

# Open Source XML & JSON Visualisation Software

1.0

Generated by Doxygen 1.8.13

Tue Apr 18 2017 16:52:59



# Contents

<b>1</b>	<b>Namespace Index</b>	<b>1</b>
1.1	Packages . . . . .	1
<b>2</b>	<b>Class Index</b>	<b>3</b>
2.1	Class List . . . . .	3
<b>3</b>	<b>File Index</b>	<b>5</b>
3.1	File List . . . . .	5
<b>4</b>	<b>Namespace Documentation</b>	<b>7</b>
4.1	OSXJV Namespace Reference . . . . .	7
4.2	OSXJV.Classes Namespace Reference . . . . .	7
4.3	OSXJV.Server Namespace Reference . . . . .	7
4.4	WebServer Namespace Reference . . . . .	8
<b>5</b>	<b>Class Documentation</b>	<b>9</b>
5.1	OSXJV.Classes.Attribute Class Reference . . . . .	9
5.1.1	Detailed Description . . . . .	10
5.1.2	Member Data Documentation . . . . .	10
5.1.2.1	name . . . . .	10
5.1.2.2	value . . . . .	10
5.1.3	Property Documentation . . . . .	10
5.1.3.1	Name . . . . .	10
5.1.3.2	Value . . . . .	10
5.2	OSXJV.Classes.CacheManager Class Reference . . . . .	11

5.2.1	Detailed Description	12
5.2.2	Constructor & Destructor Documentation	12
5.2.2.1	CacheManager()	12
5.2.3	Member Function Documentation	12
5.2.3.1	Close()	12
5.2.3.2	getFile()	13
5.2.3.3	GetInstance()	14
5.2.3.4	ManageCache()	15
5.2.3.5	saveFile()	15
5.2.3.6	Setup()	16
5.2.4	Member Data Documentation	17
5.2.4.1	Inst	17
5.2.4.2	path	17
5.3	OSXJV.Classes.Logger Class Reference	18
5.3.1	Detailed Description	19
5.3.2	Constructor & Destructor Documentation	19
5.3.2.1	Logger()	19
5.3.3	Member Function Documentation	19
5.3.3.1	Close()	19
5.3.3.2	GetInstance()	20
5.3.3.3	Setup()	20
5.3.3.4	WriteError()	21
5.3.4	Member Data Documentation	22
5.3.4.1	inst	22
5.3.4.2	location	22
5.4	OSXJV.Classes.Node Class Reference	23
5.4.1	Detailed Description	24
5.4.2	Constructor & Destructor Documentation	24
5.4.2.1	Node()	24
5.4.3	Member Data Documentation	24

5.4.3.1	attributes	24
5.4.3.2	children	24
5.4.3.3	comments	25
5.4.3.4	name	25
5.4.3.5	number	25
5.4.3.6	value	25
5.4.3.7	visited	25
5.4.4	Property Documentation	25
5.4.4.1	Attributes	26
5.4.4.2	Children	26
5.4.4.3	Comments	26
5.4.4.4	Name	26
5.4.4.5	Number	27
5.4.4.6	Value	27
5.4.4.7	Visited	27
5.5	OSXJV.Server.OSXJVServer Class Reference	28
5.5.1	Detailed Description	30
5.5.2	Constructor & Destructor Documentation	30
5.5.2.1	OSXJVServer()	30
5.5.3	Member Function Documentation	30
5.5.3.1	GetData()	30
5.5.3.2	GetFileData()	31
5.5.3.3	GetFormData()	32
5.5.3.4	HandleClient()	33
5.5.3.5	HandleGet()	33
5.5.3.6	HandleOptions()	35
5.5.3.7	HandlePost()	36
5.5.3.8	ListenerCallback()	38
5.5.3.9	ManageCache()	39
5.5.3.10	Post()	40

5.5.3.11	Run()	41
5.5.3.12	SaveFile()	41
5.5.3.13	SegmentNormalize()	42
5.5.3.14	Start()	43
5.5.3.15	Stop()	44
5.5.4	Member Data Documentation	44
5.5.4.1	cacheThread	45
5.5.4.2	listener	45
5.5.4.3	port	45
5.5.4.4	running	45
5.5.4.5	serverThread	45
5.6	OSXJV.Classes.Output Class Reference	46
5.6.1	Detailed Description	47
5.6.2	Constructor & Destructor Documentation	47
5.6.2.1	Output()	47
5.6.3	Member Function Documentation	48
5.6.3.1	CheckChildren() [1/2]	48
5.6.3.2	CheckChildren() [2/2]	49
5.6.3.3	CheckNodeNumber()	50
5.6.3.4	CreateExtraNode() [1/2]	51
5.6.3.5	CreateExtraNode() [2/2]	51
5.6.3.6	CreateGrid()	52
5.6.3.7	CreateNodeChildViewsParallel()	53
5.6.3.8	CreateNodeView() [1/2]	54
5.6.3.9	CreateNodeView() [2/2]	55
5.6.3.10	CreatePreviousNode()	56
5.6.3.11	CreateView()	57
5.6.3.12	CreateViewSingle()	59
5.6.3.13	GetParent()	60
5.6.3.14	GridGetChidren()	61

5.6.4	Member Data Documentation . . . . .	61
5.6.4.1	cNodes . . . . .	61
5.6.4.2	GotParent . . . . .	61
5.6.4.3	left . . . . .	62
5.6.4.4	nodes . . . . .	62
5.6.4.5	Parent . . . . .	62
5.6.4.6	top . . . . .	62
5.7	OSXJV.Classes.ProcessDocument Class Reference . . . . .	62
5.7.1	Detailed Description . . . . .	64
5.7.2	Constructor & Destructor Documentation . . . . .	64
5.7.2.1	ProcessDocument() . . . . .	64
5.7.3	Member Function Documentation . . . . .	64
5.7.3.1	GetProcess() . . . . .	64
5.7.3.2	Prepare() . . . . .	65
5.7.3.3	Process() . . . . .	66
5.7.3.4	ProcessComment() . . . . .	66
5.7.3.5	ProcessDocumentParallelInit() . . . . .	67
5.7.3.6	ProcessElement() [1/2] . . . . .	68
5.7.3.7	ProcessElement() [2/2] . . . . .	69
5.7.3.8	ProcessParallel() . . . . .	70
5.7.3.9	ProcessRoot() . . . . .	72
5.7.3.10	ProcessText() . . . . .	72
5.7.4	Member Data Documentation . . . . .	73
5.7.4.1	count . . . . .	73
5.7.4.2	document . . . . .	73
5.7.4.3	node . . . . .	73
5.7.4.4	ProcessedElements . . . . .	74
5.7.4.5	th . . . . .	74
5.7.4.6	ThreadList . . . . .	74
5.7.4.7	type . . . . .	74

5.8	WebServer.Program Class Reference	75
5.8.1	Detailed Description	75
5.8.2	Member Function Documentation	75
5.8.2.1	Main()	75
5.9	OSXJV.Classes.Request Class Reference	76
5.9.1	Detailed Description	78
5.9.2	Constructor & Destructor Documentation	78
5.9.2.1	Request()	78
5.9.3	Member Function Documentation	78
5.9.3.1	GetRequest()	78
5.9.4	Member Data Documentation	79
5.9.4.1	data	79
5.9.4.2	filename	80
5.9.4.3	type	80
5.9.5	Property Documentation	80
5.9.5.1	Data	80
5.9.5.2	Filename	80
5.9.5.3	Type	81
5.10	OSXJV.Classes.Response Class Reference	81
5.10.1	Detailed Description	82
5.10.2	Constructor & Destructor Documentation	82
5.10.2.1	Response()	82
5.10.3	Member Function Documentation	83
5.10.3.1	GetErrorResponse()	83
5.10.3.2	GetInvalidRequestResponse()	84
5.10.3.3	GetResponse()	84
5.10.3.4	GetResponseJSON()	85
5.10.3.5	GetResponseXML()	86
5.10.4	Member Data Documentation	86
5.10.4.1	data	87
5.10.4.2	mime	87
5.10.4.3	status	87
5.11	OSXJV.Classes.Validation Class Reference	87
5.11.1	Detailed Description	88
5.11.2	Constructor & Destructor Documentation	88
5.11.2.1	Validation()	88
5.11.3	Member Function Documentation	88
5.11.3.1	CheckDocument()	88
5.11.3.2	GetInstance()	90
5.11.4	Member Data Documentation	90
5.11.4.1	inst	90



<b>6 File Documentation</b>	<b>91</b>
6.1 <a href="#">WebServiceCSharp/OSXJVClasses/Attribute.cs File Reference</a>	91
6.2 <a href="#">Attribute.cs</a>	91
6.3 <a href="#">WebServiceCSharp/OSXJVClasses/CacheManager.cs File Reference</a>	92
6.4 <a href="#">CacheManager.cs</a>	92
6.5 <a href="#">WebServiceCSharp/OSXJVClasses/Logger.cs File Reference</a>	93
6.6 <a href="#">Logger.cs</a>	93
6.7 <a href="#">WebServiceCSharp/OSXJVClasses/Node.cs File Reference</a>	94
6.8 <a href="#">Node.cs</a>	94
6.9 <a href="#">WebServiceCSharp/OSXJVClasses/Output.cs File Reference</a>	96
6.10 <a href="#">Output.cs</a>	96
6.11 <a href="#">WebServiceCSharp/OSXJVClasses/ProcessDocument.cs File Reference</a>	103
6.12 <a href="#">ProcessDocument.cs</a>	103
6.13 <a href="#">WebServiceCSharp/OSXJVClasses/Request.cs File Reference</a>	107
6.14 <a href="#">Request.cs</a>	108
6.15 <a href="#">WebServiceCSharp/OSXJVClasses/Response.cs File Reference</a>	109
6.16 <a href="#">Response.cs</a>	109
6.17 <a href="#">WebServiceCSharp/OSXJVClasses/Validation.cs File Reference</a>	110
6.18 <a href="#">Validation.cs</a>	110
6.19 <a href="#">WebServiceCSharp/OSXJVServer.cs File Reference</a>	111
6.20 <a href="#">OSXJVServer.cs</a>	111
6.21 <a href="#">WebServiceCSharp/Program.cs File Reference</a>	116
6.22 <a href="#">Program.cs</a>	116
6.23 <a href="#">WebServiceCSharp/Properties/AssemblyInfo.cs File Reference</a>	117
6.24 <a href="#">AssemblyInfo.cs</a>	117
<b>Index</b>	<b>119</b>



# Chapter 1

## Namespace Index

### 1.1 Packages

Here are the packages with brief descriptions (if available):

<a href="#">OSXJV</a>	7
<a href="#">OSXJV.Classes</a>	7
<a href="#">OSXJV.Server</a>	7
<a href="#">WebServer</a>	8



## Chapter 2

# Class Index

### 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">OSXJV.Classes.Attribute</a>	
9	
<a href="#">OSXJV.Classes.CacheManager</a>	
Manages Saving an Retrieving Processed Documents . . . . .	11
<a href="#">OSXJV.Classes.Logger</a>	
A simple class that writes errors to a single file. . . . .	18
<a href="#">OSXJV.Classes.Node</a>	
Contain Processed Document Information . . . . .	23
<a href="#">OSXJV.Server.OSXJVServer</a>	
HTTPServer that process the incoming requests. . . . .	28
<a href="#">OSXJV.Classes.Output</a>	
Creates the <a href="#">Output</a> for the web page to display. . . . .	46
<a href="#">OSXJV.Classes.ProcessDocument</a>	
Class the Processes the document . . . . .	62
<a href="#">WebServer.Program</a>	
The Initialiser . . . . .	75
<a href="#">OSXJV.Classes.Request</a>	
A object containing the document to process, filename and type. . . . .	76
<a href="#">OSXJV.Classes.Response</a>	
The Object containing data to send to the client . . . . .	81
<a href="#">OSXJV.Classes.Validation</a>	
Perform validation on document . . . . .	87



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

WebServiceCSharp/ <a href="#">OSXJVServer.cs</a> . . . . .	111
WebServiceCSharp/ <a href="#">Program.cs</a> . . . . .	116
WebServiceCSharp/OSXJVClasses/ <a href="#">Attribute.cs</a> . . . . .	91
WebServiceCSharp/OSXJVClasses/ <a href="#">CacheManager.cs</a> . . . . .	92
WebServiceCSharp/OSXJVClasses/ <a href="#">Logger.cs</a> . . . . .	93
WebServiceCSharp/OSXJVClasses/ <a href="#">Node.cs</a> . . . . .	94
WebServiceCSharp/OSXJVClasses/ <a href="#">Output.cs</a> . . . . .	96
WebServiceCSharp/OSXJVClasses/ <a href="#">ProcessDocument.cs</a> . . . . .	103
WebServiceCSharp/OSXJVClasses/ <a href="#">Request.cs</a> . . . . .	107
WebServiceCSharp/OSXJVClasses/ <a href="#">Response.cs</a> . . . . .	109
WebServiceCSharp/OSXJVClasses/ <a href="#">Validation.cs</a> . . . . .	110
WebServiceCSharp/Properties/ <a href="#">AssemblyInfo.cs</a> . . . . .	117





## Chapter 4

# Namespace Documentation

### 4.1 OSXJV Namespace Reference

#### Namespaces

- namespace [Classes](#)
- namespace [Server](#)

### 4.2 OSXJV.Classes Namespace Reference

#### Classes

- class [Attribute](#)
- class [CacheManager](#)  
*Manages Saving an Retrieving Processed Documents*
- class [Logger](#)  
*A simple class that writes errors to a single file.*
- class [Node](#)  
*Contain Processed Document Information*
- class [Output](#)  
*Creates the [Output](#) for the web page to display.*
- class [ProcessDocument](#)  
*Class the Processes the document*
- class [Request](#)  
*A object containing the document to process, filename and type.*
- class [Response](#)  
*The Object containing data to send to the client*
- class [Validation](#)  
*Perform validation on document*

### 4.3 OSXJV.Server Namespace Reference

#### Classes

- class [OSXJVServer](#)  
*HTTPServer that process the incoming requests.*

## 4.4 WebServer Namespace Reference

### Classes

- class [Program](#)

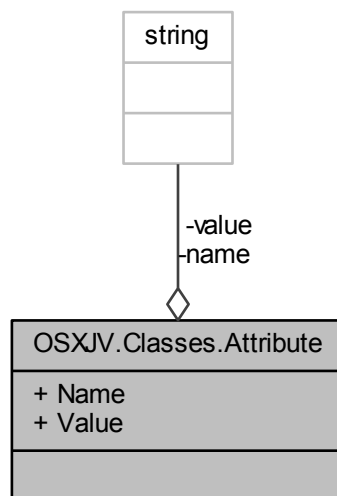
*The Initialiser*

## Chapter 5

# Class Documentation

### 5.1 OSXJV.Classes.Attribute Class Reference

Collaboration diagram for OSXJV.Classes.Attribute:



#### Properties

- string `Name` [get, set]
- string `Value` [get, set]

#### Private Attributes

- string `name`
- string `value`

### 5.1.1 Detailed Description

Definition at line 6 of file [Attribute.cs](#).

### 5.1.2 Member Data Documentation

#### 5.1.2.1 name

```
string OSXJV.Classes.Attribute.name [private]
```

Definition at line 8 of file [Attribute.cs](#).

#### 5.1.2.2 value

```
string OSXJV.Classes.Attribute.value [private]
```

Definition at line 9 of file [Attribute.cs](#).

### 5.1.3 Property Documentation

#### 5.1.3.1 Name

```
string OSXJV.Classes.Attribute.Name [get], [set]
```

Definition at line 15 of file [Attribute.cs](#).

