of the **Orb** class invokes the Orb templating engine to bind the **IOrbTemplateCursor** specified by the **cursor** parameter with the object specified by the **object** parameter.

The **IOrbTemplateCursor** specified by the **cursor** parameter is returned by the **addHtmlTemplate** method.

This method returns a **Binary** value representing the expanded HTML (or other content). The resulting **Binary** is suitable for returning by using the **render** method of an Orb route.

Unlike the **bind** method, any object properties expanded by the Orb templating engine are *not* sanitized. Although this can be useful for directly rendering properties that contain raw HTML, you must ensure that such HTML is sanitized to avoid issues such as Cross Site Scripting (XSS) attacks (https://owasp.org/www-community/attacks/xss/).

bindRawString

Signature
```
bindRawString(templateString:  String;
              object:          Object): Binary typeMethod;
```

The **bindRawString** method of the **Orb** class invokes the Orb templating engine to expand the template content specified by the **templateString** parameter with the object specified by the **object** parameter.

This method returns a **Binary** value representing the expanded HTML (or other content). The resulting **Binary** is suitable for returning by using the **render** method of an Orb route.

Unlike the **bind** method, any object properties expanded by the Orb templating engine are *not* sanitized. Although this can be useful for directly rendering properties that contain raw HTML, you must ensure that such HTML is sanitized to avoid issues such as Cross Site Scripting (XSS) attacks (https://owasp.org/www-community/attacks/xss/).

bindString

Signature
```
bindString(templateString: String;
           object:         Object): Binary typeMethod;
```

The **bindString** method of the **Orb** class invokes the Orb templating engine to expand the template content with the value specified by the **templateString** parameter using the object specified by the **object** parameter.

This method returns a **Binary** value representing the expanded HTML (or other content). The resulting **Binary** is suitable for returning by using the **render** method of an Orb route.

Any object properties expanded by the templating engine are sanitized; characters that can potentially effect the rendering of the HTML DOM are replaced with HTML entities.

errorHandler

Signature `errorHandler(errHandler: IOrbErrorHandler) typeMethod;`

The **errorHandling** method of the **Orb** class specifies an object that implements the **IOrbErrorHandler** interface which is invoked to handle errors. Additional calls replace the error handler for the application.

For details, see "IOrbErrorHandler Interface", elsewhere in this document.

liveUpdate

Signature
```
liveUpdate(target:       Object;
           templateName: StringUtf8): Binary typeMethod;
```

The **liveUpdate** method of the **Orb** class starts live updates on the target specified by the **target** parameter (which can be any object or collection).

Objects added to the collection are expanded by the templating engine, using the template specified by the **templateName** parameter.

If the target is a:

- Non-collection object, the connecting web browser is notified based on changes to the target.
- Collection, the connecting web browser is notified whenever items are added to or removed from the collection, or whenever objects in the collection are modified.

This method returns a **Binary** value that must be returned by using the **render** method of an Orb route. For details, see "Routing Engine", elsewhere in this document.

redirect

Signature `redirect(url: String): Binary typeMethod;`

The **redirect** method of the **Orb** class instructs the requesting web browser to navigate to the path specified by the **url** parameter.

Note This method performs special handling for Orb triggers and is therefore the only supported way to perform HTTP redirects in the Orb web framework.

This method returns a **Binary** value that must be returned by using the **render** method of an Orb route for the redirect to occur.

request

```
request(): IOrbRequestLens typeMethod;
```

The **request** method of the **Orb** class returns an opaque **IOrbRequestLens** instance that can be used to access metadata for the current request, by calling the appropriate interface method. For more details, see "Accessing Request Metadata", later in this document.

Accessing Request Metadata

The **Orb** class **request** method returns an opaque object that implements the **IOrbRequestLens** interface.

The following table lists the methods provided by the **IOrbRequestLens** interface that can be called to retrieve information about the current request.

Method	Returns...
`verb(): String;`	HTTP request method (for example, **GET** or **POST**)
`body(): Binary;`	Raw HTTP request body
`query(): Binary;`	Query string from the URL
`url(): String;`	URL path
`ip(): Sting;`	IP address of the client making the request
`header(headerName: String): String;`	Value of the HTTP header named by the string

sessionData

Signature `sessionData(sessionDataObject: Any output): Boolean typeMethod;`

The **sessionData** method of the **Orb** class retrieves the session data object associated with the current session.