Referenced by [OSXJV.Classes.Output.CreateNodeView\(\)](#), [OSXJV.Classes.ProcessDocument.ProcessElement\(\)](#), and [OSXJV.Classes.ProcessDocument.ProcessRoot\(\)](#).

#### 5.1.3.2 Value

```
string OSXJV.Classes.Attribute.Value [get], [set]
```

Definition at line 30 of file [Attribute.cs](#).

Referenced by [OSXJV.Classes.Output.CreateNodeView\(\)](#), [OSXJV.Classes.ProcessDocument.ProcessElement\(\)](#), and [OSXJV.Classes.ProcessDocument.ProcessRoot\(\)](#).

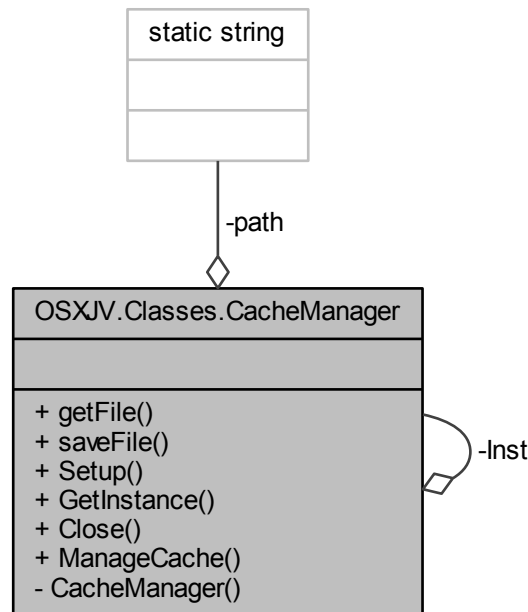
The documentation for this class was generated from the following file:

- [WebServiceCSharp/OSXJVClasses/Attribute.cs](#)

## 5.2 OSXJV.Classes.CacheManager Class Reference

Manages Saving an Retrieiving Processed Documents

Collaboration diagram for OSXJV.Classes.CacheManager:



### Public Member Functions

- string `getFile` (string ID)  
*Retrieve the file from caching*
- bool `saveFile` (string ID, string nodes)  
*Save the file to the local system for caching*

### Static Public Member Functions

- static bool `Setup` (string path)  
*Initialises `CacheManager` with input path*
- static `CacheManager` `GetInstance` ()  
*Get the single instance of the class*
- static void `Close` ()  
*Destroys the created instance*
- static void `ManageCache` ()  
*Manages files in the cache deleting old ones than 6 hours when called*

## Private Member Functions

- [CacheManager](#) (string cachePath)  
*Constructor setting store location;*

## Static Private Attributes

- static [CacheManager](#) Inst  
*Instance of [CacheManager](#)*
- static string [path](#) = null  
*Cache folder location*

### 5.2.1 Detailed Description

Manages Saving an Retrieving Processed Documents

Definition at line 9 of file [CacheManager.cs](#).

### 5.2.2 Constructor & Destructor Documentation

#### 5.2.2.1 CacheManager()

```
OSXJV.Classes.CacheManager.CacheManager (
    string cachePath ) [private]
```

Constructor setting store location;

#### Parameters

<i>path</i>	
-------------	--

Definition at line 25 of file [CacheManager.cs](#).

```
00026         {
00027             path = cachePath;
00028         }
```

### 5.2.3 Member Function Documentation

#### 5.2.3.1 Close()

```
static void OSXJV.Classes.CacheManager.Close ( ) [static]
```

Destroys the created instance

## Exceptions

<i>Exception</i>	If Instance isn't previously created
------------------	--------------------------------------

Definition at line 114 of file [CacheManager.cs](#).

```

00115     {
00116         if (Inst == null)
00117             throw new Exception("CacheManager Already Closed");
00118         else
00119         {
00120             path = null; //Clear static path
00121             Inst = null; //clear static instance
00122         }
00123     }

```

## 5.2.3.2 getFile()

```

string OSXJV.Classes.CacheManager.getFile (
    string ID )

```

Retrieve the file from caching

## Parameters

<i>ID</i>	Unique ID of the file
-----------	-----------------------

## Returns

Definition at line 63 of file [CacheManager.cs](#).

Referenced by [OSXJV.Server.OSXJVServer.HandleGet\(\)](#).

```

00064     {
00065         if (string.IsNullOrEmpty(ID))
00066             throw new ArgumentException("ID cannot be null or empty");
00067
00068         string filePath = path + "/" + ID.Replace("/", "") + ".json";
00069         string output = "";
00070
00071         using (StreamReader sr = new StreamReader(filePath))
00072         {
00073             output = sr.ReadToEnd();
00074         }
00075
00076         if (!string.IsNullOrEmpty(output))
00077             return output;
00078         else
00079             throw new Exception("Error Reading From File");
00080     }

```

Here is the caller graph for this function:



### 5.2.3.3 GetInstance()

```
static CacheManager OSXJV.Classes.CacheManager.GetInstance ( ) [static]
```

Get the single instance of the class

#### Returns

An instance of [CacheManager](#)

#### Exceptions

<i>Exception</i>	If the CacheManger has not been setup
------------------	---------------------------------------

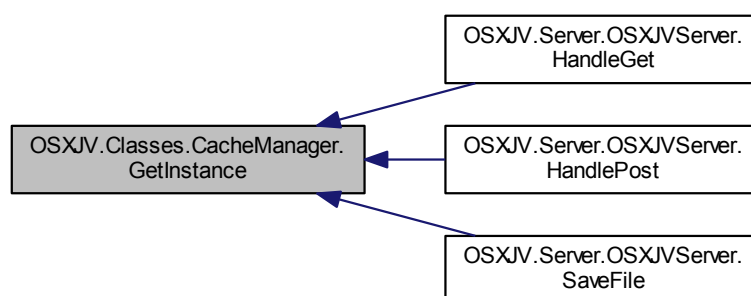
Definition at line 50 of file [CacheManager.cs](#).

Referenced by [OSXJV.Server.OSXJVServer.HandleGet\(\)](#), [OSXJV.Server.OSXJVServer.HandlePost\(\)](#), and [OSXJV.Server.OSXJVServer.SaveFile\(\)](#).

```

00051      {
00052          if (Inst != null)
00053              return Inst;
00054          else
00055              throw new Exception("CacheManger has not been setup");
00056      }
  
```

Here is the caller graph for this function:





### 5.2.3.4 ManageCache()

```
static void OSXJV.Classes.CacheManager.ManageCache ( ) [static]
```

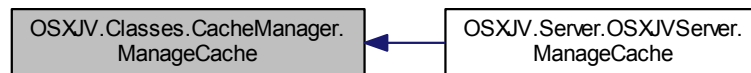
Manages files in the cache deleting old ones than 6 hours when called

Definition at line 128 of file [CacheManager.cs](#).

Referenced by [OSXJV.Server.OSXJVServer.ManageCache\(\)](#).

```
00129         {
00130             if (path != null)
00131             {
00132                 string[] files = Directory.GetFiles(path);
00133                 foreach (string file in files)
00134                 {
00135                     if (File.GetLastAccessTime(file) < DateTime.Now.AddHours(-6.0))
00136                         File.Delete(file);
00137                 }
00138             }
00139             else
00140                 throw new Exception("CacheManger not setup");
00141         }
```

Here is the caller graph for this function:



### 5.2.3.5 saveFile()

```
bool OSXJV.Classes.CacheManager.saveFile (
    string ID,
    string nodes )
```

Save the file to the local system for caching

#### Parameters

<i>ID</i>	Unique ID of the file
<i>nodes</i>	The document to be saved

Definition at line 87 of file [CacheManager.cs](#).

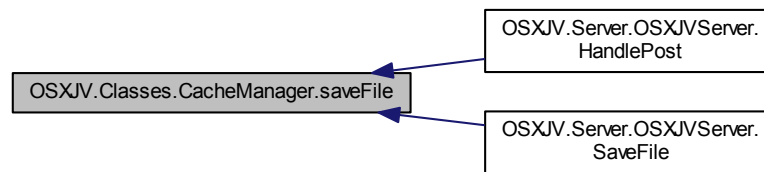
Referenced by [OSXJV.Server.OSXJVServer.HandlePost\(\)](#), and [OSXJV.Server.OSXJVServer.SaveFile\(\)](#).

```

00088      {
00089          if (string.IsNullOrEmpty(ID))
00090              throw new ArgumentException("ID cannot be null or empty");
00091
00092          if (string.IsNullOrEmpty(nodes))
00093              throw new ArgumentException("Document cannot be null or empty");
00094
00095          string filePath = path + "/" + ID + ".json";
00096          try
00097          {
00098              using (StreamWriter sw = new StreamWriter(filePath))
00099              {
00100                  sw.WriteLine(nodes);
00101              }
00102          }
00103          catch
00104          {
00105              throw new Exception("Failed to save file");
00106          }
00107
00108          return true;
00109      }

```

Here is the caller graph for this function:



### 5.2.3.6 Setup()

```

static bool OSXJV.Classes.CacheManager.Setup (
    string path ) [static]

```

Initialises [CacheManager](#) with input path

#### Parameters

<i>path</i>	
-------------	--

Definition at line 34 of file [CacheManager.cs](#).

Referenced by [OSXJV.Server.OSXJVServer.Start\(\)](#).

```

00035      {
00036          if (string.IsNullOrEmpty(path))
00037              throw new ArgumentException("Path cannot be empty");
00038

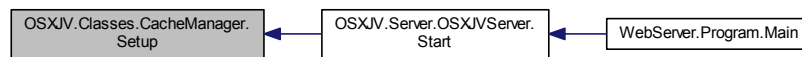
```

```

00039         if (!Directory.Exists(string.Format(@"{0}", path)))
00040             throw new Exception("Path is not a valid cache directory");
00041
00042         return (Inst = new CacheManager(path)) != null ? true : false;
00043     }

```

Here is the caller graph for this function:



## 5.2.4 Member Data Documentation

### 5.2.4.1 Inst

[CacheManager](#) OSXJV.Classes.CacheManager.Inst [static], [private]

Instance of [CacheManager](#)

Definition at line 14 of file [CacheManager.cs](#).

### 5.2.4.2 path

string OSXJV.Classes.CacheManager.path = null [static], [private]

Cache folder location

Definition at line 19 of file [CacheManager.cs](#).

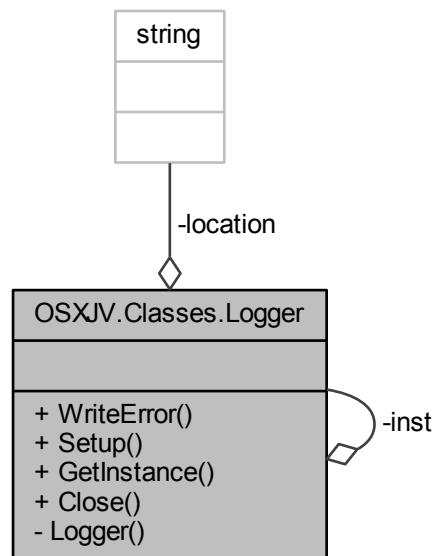
The documentation for this class was generated from the following file:

- [WebServiceCSharp/OSXJVClasses/CacheManager.cs](#)

## 5.3 OSXJV.Classes.Logger Class Reference

A simple class that writes errors to a single file.

Collaboration diagram for OSXJV.Classes.Logger:



### Public Member Functions

- void [WriteError](#) (string error)  
*Writes an error the location provided*

### Static Public Member Functions

- static bool [Setup](#) (string [location](#))
- static [Logger](#) [GetInstance](#) ()  
*Gets the single instance of [Logger](#)*
- static void [Close](#) ()

### Private Member Functions

- [Logger](#) (string [location](#))

### Private Attributes

- string [location](#)

## Static Private Attributes

- static [Logger inst](#)  
*Singleton instance of [Logger](#)*

### 5.3.1 Detailed Description

A simple class that writes errors to a single file.

Definition at line 9 of file [Logger.cs](#).

### 5.3.2 Constructor & Destructor Documentation

#### 5.3.2.1 Logger()

```
OSXJV.Classes.Logger.Logger (
    string location ) [private]
```

Definition at line 17 of file [Logger.cs](#).

```
00018         {
00019             this.location = location;
00020         }
```

### 5.3.3 Member Function Documentation

#### 5.3.3.1 Close()

```
static void OSXJV.Classes.Logger.Close ( ) [static]
```

Definition at line 72 of file [Logger.cs](#).

```
00073         {
00074             if (inst == null)
00075                 throw new Exception("Logger Already Closed");
00076             else
00077                 inst = null;
00078         }
```

### 5.3.3.2 GetInstance()

```
static Logger OSXJV.Classes.Logger.GetInstance ( ) [static]
```

Gets the single instance of [Logger](#)

#### Returns

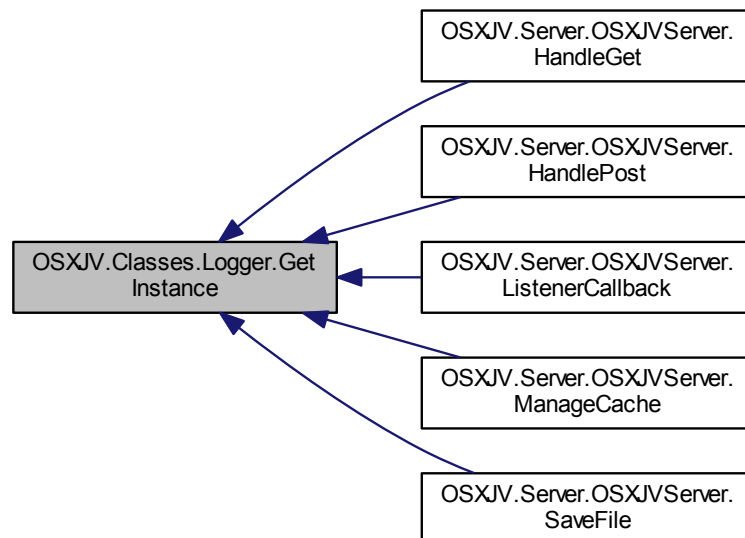
Instance of [Logger](#)

Definition at line 41 of file [Logger.cs](#).

Referenced by [OSXJV.Server.OSXJVServer.HandleGet\(\)](#), [OSXJV.Server.OSXJVServer.HandlePost\(\)](#), [OSXJV.Server.OSXJVServer.ListenerCallback\(\)](#), [OSXJV.Server.OSXJVServer.ManageCache\(\)](#), and [OSXJV.Server.OSXJVServer.SaveFile\(\)](#).

```
00042     {
00043         if (inst != null)
00044             return inst;
00045         else
00046             throw new Exception("Logger has not been setup");
00047     }
```

Here is the caller graph for this function:



### 5.3.3.3 Setup()

```
static bool OSXJV.Classes.Logger.Setup (
    string location ) [static]
```

## Parameters

<i>location</i>	
-----------------	--

Definition at line 26 of file [Logger.cs](#).

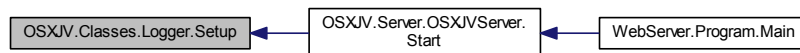
Referenced by [OSXJV.Server.OSXJVServer.Start\(\)](#).

```

00027     {
00028         if (string.IsNullOrEmpty(location))
00029             throw new ArgumentException("Location cannot be empty");
00030
00031         if (!Directory.Exists(string.Format(@"{0}", location)))
00032             throw new Exception("Location is not a valid logger directory");
00033
00034         return (inst = new Logger(location)) != null ? true:false;
00035     }

```

Here is the caller graph for this function:



## 5.3.3.4 WriteError()

```

void OSXJV.Classes.Logger.WriteError (
    string error )

```

Writes an error the location provided

## Parameters

<i>error</i>	The error message
--------------	-------------------

Definition at line 53 of file [Logger.cs](#).