This method returns **true** if the web browser has a valid session; otherwise it returns **false**.

If the web browser has a valid session, the **sessionDataObject** parameter will be set to a reference of the object that was associated with the session by using the **sessionStart** method.

Note If the object was deleted while the session was still active, the behavior can be undefined.

sessionDataCursor

Signature
```
sessionDataCursor(cursor:            IOrbSessionCursor;
                  sessionDataObject:  Any output): Boolean typeMethod;
```

The **sessionDataCursor** method of the **Orb** class retrieves the session data object associated with the session referenced by the **cursor** parameter.

The typical **sessionData** method retrieves the session data object of the current user session; that is, it can be called only during an active web request.

The **sessionDataCursor** method allows retrieving the session data associated with a session at any time, provided the underlying session still exists.

This method returns **false** if the **cursor** parameter references an invalid session, or the session data object associated with the session is invalid. Otherwise, the **sessionDataObject** parameter is set to a reference of the object that was associated with the session by the **sessionStart** method and this method returns **true**.

sessionEnd

Signature `sessionEnd() typeMethod;`

The **sessionEnd** method of the **Orb** class terminates the active session and instructs the web browser to remove any cookies associated with it.

This method cannot be called in transaction state.

sessionEndCursor

Signature `sessionEndCursor(cursor: IOrbSessionCursor) typeMethod;`

The **sessionEndCursor** method of the **Orb** class terminates the session referenced by the **cursor** parameter, provided the **cursor** parameter references a valid session. The underlying session is deleted and is longer valid.

The typical **sessionEnd** method can be called only during an active request and terminates the current session.

The **sessionEndCursor** method allows terminating any session at any time. After this method returns, the **cursor** parameter references an invalid object.

This method cannot be called in transaction state.

sessionStart

Signature
```
sessionStart(sessionDataObject: Object;
             expiryMinutes:     Integer;
             timeoutMinutes:    Integer): IOrbSessionCursor typeMethod;
```

The **sessionStart** method of the **Orb** class starts a persistent session, associating the object specified by the **sessionDataObject** parameter, which can be any object. The object must exist for the duration of the session or the behavior is undefined. The object specified by the **sessionDataObject** parameter is not affected in any way when the session is terminated.

The **sessionData** method can be used to retrieve the provided **sessionDataObject**.

The session is terminated after the number of minutes specified by the **expiryMinutes** parameter is exceeded.

Additionally, the session is terminated if no requests are made within the number of minutes specified by the **timeoutMinutes** parameter.

Each request resets the timer and if the period specified by the **timeoutMinutes** parameter elapses without a request, the session terminates. The **timeoutMinutes** parameter value can be specified as zero (**0**) to not have an idle timeout (equivalent to setting the **timeoutMinutes** parameter to the same value as the **expiryMinutes** parameter).

This method returns an opaque **IOrbSessionCursor** instance that references the underlying session. For details, see "IOrbSessionCursor Interface", elsewhere in this document.

This method cannot be called in transaction state.

setContentType

Signature `setContentType(content: StringUtf8) typeMethod;`

The **setContentType** method of the **Orb** class overrides the standard **Content-Type** header for the request with the value specified by the **content** parameter.

setDebug

Signature `setDebug(debugOn: Boolean) typeMethod;`

The **setDebug** method of the **Orb** class enables debug mode.

When the **debugOn** parameter is **true**, extra logging is enabled and any exceptions are passed back to the user application. Debug mode is off, by default. For details, see "Debugging an Orb Application", elsewhere in this document.

setStatusCode

Signature `setStatusCode(code: Integer) typeMethod;`

The **setStatusCode** method of the **Orb** class sets the HTTP status code of the response to the value specified by the **code** parameter.

setTitle

Signature `setTitle(newTitle: String) typeMethod;`

The **setTitle** method of the **Orb** class updates the title of a web page when called from an Orb trigger.

The web page title is updated to the value specified by the **newTitle** parameter.

strip

Signature `strip(pathRoot: String) typeMethod;`

The **strip** method of the **Orb** class strips the prefix specified by the **pathRoot** parameter from any incoming requests when determining which route to dispatch to.