Referenced by [OSXJV.Server.OSXJVServer.HandleGet\(\)](#), [OSXJV.Server.OSXJVServer.HandlePost\(\)](#), [OSXJV.Server.OSXJVServer.ListenerCallback\(\)](#), [OSXJV.Server.OSXJVServer.ManageCache\(\)](#), and [OSXJV.Server.OSXJVServer.SaveFile\(\)](#).

```

00054     {
00055         try
00056         {
00057             if (!string.IsNullOrEmpty(error))
00058             {
00059                 string file = string.Format(@"{0}/Error-{1}.txt", location, DateTime.Now.
ToString("dd-MM-yy hh-MM-ss"));
00060                 StreamWriter sw = new StreamWriter(file);
00061                 sw.WriteLine(error);

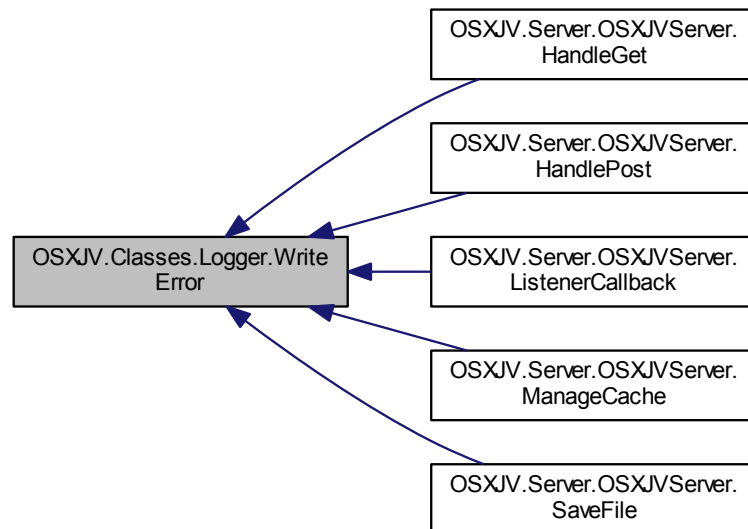
```

```

00062         sw.WriteLine();
00063         sw.Close();
00064     }
00065 }
00066 catch (IOException e)
00067 {
00068     throw e;
00069 }
00070 }

```

Here is the caller graph for this function:



### 5.3.4 Member Data Documentation

#### 5.3.4.1 inst

`Logger` `OSXJV.Classes.Logger.inst` `[static]`, `[private]`

Singleton instance of `Logger`

Definition at line 14 of file `Logger.cs`.

#### 5.3.4.2 location

`string` `OSXJV.Classes.Logger.location` `[private]`

Definition at line 15 of file `Logger.cs`.

The documentation for this class was generated from the following file:

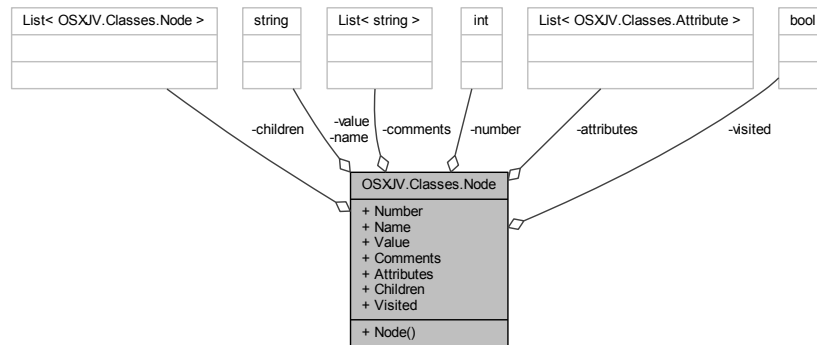
- `WebServiceCSharp/OSXJVClasses/Logger.cs`



## 5.4 OSXJV.Classes.Node Class Reference

Contain Processed Document Information

Collaboration diagram for OSXJV.Classes.Node:



### Public Member Functions

- [Node](#) ()  
*Constructor*

### Properties

- int [Number](#) [get, set]  
*The Number of the [Node](#)*
- string [Name](#) [get, set]  
*The Name of [Node](#)*
- string [Value](#) [get, set]  
*The Value of the [Node](#)*
- List< string > [Comments](#) [get, set]  
*Comments That the [Node](#) Has.*
- List< [Attribute](#) > [Attributes](#) [get, set]  
*Attributes the [Node](#) has.*
- List< [Node](#) > [Children](#) [get, set]  
*Children Nodes the [Node](#) is linked to.*
- bool [Visited](#) [get, set]  
*If the node has been visited previous by the [ProcessDocument](#), prevent multiple same Nodes.*

### Private Attributes

- string [name](#)
- List< [Attribute](#) > [attributes](#)
- string [value](#)
- List< [Node](#) > [children](#)
- int [number](#)
- bool [visited](#)
- List< string > [comments](#)

### 5.4.1 Detailed Description

Contain Processed Document Information

Definition at line 10 of file [Node.cs](#).

### 5.4.2 Constructor & Destructor Documentation

#### 5.4.2.1 Node()

```
OSXJV.Classes.Node.Node ( )
```

Constructor

Definition at line 23 of file [Node.cs](#).

```
00024     {  
00025         Attributes = new List<Attribute>();  
00026         Children = new List<Node>();  
00027         Comments = new List<string>();  
00028         number = 0;  
00029         visited = false;  
00030     }
```

### 5.4.3 Member Data Documentation

#### 5.4.3.1 attributes

```
List<Attribute> OSXJV.Classes.Node.attributes [private]
```

Definition at line 13 of file [Node.cs](#).

#### 5.4.3.2 children

```
List<Node> OSXJV.Classes.Node.children [private]
```

Definition at line 15 of file [Node.cs](#).

#### 5.4.3.3 comments

```
List<string> OSXJV.Classes.Node.comments [private]
```

Definition at line 18 of file [Node.cs](#).

#### 5.4.3.4 name

```
string OSXJV.Classes.Node.name [private]
```

Definition at line 12 of file [Node.cs](#).

#### 5.4.3.5 number

```
int OSXJV.Classes.Node.number [private]
```

Definition at line 16 of file [Node.cs](#).

#### 5.4.3.6 value

```
string OSXJV.Classes.Node.value [private]
```

Definition at line 14 of file [Node.cs](#).

#### 5.4.3.7 visited

```
bool OSXJV.Classes.Node.visited [private]
```

Definition at line 17 of file [Node.cs](#).

### 5.4.4 Property Documentation

#### 5.4.4.1 Attributes

```
List<Attribute> OSXJV.Classes.Node.Attributes [get], [set]
```

Attributes the [Node](#) has.

Definition at line 102 of file [Node.cs](#).

Referenced by [OSXJV.Classes.Output.CreateNodeView\(\)](#), [OSXJV.Classes.ProcessDocument.ProcessElement\(\)](#), and [OSXJV.Classes.ProcessDocument.ProcessRoot\(\)](#).

#### 5.4.4.2 Children

```
List<Node> OSXJV.Classes.Node.Children [get], [set]
```

Children Nodes the [Node](#) is linked to.

Definition at line 119 of file [Node.cs](#).

Referenced by [OSXJV.Classes.Output.CheckChildren\(\)](#), [OSXJV.Classes.Output.CreateGrid\(\)](#), [OSXJV.Classes.Output.CreateView\(\)](#), [OSXJV.Classes.Output.CreateViewSingle\(\)](#), [OSXJV.Classes.Output.GetParent\(\)](#), [OSXJV.Classes.Output.GridGetChidren\(\)](#), [OSXJV.Classes.ProcessDocument.ProcessDocumentParallelInit\(\)](#), [OSXJV.Classes.ProcessDocument.ProcessElement\(\)](#), and [OSXJV.Classes.ProcessDocument.ProcessParallel\(\)](#).

#### 5.4.4.3 Comments

```
List<string> OSXJV.Classes.Node.Comments [get], [set]
```

Comments That the [Node](#) Has.

Definition at line 85 of file [Node.cs](#).

Referenced by [OSXJV.Classes.Output.CreateNodeChildViewsParallel\(\)](#), [OSXJV.Classes.Output.CreateNodeView\(\)](#), and [OSXJV.Classes.ProcessDocument.ProcessComment\(\)](#).

#### 5.4.4.4 Name

```
string OSXJV.Classes.Node.Name [get], [set]
```

The Name of [Node](#)

Definition at line 52 of file [Node.cs](#).

Referenced by [OSXJV.Classes.Output.CreateGrid\(\)](#), [OSXJV.Classes.Output.CreateNodeView\(\)](#), [OSXJV.Classes.Output.GridGetChidren\(\)](#), [OSXJV.Classes.ProcessDocument.ProcessElement\(\)](#), and [OSXJV.Classes.ProcessDocument.ProcessRoot\(\)](#).

#### 5.4.4.5 Number

```
int OSXJV.Classes.Node.Number [get], [set]
```

The Number of the [Node](#)

Definition at line 36 of file [Node.cs](#).

Referenced by [OSXJV.Classes.Output.CheckChildren\(\)](#), [OSXJV.Classes.Output.CheckNodeNumber\(\)](#), [OSXJV.Classes.Output.CreateGrid\(\)](#), [OSXJV.Classes.Output.CreateNodeView\(\)](#), [OSXJV.Classes.Output.CreateView\(\)](#), [OSXJV.Classes.Output.CreateViewSingle\(\)](#), [OSXJV.Classes.Output.GetParent\(\)](#), [OSXJV.Classes.Output.Grid↵GetChidren\(\)](#), [OSXJV.Classes.ProcessDocument.ProcessElement\(\)](#), and [OSXJV.Classes.ProcessDocument.↵ProcessRoot\(\)](#).

#### 5.4.4.6 Value

```
string OSXJV.Classes.Node.Value [get], [set]
```

The Value of the [Node](#)

Definition at line 69 of file [Node.cs](#).

Referenced by [OSXJV.Classes.Output.CreateNodeView\(\)](#), and [OSXJV.Classes.ProcessDocument.ProcessText\(\)](#).

#### 5.4.4.7 Visited

```
bool OSXJV.Classes.Node.Visited [get], [set]
```

If the node has been visited previous by the [ProcessDocument](#), prevent multiple same Nodes.

Definition at line 136 of file [Node.cs](#).

Referenced by [OSXJV.Classes.ProcessDocument.ProcessElement\(\)](#), and [OSXJV.Classes.ProcessDocument.↵ProcessRoot\(\)](#).

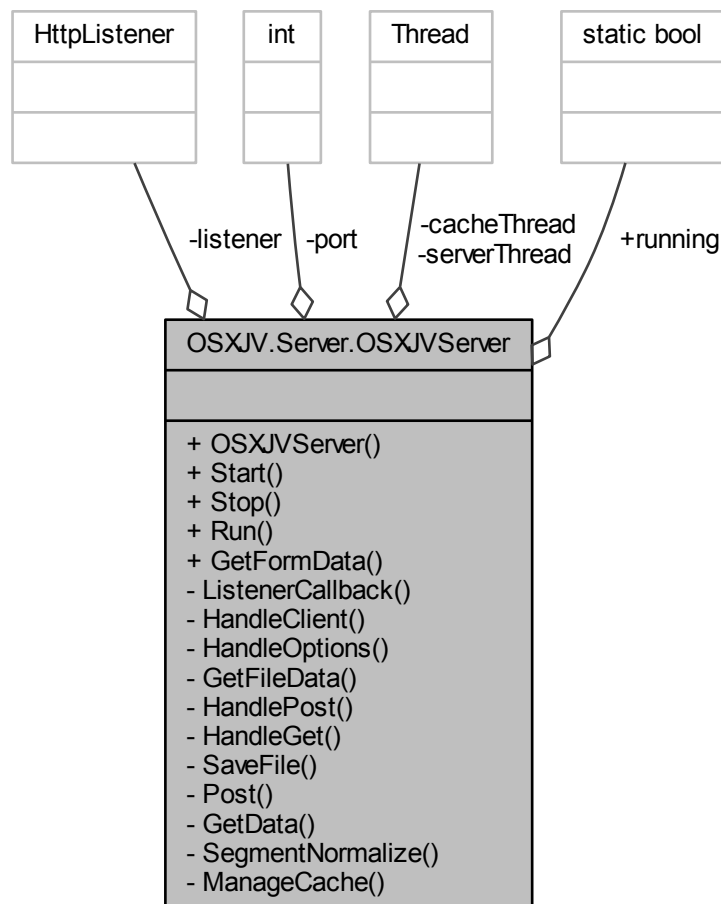
The documentation for this class was generated from the following file:

- [WebServiceCSharp/OSXJVCclasses/Node.cs](#)

## 5.5 OSXJV.Server.OSXJVServer Class Reference

HTTPServer that process the incoming requests.

Collaboration diagram for OSXJV.Server.OSXJVServer:



### Public Member Functions

- [OSXJVServer](#) ()  
*The [Server](#) Handler*
- bool [Start](#) (string cachePath, string loggerPath)  
*Starts server in new thread*

#### Parameters

cachePath	Cache Folder Location
loggerPath	Logger Folder Location

- bool [Stop](#) ()  
*Stop the listener and about all current requests*
- void [Run](#) ()  
*Function that constantly listens for connections*
- [Request GetFormData](#) (Stream input)  
*Extract the files from the request*

## Static Public Attributes

- static bool [running](#) = false  
*True if the server is able to accept requests.*

## Private Member Functions

- void [ListenerCallback](#) (IAsyncResult result)  
*Handles Requests Asynchronously*
- void [HandleClient](#) (HttpListenerContext c)  
*Handles the client*
- void [HandleOptions](#) (HttpListenerResponse response)  
*Sends to the Client What the [Server](#) Supports*
- [Request GetFileData](#) (Stream input, string type)  
*Get Data if the data is retrieved*
- [Response HandlePost](#) (HttpListenerRequest req)  
*Handles a POST request.*
- [Response HandleGet](#) (HttpListenerRequest req)  
*Handles a GET request.*
- void [SaveFile](#) (string id, [Node](#) nodes)  
*Save data recieved from client.*
- void [Post](#) ([Response](#) res, HttpListenerResponse stream)  
*Send data to the client.*
- [Request GetData](#) (HttpListenerRequest req)  
*Get the data from the client.*
- string [SegmentNormalize](#) (string input)  
*Removes '/' from the string.*
- void [ManageCache](#) ()  
*Manages cache*

## Private Attributes

- int [port](#) = 8082
- HttpListener [listener](#)  
*HttpListener*
- Thread [serverThread](#) = null  
*[Server](#) Thread*
- Thread [cacheThread](#) = null  
*Cache Controller*

### 5.5.1 Detailed Description

HTTPServer that process the incoming requests.

Definition at line 16 of file [OSXJVServer.cs](#).

### 5.5.2 Constructor & Destructor Documentation

#### 5.5.2.1 OSXJVServer()

OSXJV.Server.OSXJVServer.OSXJVServer ( )

The [Server](#) Handler

Definition at line 43 of file [OSXJVServer.cs](#).

```
00044     {
00045         listener = new HttpListener();
00046         listener.Prefixes.Add("http://localhost:" + port + "/"); //change if need be
00047     }
```

### 5.5.3 Member Function Documentation

#### 5.5.3.1 GetData()

[Request](#) OSXJV.Server.OSXJVServer.GetData (   
 HttpListenerRequest req ) [private]

Get the data from the client.

#### Parameters

<i>req</i>	The request from the client
------------	-----------------------------

#### Returns

A Request Object

Definition at line 414 of file [OSXJVServer.cs](#).

```
00415     {
00416         Request r = null;
00417
00418         if (req.ContentType.Contains("application/x-www-form-urlencoded"))
00419         {
```



```

00420         r = GetFormData(req.InputStream);
00421     }
00422     else if (req.ContentType.Contains("application/json") || req.ContentType.Contains("
application/oclet-stream"))
00423     {
00424         r = GetFileData(req.InputStream, "application/json");
00425     }
00426     else if (req.ContentType.Contains("application/xml") || req.ContentType.Contains("text/xml"))
00427     {
00428         r = GetFileData(req.InputStream, "text/xml");
00429     }
00430     return r;
00431 }

```

### 5.5.3.2 GetFileData()

**Request** OSXJV.Server.OSXJVServer.GetFileData (   
     Stream *input*,   
     string *type* ) [private]

Get Data if the data is retrieved

#### Parameters

<i>input</i>	Client Stream Input
<i>type</i>	The MIME type

#### Returns

A Response object to send to the user

Definition at line 210 of file [OSXJVServer.cs](#).

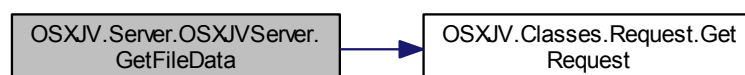
References [OSXJV.Classes.Request.GetRequest\(\)](#).

```

00211     {
00212         string request = "";
00213         using (StreamReader ms = new StreamReader(input))
00214         {
00215             request = ms.ReadToEnd();
00216         }
00217         string filename = "temp";
00218
00219         if (type == "text/xml")
00220             filename += ".xml";
00221         else if (type == "application/json")
00222             filename += ".json";
00223         else
00224             filename += ".html";
00225
00226         return Request.GetRequest(filename, type, request);
00227     }

```

Here is the call graph for this function:



### 5.5.3.3 GetFormData()

[Request](#) OSXJV.Server.OSXJVServer.GetFormData (   
 Stream *input* )

Extract the files from the request

#### Parameters

<i>input</i>	Requests input stream
--------------	-----------------------

#### Returns

New Request Object

#### Exceptions

<i>System.InvalidOperationException</i>	Thrown when no files are included with the request
---	--

Definition at line 186 of file [OSXJVServer.cs](#).

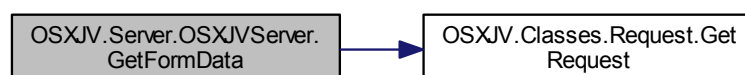
References [OSXJV.Classes.Request.GetRequest\(\)](#).

```

00187     {
00188         string request = "";
00189         MultipartFormDataParser parser = new MultipartFormDataParser(input);
00190         if (parser.Files.Count > 0)
00191         {
00192             using (StreamReader ms = new StreamReader(parser.Files[0].Data))
00193             {
00194                 request = ms.ReadToEnd();
00195             }
00196         }
00197         else
00198         {
00199             throw new InvalidOperationException();
00200         }
00201         return Request.GetRequest(parser.Files[0].FileName, parser.Files[0].
00202             ContentType, request);
00202     }

```

Here is the call graph for this function:



## 5.5.3.4 HandleClient()

```
void OSXJV.Server.OSXJVServer.HandleClient (
    HttpListenerContext c ) [private]
```

Handles the client

## Parameters

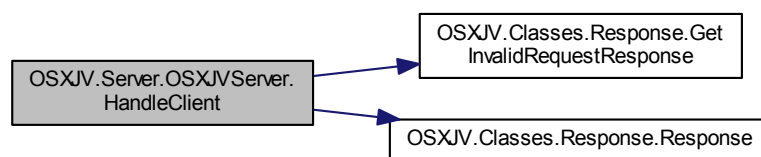
c	The Request
---	-------------

Definition at line 147 of file [OSXJVServer.cs](#).

References [OSXJV.Classes.Response.GetInvalidRequestResponse\(\)](#), and [OSXJV.Classes.Response.Response\(\)](#).

```
00148     {
00149         switch (c.Request.HttpMethod)
00150         {
00151             case "POST":
00152                 Post (HandlePost (c.Request), c.Response);
00153                 break;
00154             case "GET":
00155                 Post (HandleGet (c.Request), c.Response);
00156                 break;
00157             case "OPTIONS":
00158                 HandleOptions (c.Response);
00159                 c.Response.Close();
00160                 break;
00161             default:
00162                 Post (Response.GetInvalidRequestResponse(), c.
00163                     Response);
00164                 break;
00165         }
00166     }
```

Here is the call graph for this function:



## 5.5.3.5 HandleGet()

```
Response OSXJV.Server.OSXJVServer.HandleGet (
    HttpListenerRequest req ) [private]
```

Handles a GET request.

## Parameters

<i>req</i>	The request to be processed.
------------	------------------------------

## Returns

A Response object to send to the user

Definition at line 313 of file [OSXJVServer.cs](#).

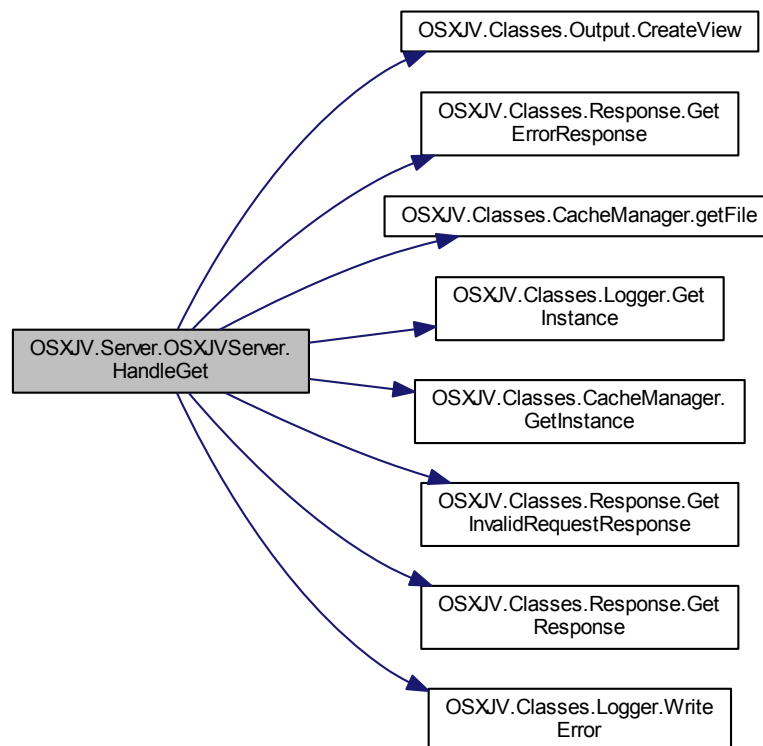
References [OSXJV.Classes.Output.CreateView\(\)](#), [OSXJV.Classes.Response.GetErrorResponse\(\)](#), [OSXJV.Classes.CacheManager.getFile\(\)](#), [OSXJV.Classes.Logger.GetInstance\(\)](#), [OSXJV.Classes.CacheManager.GetInstance\(\)](#), [OSXJV.Classes.Response.GetInvalidRequestResponse\(\)](#), [OSXJV.Classes.Response.GetResponse\(\)](#), and [OSXJV.Classes.Logger.WriteError\(\)](#).

```

00314     {
00315         if (SegmentNormalize(req.Url.Segments[1]).Equals("Process"))
00316         {
00317             if (req.Url.Segments.Length == 4)
00318             {
00319                 Node cached;
00320                 try
00321                 {
00322                     cached = JsonConvert.DeserializeObject<Node>(
00323 CacheManager.GetInstance().getFile(req.Url.Segments[2]));
00324                 }
00325                 catch (Exception e)
00326                 {
00327                     Logger.GetInstance().WriteError(e.Message);
00328                     JObject eRes = new JObject();
00329                     eRes.Add("Error", "Error Creating Response");
00330                     return Response.GetErrorResponse(eRes.ToString());
00331                 }
00332                 Output o = new Output(cached);
00333                 JObject response = new JObject();
00334                 response.Add("view", o.CreateView(int.Parse(req.Url.Segments[3])));
00335                 byte[] bytes = Encoding.UTF8.GetBytes(response.ToString());
00336                 return Response.GetResponse(200, "application/json", bytes);
00337             }
00338             else if (req.Url.Segments.Length == 5)
00339             {
00340                 Node cached;
00341                 try
00342                 {
00343                     cached = JsonConvert.DeserializeObject<Node>(
00344 CacheManager.GetInstance().getFile(req.Url.Segments[2]));
00345                 }
00346                 catch (Exception e)
00347                 {
00348                     Logger.GetInstance().WriteError(e.Message);
00349                     JObject eRes = new JObject();
00350                     eRes.Add("Error", "Error Creating Response");
00351                     return Response.GetErrorResponse(eRes.ToString());
00352                 }
00353                 Output o = new Output(cached);
00354                 JObject response = new JObject();
00355                 response.Add("view", o.CreateView(int.Parse(
SegmentNormalize(req.Url.Segments[3]), 4, int.Parse(
SegmentNormalize(req.Url.Segments[4]))));
00356                 byte[] bytes = Encoding.UTF8.GetBytes(response.ToString());
00357                 return Response.GetResponse(200, "application/json", bytes);
00358             }
00359             else
00360                 return Response.GetInvalidRequestResponse();
00361         }
00362         //If it got here its an invalid response.
00363         return Response.GetInvalidRequestResponse();
00364     }

```

Here is the call graph for this function:



#### 5.5.3.6 HandleOptions()

```
void OSXJV.Server.OSXJVServer.HandleOptions (
    HttpListenerResponse response ) [private]
```

Sends to the Client What the [Server](#) Supports

##### Parameters

<i>response</i>	The Request Response Object
-----------------	-----------------------------

Definition at line 171 of file [OSXJVServer.cs](#).

```
00172     {
00173         response.AddHeader("Access-Control-Allow-Headers", "Content-Type, Accept, X-Requested-With");
00174         response.AddHeader("Access-Control-Allow-Methods", "POST");
00175         response.AddHeader("Access-Control-Allow-Methods", "GET");
00176         response.AddHeader("Access-Control-Max-Age", "1728000");
00177         response.AppendHeader("Access-Control-Allow-Origin", "*");
00178     }
```

### 5.5.3.7 HandlePost()

```
Response OSXJV.Server.OSXJVServer.HandlePost (
    HttpListenerRequest req ) [private]
```

Handles a POST request.

#### Parameters

<i>req</i>	The request to be processed.
------------	------------------------------

#### Returns

A Response object to send to the user

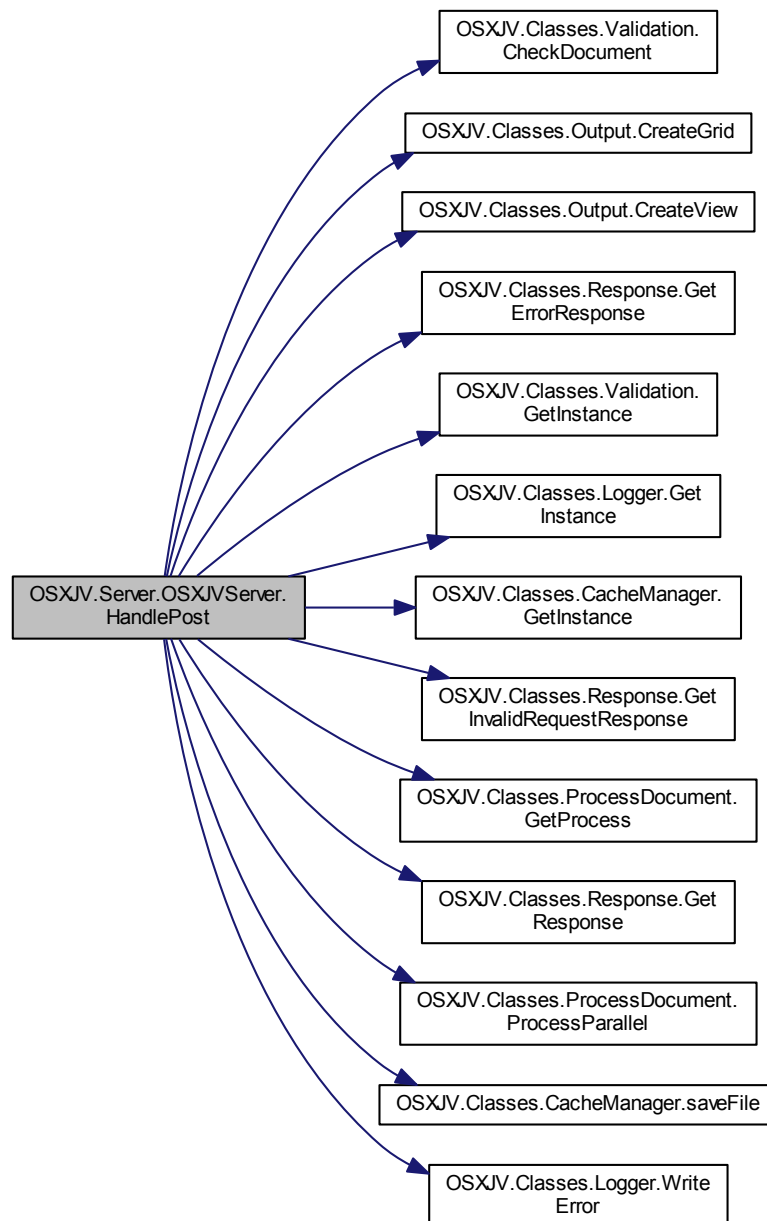
Definition at line 234 of file [OSXJVServer.cs](#).

References [OSXJV.Classes.Validation.CheckDocument\(\)](#), [OSXJV.Classes.Output.CreateGrid\(\)](#), [OSXJV.Classes.Output.CreateView\(\)](#), [OSXJV.Classes.Request.Data](#), [OSXJV.Classes.Response.GetErrorResponse\(\)](#), [OSXJV.Classes.Validation.GetInstance\(\)](#), [OSXJV.Classes.Logger.GetInstance\(\)](#), [OSXJV.Classes.CacheManager.GetInstance\(\)](#), [OSXJV.Classes.Response.GetInvalidRequestResponse\(\)](#), [OSXJV.Classes.ProcessDocument.GetProcess\(\)](#), [OSXJV.Classes.Response.GetResponse\(\)](#), [OSXJV.Classes.ProcessDocument.ProcessParallel\(\)](#), [OSXJV.Classes.CacheManager.saveFile\(\)](#), [OSXJV.Classes.Request.Type](#), and [OSXJV.Classes.Logger.WriteError\(\)](#).

```
00235     {
00236
00237         JObject eRes = new JObject();
00238
00239         if (SegmentNormalize(req.RawUrl).Equals("Process"))
00240         {
00241             if (req.HasEntityBody)
00242             {
00243
00244
00245                 Request r = null;
00246                 try
00247                 {
00248                     r = GetData(req);
00249                     if (r == null)
00250                         return Response.GetInvalidRequestResponse();
00251                 }
00252                 catch
00253                 {
00254                     return Response.GetInvalidRequestResponse();
00255                 }
00256
00257
00258
00259                 try
00260                 {
00261                     Validation.GetInstance().
CheckDocument(r.Data, r.Type);
00262                 }
00263                 catch (Exception e)
00264                 {
00265                     eRes.Add("Error", e.Message);
00266                     return Response.GetErrorResponse(eRes.ToString());
00267                 }
00268
00269                 string id = Guid.NewGuid().ToString();
00270                 ProcessDocument pro = ProcessDocument.
GetProcess(r.Data, r.Type);
00271                 Node n = pro.ProcessParallel();
00272                 Output o = new Output(n); //new output object
00273                 try
00274                 {
00275                     CacheManager.GetInstance().
saveFile(id, JsonConvert.SerializeObject(n));
00276                     JObject response = new JObject();
00277
00278                     n = null; //remove node as its completed;
```

```
00279
00280         response.Add("filename", id);
00281         response.Add("grid", o.CreateGrid());
00282         response.Add("view", o.CreateView());
00283
00284
00285
00286         byte[] bytes = Encoding.UTF8.GetBytes(response.ToString());
00287         return Response.GetResponse(200, "application/json", bytes);
00288     }
00289     catch (Exception e)
00290     {
00291         Logger.GetInstance().WriteError(e.Message);
00292         eRes.Add("Error", "Error Creating Response");
00293         return Response.GetErrorResponse(eRes.ToString());
00294     }
00295
00296     }
00297     eRes.Add("Error", "No File Recieved By Server");
00298     return Response.GetErrorResponse(eRes.ToString());
00299 }
00300 else if (req.RawUrl.Equals("/Output"))
00301 {
00302     return Response.GetInvalidRequestResponse();
00303 }
00304 else
00305     return Response.GetInvalidRequestResponse();
00306 }
```

Here is the call graph for this function:



#### 5.5.3.8 ListenerCallback()

```
void OSXJV.Server.OSXJVServer.ListenerCallback (
    IAsyncResult result ) [private]
```

Handles Requests Asynchronously



## Parameters

<i>result</i>	The Request Object Coming In.
---------------	-------------------------------

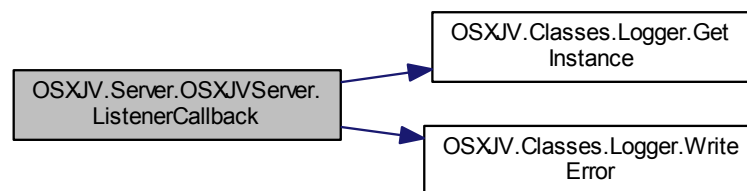
Definition at line 125 of file [OSXJVServer.cs](#).