This can be useful if your application is behind a gateway that forwards requests based on the URL path.

usePresentationClient

Signature `usePresentationClient(usePresentationClientParam: Boolean) typeMethod;`

The **usePresentationClient** method of the **Orb** class specifies whether Orb opens template and virtual directory files from the file system of the application server or the presentation client when running in thin client mode.

By default, Orb opens template and virtual directory files from the file system of the application server. To open template and virtual directory files from the file system of the presentation client, set the **usePresentationClientParam** parameter value to **true**.

If the Orb application is not started in thin client mode, this method has no effect.

For more details, see the **FileNode** class **usePresentationFileSystem** property, in Volume 1 of the *Encyclopaedia of Classes*.

view

Signature `view(viewController: IOrbViewController): Binary typeMethod;`

The **view** method of the **Orb** class responds to a request with an Orb view specified by the **viewController** parameter. For details, see "IOrbViewController Interface", elsewhere in this document.

wrap

Signature `wrap(any: Any): Binary typeMethod;`

The **wrap** method of the **Orb** class converts a value specified by the **any** parameter to a **Binary** representation suitable for being sent to the requesting web client to display. For example, the typical **Binary** representation of an **Integer** type is unsuitable for displaying in a web browser, therefore if this method is passed an **Integer**, it returns a **Binary** representation of a human-readable form of the **Integer**.

As another example, the **Binary** representation of a **String** type differs depending on if the Jade Platform system is an ANSI or Unicode system. If this method is passed a **String**, it returns a suitable **Binary** representation of the **String** independent of the Jade Platform encoding.

Tip The opposite transformation to this method is performed when a property on a route class is set from request data.

Debugging an Orb Application

For debugging and development, Orb has a debug mode that can be enabled by calling **Orb@setDebug(true)** in the initialization method of an Orb application.

Note Debug mode can have an impact on performance and other behaviors, and should not be enabled in a production environment.

Disabling debug mode after an application has started up is not supported. To stop debugging your Orb application, restart the application and do not call **Orb@setDebug(true)**.

The following is enabled by debug mode.

- Behavior changes
  - Any exceptions that occur are passed back to the application
  - HTML template files are automatically re-read each time the template is rendered

    Note This is to assist development so you do not need to restart the application to see the effect of changes to template files.
- Diagnostics

  The following additional information is logged to the Web Application Monitor window.
  - New live connections and the channel identifier
  - Events sent by using live updates
  - The route class and methods invoked on each request

- Warnings

  The following informational warnings that can assist in debugging are logged to the Web Application Monitor window.

  - Missing **data-orb-id** attribute

Warning	Description
Warning text	Warning: Template <*name*> missing [data-orb-id] attribute Updates and removes will not be reflected by live updates for this template
When it happens	Live updates sent an **add** or **update** event and the outer HTML element did not contain a **data-orb-id** attribute.
What it means	For updates and removes to work in live updates, the outer element of the HTML must have a **data-orb-id** attribute that is set to the expanded value of **<@@ orb-id @@>**.

What to do	Add a **data-orb-id** attribute to the outer element of any template that is expanded by live updates; for example:
	`<div class="outer" data-orb-id="<@@ orb-id @@>">`
	`  <div class="inner">`
	`  <@@ value @@>`
	`  </div>`
	`</div>`

    For details, see "HTML Data Attributes", elsewhere in this document.

  - Direct bind on collection

Warning	Description
Warning text	Warning: Collection (<*collection-class-and-oid*>) directly passed to bind This does not expand the collection, pass the parent instead
When it happens	The **Orb@bind** method (or its variants) is called with an object parameter that is a collection.
What it means	Passing an object to bind expands its properties by the template. If any of those properties is a reference to a collection, the members of that collection are expanded. Directly passing a collection reference to bind would cause any properties *on the collection object* to be expanded, not the *members of the collection*. A common error is passing the collection that the template is to expand, instead of the parent object that has the collection reference. As collection objects typically do not have properties, a bind of a collection is probably an error.
What to do	Pass the parent object of the collection to bind, not the collection itself.

    For details, see "bind", elsewhere in this document.