References [OSXJV.Classes.Logger.GetInstance\(\)](#), and [OSXJV.Classes.Logger.WriteError\(\)](#).

```

00126     {
00127         HttpListener listener = (HttpListener)result.AsyncState;
00128         HttpListenerContext context = listener.EndGetContext(result);
00129         try
00130         {
00131             HandleClient(context);
00132         }
00133         catch (Exception e)
00134         {
00135             Logger.GetInstance().WriteError(e.Message);
00136             context.Response.StatusCode = 500;
00137             context.Response.Close();
00138         }
00139     }
00140 }
```

Here is the call graph for this function:



### 5.5.3.9 ManageCache()

```
void OSXJV.Server.OSXJVServer.ManageCache ( ) [private]
```

Manages cache

Definition at line 446 of file [OSXJVServer.cs](#).

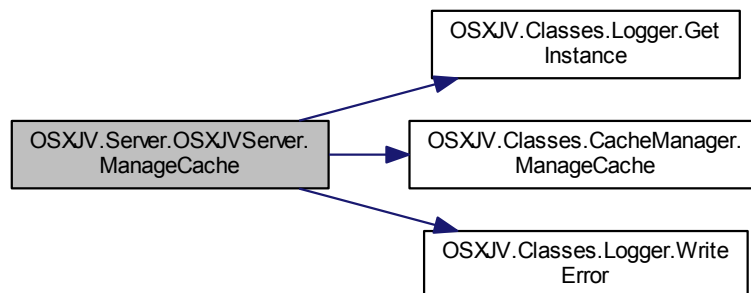
References [OSXJV.Classes.Logger.GetInstance\(\)](#), [OSXJV.Classes.CacheManager.ManageCache\(\)](#), and [OSXJV.Classes.Logger.WriteError\(\)](#).

```

00447     {
00448         while (true)
00449         {
00450             Thread.Sleep(3600000);
00451             try
00452             {
00453                 CacheManager.ManageCache();
00454             }
00455             catch (Exception e)
00456             {
00457                 try
00458                 {
00459                     Logger.GetInstance().WriteError(e.Message);
00460                 }
00461                 catch
00462                 {
00463                     Console.WriteLine("Logger and Cache Manager not setup");
00464                 }
00465             }
00466         }
00467     }
00468 }

```

Here is the call graph for this function:



### 5.5.3.10 Post()

```

void OSXJV.Server.OSXJVServer.Post (
    Response res,
    HttpListenerResponse stream ) [private]

```

Send data to the client.

#### Parameters

<i>res</i>	The Response Object
<i>stream</i>	The Client Output Stream

///

## Exceptions

<i>ArgumentException</i>	Thrown when Response is null or HttpListenerResponse is null or empty
--------------------------	---

Definition at line 395 of file [OSXJVServer.cs](#).

References [OSXJV.Classes.Response.data](#), [OSXJV.Classes.Response.mime](#), and [OSXJV.Classes.Response.status](#).

```

00396         {
00397             if (res == null || stream == null)
00398                 throw new ArgumentException("Response or Client Stream cannot be NULL");
00399
00400             HandleOptions(stream);
00401             stream.ProtocolVersion = new Version(1, 1);
00402             stream.StatusCode = res.status;
00403             stream.ContentType = res.mime;
00404             stream.ContentLength64 = res.data.Length;
00405             stream.OutputStream.Write(res.data, 0, res.data.Length);
00406             stream.Close();
00407         }

```

## 5.5.3.11 Run()

```
void OSXJV.Server.OSXJVServer.Run ( )
```

Function that constantly listens for connections

Definition at line 100 of file [OSXJVServer.cs](#).

```

00101         {
00102             running = true;
00103             listener.Start();
00104
00105             while(listener.IsListening)
00106             {
00107
00108                 Console.WriteLine("Waiting");
00109
00110                 //Wait for Listener
00111                 IAsyncResult result = listener.BeginGetContext(new AsyncCallback(
00112                     ListenerCallback), listener);
00113                 result.AsyncWaitHandle.WaitOne();
00114
00115                 if (result.CompletedSynchronously)
00116                     Console.WriteLine("Completed Synchronously");
00117             }
00118         }

```

## 5.5.3.12 SaveFile()

```
void OSXJV.Server.OSXJVServer.SaveFile (
    string id,
    Node nodes ) [private]
```

Save data recieved from client.

## Parameters

<i>id</i>	Unique ID
<i>nodes</i>	The Processed Data

## Exceptions

<i>ArgumentException</i>	Thrown when nodes is null or id is null or empty
--------------------------	--

Definition at line 372 of file [OSXJVServer.cs](#).

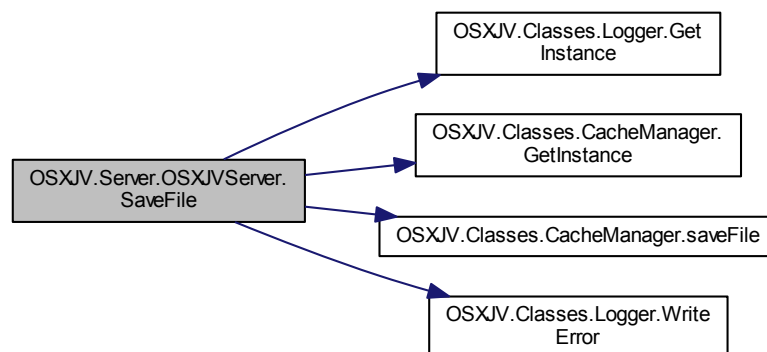
References [OSXJV.Classes.Logger.GetInstance\(\)](#), [OSXJV.Classes.CacheManager.GetInstance\(\)](#), [OSXJV.Classes.CacheManager.saveFile\(\)](#), and [OSXJV.Classes.Logger.WriteError\(\)](#).

```

00373     {
00374         if(nodes == null || string.IsNullOrEmpty(id))
00375         {
00376             throw new ArgumentException();
00377         }
00378
00379         try
00380         {
00381             CacheManager.GetInstance().saveFile(id, JsonConvert.
SerializeObject(nodes));
00382         }
00383         catch (Exception e)
00384         {
00385             Logger.GetInstance().WriteError(e.Message);
00386         }
00387     }

```

Here is the call graph for this function:



### 5.5.3.13 SegmentNormalize()

```

string OSXJV.Server.OSXJVServer.SegmentNormalize (
    string input ) [private]

```

Removes '/' from the string.

## Parameters

<i>input</i>	A string from the URL
--------------	-----------------------

## Returns

Normalised String

Definition at line 438 of file [OSXJVServer.cs](#).

```
00439         {
00440             return input.Replace("/", "");
00441         }
```

## 5.5.3.14 Start()

```
bool OSXJV.Server.OSXJVServer.Start (
    string cachePath,
    string loggerPath )
```

Starts server in new thread

## Parameters

<i>cachePath</i>	Cache Folder Location
<i>loggerPath</i>	Logger Folder Location

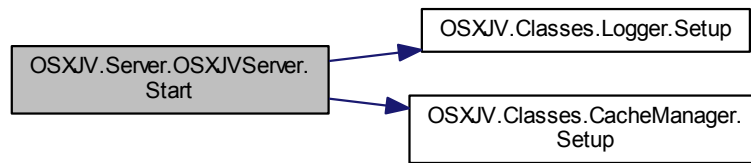
Definition at line 54 of file [OSXJVServer.cs](#).

References [OSXJV.Classes.Logger.Setup\(\)](#), and [OSXJV.Classes.CacheManager.Setup\(\)](#).

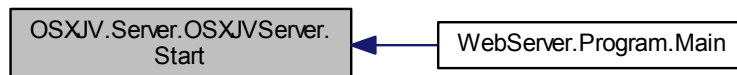
Referenced by [WebServer.Program.Main\(\)](#).

```
00055         {
00056             bool success = false;
00057
00058             success = CacheManager.Setup(cachePath);
00059             success = Logger.Setup(loggerPath);
00060
00061             serverThread = new Thread(new ThreadStart(Run)); //Server thread
00062             cacheThread = new Thread(new ThreadStart(ManageCache)); //Cache manage
00063
00064             thread
00065             {
00066                 try
00067                 {
00068                     serverThread.Start();
00069                     cacheThread.Start();
00070                 }
00071                 catch (Exception e)
00072                 {
00073                     throw e;
00074                 }
00075
00076                 success = cacheThread.IsAlive;
00077                 success = serverThread.IsAlive;
00078
00079                 return success;
00080             }
00081         }
```

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.5.3.15 Stop()

```
bool OSXJV.Server.OSXJVServer.Stop ( )
```

Stop the listener and about all current requests

Definition at line 82 of file [OSXJVServer.cs](#).

```

00083     {
00084         if (listener != null)
00085             if (listener.IsListening)
00086                 listener.Abort();
00087
00088
00089         if (serverThread != null)
00090         {
00091             serverThread.Join();
00092             serverThread = null;
00093         }
00094
00095         return serverThread == null ? true : false;
00096     }
  
```

## 5.5.4 Member Data Documentation

#### 5.5.4.1 cacheThread

```
Thread OSXJV.Server.OSXJVServer.cacheThread = null [private]
```

Cache Controller

Definition at line 38 of file [OSXJVServer.cs](#).

#### 5.5.4.2 listener

```
HttpListener OSXJV.Server.OSXJVServer.listener [private]
```

HttpListener

Definition at line 28 of file [OSXJVServer.cs](#).

#### 5.5.4.3 port

```
int OSXJV.Server.OSXJVServer.port = 8082 [private]
```

Definition at line 18 of file [OSXJVServer.cs](#).

#### 5.5.4.4 running

```
bool OSXJV.Server.OSXJVServer.running = false [static]
```

True if the server is able to accept requests.

Definition at line 23 of file [OSXJVServer.cs](#).

#### 5.5.4.5 serverThread

```
Thread OSXJV.Server.OSXJVServer.serverThread = null [private]
```

[Server](#) Thread

Definition at line 33 of file [OSXJVServer.cs](#).

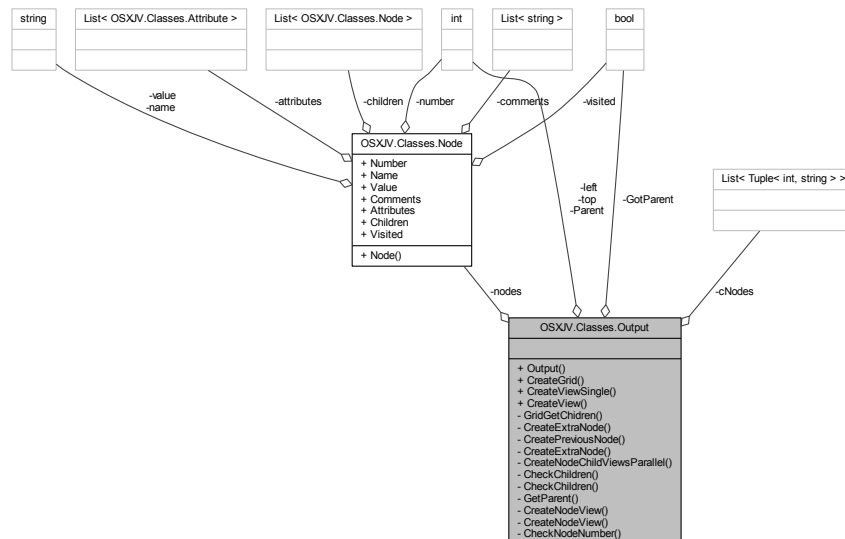
The documentation for this class was generated from the following file:

- [WebServiceCSharp/OSXJVServer.cs](#)

## 5.6 OSXJV.Classes.Output Class Reference

Creates the [Output](#) for the web page to display.

Collaboration diagram for OSXJV.Classes.Output:



### Public Member Functions

- [Output](#) ([Node](#) nodes)  
*Creation of a [Output](#) object.*
- [JObject](#) [CreateGrid](#) ()  
*Creates the grid data.*
- [string](#) [CreateViewSingle](#) (int node, int nodeStart=0)  
*CreateView using a Single Thread*
- [string](#) [CreateView](#) (int node=1, int pCount=4, int nodeStart=0)  
*Creates the view of nodes using multiple threads.*

### Private Member Functions

- [JObject](#) [GridGetChildren](#) ([Node](#) n)  
*Recursive function to get all the nodes data for the grid .*
- [string](#) [CreateExtraNode](#) (string type, int id)  
*Builds a get more button to display*
- [string](#) [CreatePreviousNode](#) (string type, int leftVal, int topVal, int id)  
*Create a previous node button*
- [string](#) [CreateExtraNode](#) (string type, int leftVal, int topVal, int id)  
*Create a extra node button*
- [void](#) [CreateNodeChildViewsParallel](#) (List< [Node](#) > job, int start, bool showHigher, int next, int previous)  
*Generate [Output](#) HTML when using multi-threads*
- [string](#) [CheckChildren](#) ([Node](#) n, int number)



*Check child nodes if the are to be part of the output.*

- string [CheckChildren](#) ([Node](#) n, int number, int pCount, int nodeStart, ref bool found)
- void [GetParent](#) ([Node](#) node, int number)

*Finds the parent node.*

- string [CreateNodeView](#) ([Node](#) n, string type, int leftVal, int topVal)

*Generates HTML for the Specific [Node](#) (Multi-Threaded Version)*

- string [CreateNodeView](#) ([Node](#) n, string type)

*Generates HTML for specific [Node](#) (Single Threaded Version)*

- bool [CheckNodeNumber](#) ([Node](#) n, int number)

*Checks if [Node](#) number and inputted number match.*

## Private Attributes

- int [left](#) = 100
- int [top](#) = 130
- [Node](#) [nodes](#)
- bool [GotParent](#) = false
- int [Parent](#) = 0

*Parent of node when building output (Used when getting [Node](#) other than root).*

- List< Tuple< int, string > > [cNodes](#) = new List<Tuple<int, string>>()

*Used in Threading, list of calculated HTML strings.*

### 5.6.1 Detailed Description

Creates the [Output](#) for the web page to display.

Definition at line 11 of file [Output.cs](#).

### 5.6.2 Constructor & Destructor Documentation

#### 5.6.2.1 Output()

```
OSXJV.Classes.Output.Output (
    Node nodes )
```

Creation of a [Output](#) object.

#### Parameters

<i>nodes</i>	A processed object of Nodes
--------------	-----------------------------

Definition at line 42 of file [Output.cs](#).

```
00043 {
```

```

00044         if (nodes == null)
00045             throw new ArgumentException();
00046         this.nodes = nodes;
00047     }

```

## 5.6.3 Member Function Documentation

### 5.6.3.1 CheckChildren() [1/2]

```

string OSXJV.Classes.Output.CheckChildren (
    Node n,
    int number ) [private]

```

Check child nodes if the are to be part of the output.

#### Parameters

<i>n</i>	Node to search
<i>number</i>	Number to check

#### Returns

String of calculated HTML

Definition at line 399 of file [Output.cs](#).

References [OSXJV.Classes.Node.Children](#), and [OSXJV.Classes.Node.Number](#).

```

00400     {
00401         string output = "";
00402         if (CheckNodeNumber(n, number))
00403         {
00404             int count = 0;
00405             output += CreateNodeView(n, "node");
00406             foreach (Node n2 in n.Children)
00407             {
00408                 count++;
00409                 output += CreateNodeView(n2, "node-child");
00410             }
00411         }
00412         else if (n.Children.Count > 0)
00413         {
00414             foreach (Node n2 in n.Children)
00415             {
00416                 if (GotParent)
00417                 {
00418                     if (n2.Number == Parent)
00419                     {
00420                         output += CreateNodeView(n2, "node-parent");
00421                     }
00422                 }
00423                 output += CheckChildren(n2, number);
00424             }
00425         }
00426     }
00427     return output;
00428 }
00429

```

## 5.6.3.2 CheckChildren() [2/2]

```
string OSXJV.Classes.Output.CheckChildren (
    Node n,
    int number,
    int pCount,
    int nodeStart,
    ref bool found ) [private]
```

## Parameters

<i>n</i>	
<i>number</i>	
<i>pCount</i>	
<i>nodeStart</i>	
<i>found</i>	

## Returns

String of calculated HTML

Definition at line 440 of file [Output.cs](#).

References [OSXJV.Classes.Node.Children](#), and [OSXJV.Classes.Node.Number](#).

```
00441     {
00442         string output = "";
00443         if (CheckNodeNumber(n, number))
00444         {
00445             found = true;
00446             List<Thread> threadList = new List<Thread>();
00447
00448             int count = 0;
00449             output += CreateNodeView(n, "node");
00450             count++;
00451             //output += CreateNodeView(n2, "node-child");
00452             int childCount = 0;
00453
00454             if (n.Children.Count < 200)
00455                 childCount = n.Children.Count;
00456             else
00457             {
00458                 childCount = 200;
00459             }
00460             if (childCount < pCount * 2)
00461             {
00462                 foreach(Node n2 in n.Children)
00463                 {
00464                     output += CreateNodeView(n2, "node-child");
00465                 }
00466             }
00467             else
00468             {
00469                 int spread = (int)Math.Ceiling((double)childCount / (double)pCount);
00470
00471                 if (childCount > 0)
00472                 {
00473                     for (int i = 0; i < pCount; i++)
00474                     {
00475                         int neg = 0;
00476                         if ((spread * (i + 1)) > childCount)
00477                         {
00478                             neg = childCount - (spread * (i + 1));
00479                         }
00480                         int start = (spread * i);
00481                         int rangeStart = (spread * i) + nodeStart;
00482                         bool showHigher = nodeStart != 0 ? true : false;
00483
00484                         List<Node> NodesToProcess = n.Children.GetRange(rangeStart, spread + neg);
00485                     }
00486                 }
00487             }
00488         }
00489     }
```

```

00486             if (NodesToProcess.Count > 0)
00487             {
00488                 Thread threadJob = new Thread(() =>
CreateNodeChildViewsParallel(NodesToProcess, start, showHigher, childCount +
nodeStart, nodeStart - childCount));
00489                 threadJob.Name = i.ToString();
00490                 threadJob.Start();
00491                 threadList.Add(threadJob);
00492             }
00493         }
00494         foreach (Thread t in threadList)
00495         {
00496             t.Join();
00497         }
00498         cNodes.Sort((x, y) => x.Item1.CompareTo(y.Item1));
00499         foreach (Tuple<int, string> tup in cNodes)
00500         {
00501             output += tup.Item2;
00502         }
00503     }
00504 }
00505 }
00506 }
00507 else if (n.Children.Count > 0)
00508 {
00509     foreach (Node n2 in n.Children)
00510     {
00511         if (GotParent)
00512         {
00513             if (n2.Number == Parent)
00514             {
00515                 output += CreateNodeView(n2, "node-parent");
00516             }
00517         }
00518         output += CheckChildren(n2, number, pCount, nodeStart, ref found);
00519     }
00520 }
00521 }
00522 return output;
00523 }

```

### 5.6.3.3 CheckNodeNumber()

```

bool OSXJV.Classes.Output.CheckNodeNumber (
    Node n,
    int number ) [private]

```

Checks if [Node](#) number and inputted number match.

#### Parameters

<i>n</i>	<a href="#">Node</a> to search
<i>number</i>	Number to match with

#### Returns

Definition at line 661 of file [Output.cs](#).

References [OSXJV.Classes.Node.Number](#).

```

00662     {
00663         return n.Number.Equals(number);
00664     }

```

## 5.6.3.4 CreateExtraNode() [1/2]

```
string OSXJV.Classes.Output.CreateExtraNode (
    string type,
    int id ) [private]
```

Builds a get more button to display

## Parameters

<i>type</i>	<a href="#">Node</a> type e.g. 'node-child'
<i>id</i>	The id of the node to start from

## Returns

String of calculated HTML

Definition at line 164 of file [Output.cs](#).

```
00165     {
00166         string node = "";
00167
00168         if (type == "node")
00169         {
00170             if (GotParent)
00171             {
00172                 left = left + 400;
00173             }
00174         }
00175         if (type == "node-child")
00176         {
00177             left = left + 400;
00178         }
00179
00180         node += "<div class='node-child type ui-draggable ui-selecttee' style='left:" +
00181 left + "px; top:" + top + "px;margin-bottom:50px;'>";
00181         node += "<div class='head'><span><button class='nameBtn' onclick='GetMoreNodes(" + id + "
00182 )'>Show Lower</button></span></div>";
00182         node += "</div></div>";
00183         return node;
00184     }
```

## 5.6.3.5 CreateExtraNode() [2/2]

```
string OSXJV.Classes.Output.CreateExtraNode (
    string type,
    int leftVal,
    int topVal,
    int id ) [private]
```

Create a extra node button

## Parameters

<i>type</i>	<a href="#">Node</a> type e.g. 'node-child'
<i>leftVal</i>	Margin from the left of the display
<i>topVal</i>	Margin from the top of the display
<i>id</i>	The id of the node to start from

**Returns**

String of calculated HTML

Definition at line 223 of file [Output.cs](#).

```

00224     {
00225         string node = "";
00226
00227         if (type == "node")
00228         {
00229             if (GotParent)
00230             {
00231                 leftVal = leftVal + 400;
00232             }
00233         }
00234         if (type == "node-child")
00235         {
00236             leftVal = leftVal + 400;
00237         }
00238         node += "<div class='node-child type ui-draggable ui-selectee' style='left:" + leftVal + "px;
top:" + topVal + "px;margin-bottom:50px;'>";
00239         node += "<div class='head'><span><button class='nameBtn' onclick='GetMoreNodes(" + id + "
)'>Show Lower</button></span></div>";
00240         node += "</div></div>";
00241         return node;
00242     }

```

**5.6.3.6 CreateGrid()**

`JSONObject OSXJV.Classes.Output.CreateGrid ( )`

Creates the grid data.

**Returns**

A JSON object

Definition at line 53 of file [Output.cs](#).

References [OSXJV.Classes.Node.Children](#), [OSXJV.Classes.Node.Name](#), and [OSXJV.Classes.Node.Number](#).

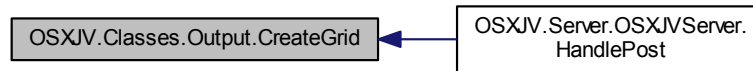
Referenced by [OSXJV.Server.OSXJVServer.HandlePost\(\)](#).

```

00054     {
00055         JSONObject obj = new JSONObject();
00056         obj.Add("text", nodes.Name);
00057         obj.Add("id", nodes.Number);
00058         obj.Add("state", new JSONObject(new JProperty("selected", true)));
00059
00060         if(nodes.Children.Count > 0)
00061         {
00062             JSONArray array = new JSONArray();
00063             foreach (Node n2 in nodes.Children)
00064             {
00065                 array.Add(GridGetChidren(n2));
00066             }
00067             obj.Add("children", array);
00068         }
00069         return obj;
00070     }

```

Here is the caller graph for this function:



### 5.6.3.7 CreateNodeChildViewsParallel()

```

void OSXJV.Classes.Output.CreateNodeChildViewsParallel (
    List< Node > job,
    int start,
    bool showHigher,
    int next,
    int previous ) [private]
  
```

Generate [Output](#) HTML when using multi-threads

#### Parameters

<i>job</i>	The Nodes to process
<i>start</i>	Start index
<i>showHigher</i>	if the are nodes higher up, show previous button
<i>next</i>	Next value for next button
<i>previous</i>	Previous value for previous button

Definition at line [252](#) of file [Output.cs](#).

References [OSXJV.Classes.Node.Comments](#).

```

00253     {
00254         int threadID = int.Parse(Thread.CurrentThread.Name);
00255         string type = "node-child";
00256         string output = "";
00257
00258         if(start == 0 && showHigher)
00259         {
00260             output += CreatePreviousNode(type, left,
00261 top, previous);
00262         }
00263         bool hadCommentsPrev = false;
00264         int numCommentsPrevious = 0;
00265         foreach(Node n in job)
00266         {
00267             int extra = showHigher ? 130 * (start +1) : 130 * start;
00268
00269             if (hadCommentsPrev)
00270                 extra += (numCommentsPrevious * 25);
00271
00272             if (n.Comments.Count > 0)
00273             {
00274                 hadCommentsPrev = true;
00275                 numCommentsPrevious = n.Comments.Count;
  
```

```

00276         }
00277         else
00278             hadCommentsPrev = false;
00279
00280         output += CreateNodeView(n, type, left, top + extra);
00281         start++;
00282         if (start == 200)
00283         {
00284             output += CreateExtraNode(type, left, top + extra + 130, next);
00285             break;
00286         }
00287     }
00288 }
00289
00290 cNodes.Add(new Tuple<int, string>(threadID, output));
00291 }

```

### 5.6.3.8 CreateNodeView() [1/2]

```

string OSXJV.Classes.Output.CreateNodeView (
    Node n,
    string type,
    int leftVal,
    int topVal ) [private]

```

Generates HTML for the Specific [Node](#) (Multi-Threaded Version)

#### Parameters

<i>n</i>	<a href="#">Node</a> to parse
<i>type</i>	Type of node
<i>leftVal</i>	Margin left of display
<i>topVal</i>	Margin top of display

#### Returns

String of calculated HTML

Definition at line 557 of file [Output.cs](#).

References [OSXJV.Classes.Node.Attributes](#), [OSXJV.Classes.Node.Comments](#), [OSXJV.Classes.Attribute.Name](#), [OSXJV.Classes.Node.Name](#), [OSXJV.Classes.Node.Number](#), [OSXJV.Classes.Attribute.Value](#), and [OSXJV.Classes.Node.Value](#).

```

00558     {
00559         string node = "";
00560
00561         if(type == "node")
00562         {
00563             if(GotParent)
00564             {
00565                 leftVal = leftVal + 400;
00566             }
00567         }
00568         if(type == "node-child")
00569         {
00570             leftVal = leftVal + 400;
00571         }
00572         node += "<div id='" + n.Number + "' class='" + type + " type ui-draggable ui-selectee'
style='left:" + leftVal + "px; top:" + topVal + "px;'>";
00573         node += "<div class='head'><span><button class='nameBtn' onclick='GetNode(\"+n.Number+\")'>\" + n.

```



```

        Name + "</button></span></div>";
00574         if (!string.IsNullOrEmpty(n.Value))
00575         {
00576             node += string.Format("<div class='blockR'><p>Value</p></div><div
class=comment><span>{0}</span></div>", n.Value);
00577         }
00578         if(n.Comments.Count >0)
00579         {
00580             node += "<div><p class='text-center'>Comments</p></div>";
00581             foreach(string com in n.Comments)
00582             {
00583                 node += "<div class='comment'>" + com + "</div>";
00584             }
00585         }
00586         if (n.Attributes.Count > 0)
00587         {
00588             node += "<div class='attribute'><div class='aHeader'><p><button><i class='fa
fa-plus'></i></button>Attributes</p></div><div class='options'>";
00589             foreach (Attribute a in n.Attributes)
00590             {
00591                 node += string.Format("<div class='blockR'><p>{0}</p></div><div
class='comment'><p>{1}</p></div>", a.Name, a.Value);
00592             }
00593             node += "</div>";
00594         }
00595         node += "</div></div>";
00596         return node;
00597     }
00598 }

```

### 5.6.3.9 CreateNodeView() [2/2]

```

string OSXJV.Classes.Output.CreateNodeView (
    Node n,
    string type ) [private]

```

Generates HTML for specific [Node](#) (Single Threaded Version)

#### Parameters

<i>n</i>	<a href="#">Node</a> to parse
<i>type</i>	Type of node

#### Returns

String of calculated HTML

Definition at line 606 of file [Output.cs](#).

References [OSXJV.Classes.Node.Attributes](#), [OSXJV.Classes.Node.Comments](#), [OSXJV.Classes.Attribute.Name](#), [OSXJV.Classes.Node.Name](#), [OSXJV.Classes.Node.Number](#), [OSXJV.Classes.Attribute.Value](#), and [OSXJV.Classes.Node.Value](#).

```

00607     {
00608         string node = "";
00609         int leftVal = left;
00610         if (type == "node")
00611         {
00612             if (GotParent)
00613             {
00614                 left = left + 400;
00615                 leftVal = left;
00616             }

```

```

00617         }
00618         if (type == "node-child")
00619         {
00620             leftVal = leftVal + 400;
00621         }
00622         node += "<div id='" + n.Number + "' class='" + type + " type ui-draggable ui-selectee'
style='left:" + leftVal + "px; top:" + top + "px;'>";
00623         node += "<div class='head'><span><button class='nameBtn' onclick='GetNode(" + n.Number + ")'>"
+ n.Name + "</button></span></div>";
00624         if (!string.IsNullOrEmpty(n.Value))
00625         {
00626             node += string.Format("<div class='blockR'><p>Value</p></div><div
class=comment><span>{0}</span></div>", n.Value);
00627         }
00628         if (n.Comments.Count > 0)
00629         {
00630             node += "<div><p class='text-center'>Comments</p></div>";
00631
00632             foreach (string com in n.Comments)
00633             {
00634                 node += "<div class='comment'>" + com + "</div>";
00635             }
00636         }
00637         if (n.Attributes.Count > 0)
00638         {
00639             node += "<div class='attribute'><div class='aHeader'><p><button><i class='fa
fa-plus'></i></button>Attributes</p></div><div class='options'>";
00640             foreach (Attribute a in n.Attributes)
00641             {
00642                 node += string.Format("<div class='blockR'><p>{0}</p></div><div
class='comment'><p>{1}</p></div>", a.Name, a.Value);
00643             }
00644             node += "</div>";
00645         }
00646         node += "</div></div>";
00647
00648         if (type == "node-child")
00649         {
00650             top = top + 130;
00651         }
00652         return node;
00653     }

```

### 5.6.3.10 CreatePreviousNode()

```

string OSXJV.Classes.Output.CreatePreviousNode (
    string type,
    int leftVal,
    int topVal,
    int id ) [private]

```

Create a previous node button

#### Parameters

<i>type</i>	Node type e.g. 'node-child'
<i>leftVal</i>	Margin from the left of the display
<i>topVal</i>	Margin from the top of the display
<i>id</i>	The id of the node to start from

#### Returns

String of calculated HTML

Definition at line 194 of file [Output.cs](#).

```

00195     {
00196         string node = "";
00197
00198         if (type == "node")
00199         {
00200             if (GotParent)
00201             {
00202                 leftVal = leftVal + 400;
00203             }
00204         }
00205         if (type == "node-child")
00206         {
00207             leftVal = leftVal + 400;
00208         }
00209         node += "<div class='node-child type ui-draggable ui-selectee' style='left:" + leftVal + "px;
top:" + topVal + "px;'>";
00210         node += "<div class='head'><span><button class='nameBtn' onclick='GetMoreNodes(" + id + "
)'>Show Higher</button></span></div>";
00211         node += "</div></div>";
00212         return node;
00213     }

```

### 5.6.3.11 CreateView()

```

string OSXJV.Classes.Output.CreateView (
    int node = 1,
    int pCount = 4,
    int nodeStart = 0 )

```

Creates the view of nodes using multiple threads.

#### Parameters

<i>node</i>	Number of node to start from. Default is 1(Root)
<i>pCount</i>	Number of Threads to use. Default is 4
<i>nodeStart</i>	Where to start the child nodes from

#### Returns

String of calculated HTML

Definition at line 300 of file [Output.cs](#).

References [OSXJV.Classes.Node.Children](#), and [OSXJV.Classes.Node.Number](#).

Referenced by [OSXJV.Server.OSXJVServer.HandleGet\(\)](#), and [OSXJV.Server.OSXJVServer.HandlePost\(\)](#).

```

00301     {
00302
00303         List<Thread> threadList = new List<Thread>();
00304
00305         string output = "<div class='text-center ui-layout-center ui-layout-pane
ui-layout-pane-center'><div style ='display:inline-block' class='ui-selectable ui-droppable'>";
00306         if (nodes.Number.Equals(node))
00307         {
00308             int childCount = 0;
00309
00310             if (nodes.Children.Count < 200)
00311                 childCount = nodes.Children.Count;
00312             else
00313             {
00314                 childCount = 200;

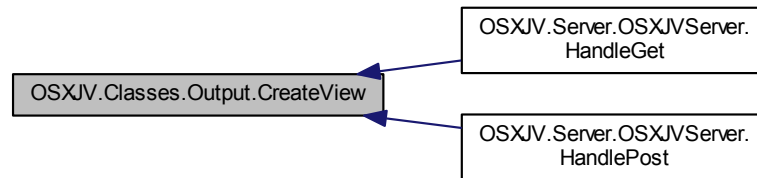
```

```

00315         }
00316
00317         if(childCount < pCount * 2)
00318         {
00319             output += CreateNodeView(nodes, "node",
00320 left, top);
00321             foreach(Node n2 in nodes.Children)
00322             {
00323                 output += CreateNodeView(n2, "node-child");
00324             }
00325         }
00326         else
00327         {
00328             int spread = (int)Math.Ceiling((double)childCount / (double)pCount);
00329             output += CreateNodeView(nodes, "node",
00330 left,top); //Parent(Node) Thread
00331             for (int i = 0; i < pCount; i++)
00332             {
00333                 int neg = 0;
00334                 if ((spread * (i + 1)) > childCount)
00335                 {
00336                     neg = childCount - (spread * (i + 1));
00337                 }
00338                 int start = (spread * i) ;
00339                 int rangeStart = (spread * i) + nodeStart;
00340                 bool showHigher = nodeStart != 0 ? true : false;
00341
00342                 List<Node> NodesToProcess = nodes.Children.GetRange(rangeStart, spread
+ neg);
00343                 Thread threadJob = new Thread(() =>
CreateNodeChildViewsParallel(NodesToProcess, start, showHigher, childCount +
nodeStart, nodeStart - childCount));
00344                 threadJob.Name = i.ToString();
00345                 threadJob.Start();
00346                 threadList.Add(threadJob);
00347             }
00348             foreach(Thread t in threadList)
00349             {
00350                 t.Join();
00351             }
00352
00353             cNodes.Sort((x, y) => x.Item1.CompareTo(y.Item1));
00354
00355             foreach(Tuple<int,string> tup in cNodes)
00356             {
00357                 output += tup.Item2;
00358             }
00359         }
00360     }
00361     else
00362     {
00363         GetParent(nodes, node);
00364         string temp = "";
00365         if (GotParent)
00366         {
00367             if (nodes.Number == Parent)
00368             {
00369                 output += CreateNodeView(nodes, "node-parent");
00370             }
00371         }
00372         bool found =false;
00373         foreach (Node n2 in nodes.Children)
00374         {
00375             if (GotParent)
00376             {
00377                 if (n2.Number == Parent)
00378                 {
00379                     output += CreateNodeView(n2, "node-parent");
00380                 }
00381             }
00382             temp += CheckChildren(n2, node,pCount,nodeStart,ref found);
00383             if (found)
00384                 break;
00385         }
00386         if (!string.IsNullOrEmpty(temp))
00387             output += temp;
00388     }
00389     output += "</div></div>";
00390     return output;
00391 }

```

Here is the caller graph for this function:



### 5.6.3.12 CreateViewSingle()

```
string OSXJV.Classes.Output.CreateViewSingle (
    int node,
    int nodeStart = 0 )
```

CreateView using a Single Thread

#### Parameters

<i>node</i>	Index of node to start from
<i>nodeStart</i>	Where to start the child nodes from

#### Returns

String of calculated HTML

Definition at line 101 of file [Output.cs](#).

References [OSXJV.Classes.Node.Children](#), and [OSXJV.Classes.Node.Number](#).

```

00102         {
00103             string output = "<div class='text-center ui-layout-center ui-layout-pane
00104 ui-layout-pane-center'><div style ='display:inline-block' class='ui-selectable ui-droppable'>";
00105             if (nodes.Number.Equals(node))
00106             {
00107                 int count = 0;
00108                 output += CreateNodeView(nodes, "node");
00109
00110                 foreach (Node n in nodes.Children)
00111                 {
00112                     if (nodeStart > 0)
00113                     {
00114                         if (count != nodeStart)
00115                             continue;
00116                     }
00117                     count++;
00118                     output += CreateNodeView(n, "node-child"); //Child(Nodes) Thread
00119
00120                     if ((count-nodeStart) == 200)
00121                     {
00122
```

```

00123             output += CreateExtraNode("node-child",count);
00124             break;
00125         }
00126     }
00127 }
00128
00129 else
00130 {
00131     GetParent(nodes, node);
00132     string temp = "";
00133     if (GotParent)
00134     {
00135         if (nodes.Number == Parent)
00136         {
00137             output += CreateNodeView(nodes, "node-parent");
00138         }
00139     }
00140     foreach (Node n2 in nodes.Children)
00141     {
00142         if (GotParent)
00143         {
00144             if (n2.Number == Parent)
00145             {
00146                 output += CreateNodeView(n2, "node-parent");
00147             }
00148         }
00149         temp += CheckChildren(n2, node);
00150     }
00151     if (!string.IsNullOrEmpty(temp))
00152         output += temp;
00153 }
00154 output += "</div></div>"; //Close out divs
00155 return output;
00156 }

```

### 5.6.3.13 GetParent()

```

void OSXJV.Classes.Output.GetParent (
    Node node,
    int number ) [private]

```

Finds the parent node.

#### Parameters

<i>node</i>	<a href="#">Node</a> to search
<i>number</i>	<a href="#">Node</a> number to find

Definition at line 530 of file [Output.cs](#).

References [OSXJV.Classes.Node.Children](#), and [OSXJV.Classes.Node.Number](#).

```

00531     {
00532         if (!CheckNodeNumber(node,number))
00533         {
00534             foreach (Node n in node.Children)
00535             {
00536                 if (CheckNodeNumber(n, number))
00537                 {
00538                     Parent = node.Number;
00539                     GotParent = true;
00540                 }
00541             }
00542             else
00543             {
00544                 GetParent(n, number);
00545             }
00546         }
00547     }

```

### 5.6.3.14 GridGetChidren()

```
JsonObject OSXJV.Classes.Output.GridGetChidren (
    Node n ) [private]
```

Recursive function to get all the nodes data for the grid .

#### Parameters

<i>n</i>	Child <a href="#">Node</a>
----------	----------------------------

#### Returns

JSON object

Definition at line 77 of file [Output.cs](#).

References [OSXJV.Classes.Node.Children](#), [OSXJV.Classes.Node.Name](#), and [OSXJV.Classes.Node.Number](#).

```
00078     {
00079         JObject child = new JObject();
00080         child.Add("id", n.Number);
00081         child.Add("text", n.Name);
00082
00083         if (n.Children.Count > 0)
00084         {
00085             JArray array = new JArray();
00086             foreach (Node n2 in n.Children)
00087             {
00088                 array.Add(GridGetChidren(n2));
00089             }
00090             child.Add("children", array);
00091         }
00092         return child;
00093     }
```

## 5.6.4 Member Data Documentation

### 5.6.4.1 cNodes

```
List<Tuple<int, string> > OSXJV.Classes.Output.cNodes = new List<Tuple<int, string>>()
[private]
```

Used in Threading, list of calculated HTML strings.

Definition at line 36 of file [Output.cs](#).

### 5.6.4.2 GotParent

```
bool OSXJV.Classes.Output.GotParent = false [private]
```

Definition at line 26 of file [Output.cs](#).

### 5.6.4.3 left

```
int OSXJV.Classes.Output.left = 100 [private]
```

Definition at line 16 of file [Output.cs](#).

### 5.6.4.4 nodes

```
Node OSXJV.Classes.Output.nodes [private]
```

Definition at line 21 of file [Output.cs](#).

### 5.6.4.5 Parent

```
int OSXJV.Classes.Output.Parent = 0 [private]
```

Parent of node when building output (Used when getting [Node](#) other than root).

Definition at line 31 of file [Output.cs](#).

### 5.6.4.6 top

```
int OSXJV.Classes.Output.top = 130 [private]
```

Definition at line 16 of file [Output.cs](#).

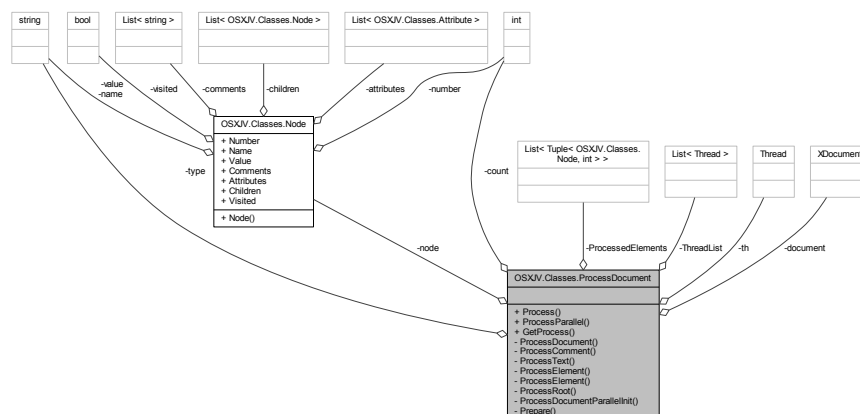
The documentation for this class was generated from the following file:

- [WebServiceCSharp/OSXJVClasses/Output.cs](#)

## 5.7 OSXJV.Classes.ProcessDocument Class Reference

Class the Processes the document

Collaboration diagram for OSXJV.Classes.ProcessDocument:





## Public Member Functions

- [Node Process](#) ()  
*Single Threaded Process.*
- [Node ProcessParallel](#) (int pCount=4)  
*Parse Document Using Multiple Threads*

## Static Public Member Functions

- static [ProcessDocument GetProcess](#) (string data, string [type](#))  
*Gets an instance of the [ProcessDocument](#) and prepare object.*

## Private Member Functions

- [ProcessDocument](#) (XDocument doc, string [type](#))  
*Constructor*
- void [ProcessComment](#) (XComment e, [Node node](#))  
*Extract Comment*
- void [ProcessText](#) (XText e, [Node n](#))  
*Get text from the data*
- [Node ProcessElement](#) (XElement e, [Node node](#))  
*Single Threaded Process Element Version*
- [Node ProcessElement](#) (XElement e, [Node node](#), ref int nodeNumber)  
*Multi-Threaded Version to process element*
- [Node ProcessRoot](#) (XElement e, [Node node](#))  
*Processes first element in the document.*
- void [ProcessDocumentParallelInit](#) (XDocument doc, int start)  
*Method that each thread uses to process the document*

## Static Private Member Functions

- static XDocument [Prepare](#) (string data, string [type](#))  
*Prepares the object with setting the XDocument object to process*

## Private Attributes

- XDocument [document](#)  
*Object the contains the parsed data ready to be processed.*
- [Node node](#) = new [Node](#)()  
*The Initial [Node](#).*
- List< Tuple< [Node](#), int > > [ProcessedElements](#) = new List<Tuple<[Node](#), int>>()  
*Used with threading to keep list of processed Nodes.*
- List< Thread > [ThreadList](#) = new List<Thread>()  
*Used with threading to keep list of running threads.*
- string [type](#)  
*Document Type.*
- Thread [th](#)
- int [count](#)  
*Used to by single thread operation to keep track of node id.*

### 5.7.1 Detailed Description

Class the Processes the document

Definition at line 15 of file [ProcessDocument.cs](#).

### 5.7.2 Constructor & Destructor Documentation

#### 5.7.2.1 ProcessDocument()

```
OSXJV.Classes.ProcessDocument.ProcessDocument (
    XDocument doc,
    string type ) [private]
```

Constructor

Parameters

<i>doc</i>	Parsed document
<i>type</i>	Type of document

Definition at line 53 of file [ProcessDocument.cs](#).

```
00054     {
00055         document = doc;
00056         this.type = type;
00057     }
```

### 5.7.3 Member Function Documentation

#### 5.7.3.1 GetProcess()

```
static ProcessDocument OSXJV.Classes.ProcessDocument.GetProcess (
    string data,
    string type ) [static]
```

Gets an instance of the [ProcessDocument](#) and prepare object.

Parameters

<i>data</i>	String of the document
<i>type</i>	Type of document

**Returns**

Definition at line 77 of file [ProcessDocument.cs](#).

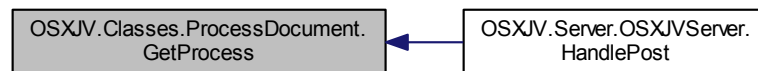
Referenced by [OSXJV.Server.OSXJVServer.HandlePost\(\)](#).

```

00078         {
00079             if (string.IsNullOrEmpty(data) || string.IsNullOrEmpty(type))
00080             {
00081                 throw new ArgumentException();
00082             }
00083             try
00084             {
00085                 XDocument doc = null;
00086                 doc = Prepare(data, type);
00087                 return new ProcessDocument(doc, type);
00088             }
00089             catch (System.Xml.XmlException e)
00090             {
00091                 throw e;
00092             }
00093         }

```

Here is the caller graph for this function:

**5.7.3.2 Prepare()**

```

static XDocument OSXJV.Classes.ProcessDocument.Prepare (
    string data,
    string type ) [static], [private]

```

Prepares the object with setting the XDocument object to process

**Parameters**

<i>data</i>	String of data
<i>type</i>	Data type

**Returns**

A XDocument object

Definition at line 111 of file [ProcessDocument.cs](#).

```

00112     {
00113
00114         if (type.Equals("JSON"))
00115             return new XDocument(JsonConvert.DeserializeXmlNode(data, "Root", false).Root.FirstNode);
00116         else if (type.Equals("XML") || type.Equals("HTML"))
00117             return XDocument.Parse(data);
00118
00119         return null;
00120     }

```

### 5.7.3.3 Process()

[Node](#) OSXJV.Classes.ProcessDocument.Process ( )

Single Threaded Process.

#### Returns

Object of Nodes

Definition at line 127 of file [ProcessDocument.cs](#).

```

00128     {
00129         if (document.Nodes() != null)
00130         {
00131             foreach (XmlNode n in document.Nodes())
00132             {
00133                 switch (n.NodeType)
00134                 {
00135                     case System.Xml.XmlNodeType.Element:
00136                         count++;
00137                         ProcessElement(XElement.Parse(n.ToString()),
node);
00138                         break;
00139                     case System.Xml.XmlNodeType.Comment:
00140                         ProcessComment(n as XComment, node);
00141                         break;
00142                     case System.Xml.XmlNodeType.Text:
00143                         ProcessText(n as XText, node);
00144                         break;
00145                     case System.Xml.XmlNodeType.Notation:
00146                         break;
00147                     case System.Xml.XmlNodeType.EndElement:
00148                         break;
00149                     default:
00150                         break;
00151                 }
00152             }
00153         }
00154         //SortArray(ref node);
00155         document = null;
00156         return node;
00157     }

```

### 5.7.3.4 ProcessComment()

```

void OSXJV.Classes.ProcessDocument.ProcessComment (
    XComment e,
    Node node ) [private]

```

Extract Comment

## Parameters

<i>e</i>	Comment object to be parsed
<i>node</i>	<a href="#">Node</a> to input data

Definition at line 64 of file [ProcessDocument.cs](#).

References [OSXJV.Classes.Node.Comments](#).

```

00065     {
00066         string s = "";
00067         s = Regex.Replace(e.Value, @"^\w\s\.-", "");
00068         node.Comments.Add(s);
00069     }

```

## 5.7.3.5 ProcessDocumentParallelInit()

```

void OSXJV.Classes.ProcessDocument.ProcessDocumentParallelInit (
    XDocument doc,
    int start ) [private]

```

Method that each thread uses to process the document

## Parameters

<i>doc</i>	A subset of the full document
<i>start</i>	Start index number

Definition at line 336 of file [ProcessDocument.cs](#).

References [OSXJV.Classes.Node.Children](#).

```

00337     {
00338         int nodeNum = start;
00339
00340         Node node = new Node();
00341         if (doc.Root.Nodes() != null)
00342         {
00343             List<XNode> list = doc.Root.Nodes().ToList();
00344             foreach (XNode n in doc.Root.Nodes())
00345             {
00346                 switch (n.NodeType)
00347                 {
00348                     case System.Xml.XmlNodeType.Element:
00349                         nodeNum++;
00350                         Node n2 = new Node();
00351                         node.Children.Add(ProcessElement(XElement.Parse(n.ToString()), n2
00352 , ref nodeNum));
00353                         break;
00354                     case System.Xml.XmlNodeType.Comment:
00355                         ProcessComment(n as XComment, node);
00356                         break;
00357                     case System.Xml.XmlNodeType.Text:
00358                         ProcessText(n as XText, node);
00359                         break;
00360                     case System.Xml.XmlNodeType.Nototation:
00361                         break;
00362                     case System.Xml.XmlNodeType.EndElement:
00363                         break;
00364                     default:

```

```

00364             break;
00365         }
00366     }
00367 }
00368 document = null;
00369 ProcessedElements.Add(new Tuple<Node, int>(node, start));
00370 }

```

### 5.7.3.6 ProcessElement() [1/2]

```

Node OSXJV.Classes.ProcessDocument.ProcessElement (
    XElement e,
    Node node ) [private]

```

Single Threaded Process Element Version

#### Parameters

<i>e</i>	Element to Process
<i>node</i>	The <a href="#">Node</a> to fill data with

#### Returns

Definition at line 165 of file [ProcessDocument.cs](#).

References [OSXJV.Classes.Node.Attributes](#), [OSXJV.Classes.Node.Children](#), [OSXJV.Classes.Attribute.Name](#), [OSXJV.Classes.Node.Name](#), [OSXJV.Classes.Node.Number](#), [OSXJV.Classes.Attribute.Value](#), and [OSXJV.Classes.Node.Visited](#).

```

00166     {
00167         if (node.Number == 0)
00168         {
00169             node.Number = count;
00170         }
00171         if (!node.Visited)
00172         {
00173             node.Name = e.Name.LocalName;
00174             foreach (XAttribute ax in e.Attributes())
00175             {
00176                 if (ax.Name == "id")
00177                 {
00178                     node.Name = node.Name + " #" + ax.Value;
00179                 }
00180             }
00181             if (type == "HTML")
00182             {
00183                 if (ax.IsNamespaceDeclaration)
00184                     continue;
00185             }
00186             Attribute att = new Attribute();
00187             att.Name = ax.Name.LocalName;
00188             att.Value = ax.Value;
00189             node.Attributes.Add(att);
00190         }
00191     }
00192 }
00193 if (e.Nodes() != null)
00194 {
00195     foreach (XNode n in e.Nodes())
00196     {
00197         switch (n.NodeType)
00198     }

```

```

00199         {
00200             case System.Xml.XmlNodeType.EndElement:
00201                 break;
00202             case System.Xml.XmlNodeType.Element:
00203                 count++;
00204                 Node n2 = new Node();
00205                 node.Children.Add(ProcessElement (XElement.Parse(n.
ToString()), n2));
00206                 break;
00207             case System.Xml.XmlNodeType.Comment:
00208                 ProcessComment (n as XComment, node);
00209                 break;
00210             case System.Xml.XmlNodeType.Text:
00211                 ProcessText (n as XText, node);
00212                 break;
00213             case System.Xml.XmlNodeType.Notation:
00214                 break;
00215
00216             default:
00217                 break;
00218         }
00219     }
00220 }
00221 node.Visited = true;
00222 return node;
00223 }

```

### 5.7.3.7 ProcessElement() [2/2]

```

Node OSXJV.Classes.ProcessDocument.ProcessElement (
    XElement e,
    Node node,
    ref int nodeNumber ) [private]

```

Multi-Threaded Version to process element

#### Parameters

<i>e</i>	Element to process
<i>node</i>	<a href="#">Node</a> to extract data from
<i>nodeNumber</i>	The Thread internal node number

#### Returns

Definition at line 232 of file [ProcessDocument.cs](#).

References [OSXJV.Classes.Node.Attributes](#), [OSXJV.Classes.Node.Children](#), [OSXJV.Classes.Attribute.Name](#), [OSXJV.Classes.Node.Name](#), [OSXJV.Classes.Node.Number](#), [OSXJV.Classes.Attribute.Value](#), and [OSXJV.Classes.Node.Visited](#).

```

00233     {
00234         if (!node.Visited)
00235         {
00236             if (node.Number == 0)
00237             {
00238                 node.Number = nodeNumber;
00239             }
00240             if (!node.Visited)
00241             {
00242

```

```

00243         node.Name = e.Name.LocalName;
00244         foreach (XmlAttribute ax in e.Attributes())
00245         {
00246             if (ax.Name == "id")
00247             {
00248                 node.Name = node.Name + " #" + ax.Value;
00249             }
00250
00251             if (type == "HTML")
00252             {
00253                 if (ax.IsNamespaceDeclaration)
00254                     continue;
00255             }
00256             Attribute att = new Attribute();
00257             att.Name = ax.Name.LocalName;
00258             att.Value = ax.Value;
00259             node.Attributes.Add(att);
00260         }
00261     }
00262
00263     if (e.Nodes() != null)
00264     {
00265         foreach (XmlNode n in e.Nodes())
00266         {
00267             switch (n.NodeType)
00268             {
00269                 case System.Xml.XmlNodeType.EndElement:
00270                     break;
00271                 case System.Xml.XmlNodeType.Element:
00272                     nodeNumber++;
00273                     Node n2 = new Node();
00274                     node.Children.Add(ProcessElement(XElement.Parse(n
00275                         .ToString()), n2, ref nodeNumber));
00276                     break;
00277                 case System.Xml.XmlNodeType.Comment:
00278                     ProcessComment(n as XComment, node);
00279                     break;
00280                 case System.Xml.XmlNodeType.Text:
00281                     ProcessText(n as XText, node);
00282                     break;
00283                 case System.Xml.XmlNodeType.Notation:
00284                     break;
00285                 default:
00286                     break;
00287             }
00288         }
00289     }
00290     node.Visited = true;
00291     return node;
00292 }
00293

```

### 5.7.3.8 ProcessParallel()

```

Node OSXJV.Classes.ProcessDocument.ProcessParallel (
    int pCount = 4 )

```

#### Parse Document Using Multiple Threads

##### Parameters

<i>pCount</i>	Number of Threads to run Default = 4
---------------	--------------------------------------

##### Returns

A object of [Node](#) that has been processed

Definition at line 377 of file [ProcessDocument.cs](#).



References [OSXJV.Classes.Node.Children](#).

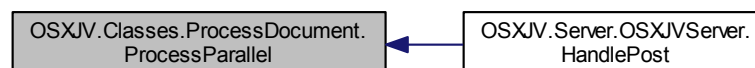
Referenced by [OSXJV.Server.OSXJVServer.HandlePost\(\)](#).

```

00378     {
00379         node = ProcessRoot(document.Root, node);
00380
00381         int nodeCount = document.Root.Nodes().Count();
00382
00383         if(nodeCount <= pCount)
00384         {
00385             return Process();
00386         }
00387         else if (nodeCount > pCount)
00388         {
00389
00390             List<XNode> List = document.Root.Nodes().ToList();
00391             int spread = 0;
00392
00393             spread = (int)Math.Ceiling((double)nodeCount / (double)pCount);
00394
00395             int totalNodes = 1;
00396
00397             for (int i = 0; i < pCount; i++)
00398             {
00399                 int neg = 0;
00400                 int start = totalNodes;
00401                 if ((spread * (i+1)) > nodeCount)
00402                 {
00403                     neg = nodeCount - (spread * (i + 1));
00404                 }
00405
00406                 List<XNode> list = List.GetRange((spread * i), spread + neg);
00407                 XElement root = new XElement("Root", list);
00408                 XDocument doc = new XDocument(root);
00409
00410                 (th = new Thread(() => ProcessDocumentParallelInit(doc,
00411 start)).Start();
00412
00413                 ThreadList.Add(th); //Add to Threads list to keep recored of threads
00414
00415                 running
00416                     totalNodes += root.Descendants().Count(); //Increment start position.
00417                 }
00418                 document = null;
00419                 foreach (Thread t in ThreadList)
00420                 {
00421                     t.Join(); //Wait for threads to join
00422                 }
00423                 ProcessedElements.Sort((x, y) => x.Item2.CompareTo(y.Item2)); //Sort List
00424                 by start index so they are in order.
00425
00426                 foreach (Tuple<Node,int> tup in ProcessedElements)
00427                 {
00428                     foreach (Node n in tup.Item1.Children)
00429                     {
00430                         node.Children.Add(n);
00431                     }
00432                 }
00433                 return node;
00434             }
00435         }

```

Here is the caller graph for this function:



### 5.7.3.9 ProcessRoot()

```
Node OSXJV.Classes.ProcessDocument.ProcessRoot (
    XElement e,
    Node node ) [private]
```

Processes first element in the document.

#### Parameters

<i>e</i>	Element object to process
<i>node</i>	<a href="#">Node</a> to insert data to

#### Returns

Definition at line 301 of file [ProcessDocument.cs](#).

References [OSXJV.Classes.Node.Attributes](#), [OSXJV.Classes.Attribute.Name](#), [OSXJV.Classes.Node.Name](#), [OSXJV.Classes.Node.Number](#), [OSXJV.Classes.Attribute.Value](#), and [OSXJV.Classes.Node.Visited](#).

```
00302     {
00303         node.Number = 1;
00304
00305         if (!node.Visited)
00306         {
00307
00308             node.Name = e.Name.LocalName;
00309             foreach (XAttribute ax in e.Attributes())
00310             {
00311                 if (ax.Name == "id")
00312                 {
00313                     node.Name = node.Name + " #" + ax.Value;
00314                 }
00315
00316                 if (type == "HTML")
00317                 {
00318                     if (ax.IsNamespaceDeclaration)
00319                         continue;
00320
00321                     Attribute att = new Attribute();
00322                     att.Name = ax.Name.LocalName;
00323                     att.Value = ax.Value;
00324                     node.Attributes.Add(att);
00325                 }
00326             }
00327             node.Visited = true;
00328             return node;
00329         }
00330     }
```

### 5.7.3.10 ProcessText()

```
void OSXJV.Classes.ProcessDocument.ProcessText (
    XText e,
    Node n ) [private]
```

Get text from the data

## Parameters

<i>e</i>	Text Element
<i>n</i>	<a href="#">Node</a> to input data

Definition at line 100 of file [ProcessDocument.cs](#).

References [OSXJV.Classes.Node.Value](#).

```
00101         {  
00102             n.Value = e.Value;  
00103         }
```

## 5.7.4 Member Data Documentation

### 5.7.4.1 count

```
int OSXJV.Classes.ProcessDocument.count [private]
```

Used to by single thread operation to keep track of node id.

Definition at line 46 of file [ProcessDocument.cs](#).

### 5.7.4.2 document

```
XDocument OSXJV.Classes.ProcessDocument.document [private]
```

Object the contains the parsed data ready to be processed.

Definition at line 20 of file [ProcessDocument.cs](#).

### 5.7.4.3 node

```
Node OSXJV.Classes.ProcessDocument.node = new Node() [private]
```

The Initial [Node](#).

Definition at line 25 of file [ProcessDocument.cs](#).

#### 5.7.4.4 ProcessedElements

```
List<Tuple<Node, int> > OSXJV.Classes.ProcessDocument.ProcessedElements = new List<Tuple<Node,  
int>>() [private]
```

Used with threading to keep list of processed Nodes.

Definition at line 30 of file [ProcessDocument.cs](#).

#### 5.7.4.5 th

```
Thread OSXJV.Classes.ProcessDocument.th [private]
```

Definition at line 41 of file [ProcessDocument.cs](#).

#### 5.7.4.6 ThreadList

```
List<Thread> OSXJV.Classes.ProcessDocument.ThreadList = new List<Thread>() [private]
```

Used with threading to keep list of running threads.

Definition at line 35 of file [ProcessDocument.cs](#).

#### 5.7.4.7 type

```
string OSXJV.Classes.ProcessDocument.type [private]
```

Document Type.

Definition at line 40 of file [ProcessDocument.cs](#).

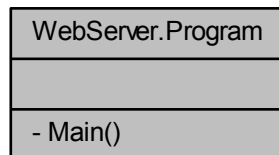
The documentation for this class was generated from the following file:

- [WebServiceCSharp/OSXJVCclasses/ProcessDocument.cs](#)

## 5.8 WebServer.Program Class Reference

The Initialiser

Collaboration diagram for WebServer.Program:



### Static Private Member Functions

- static void [Main](#) (string[] args)  
*The Main function that starts the HttpServer*

#### 5.8.1 Detailed Description

The Initialiser

Definition at line 12 of file [Program.cs](#).

#### 5.8.2 Member Function Documentation

##### 5.8.2.1 Main()

```
static void WebServer.Program.Main (  
    string [] args ) [static], [private]
```

The Main function that starts the HttpServer

Parameters

<i>args</i>	Pass Cache Folder and Logger (Optional)
-------------	---

Definition at line 18 of file [Program.cs](#).

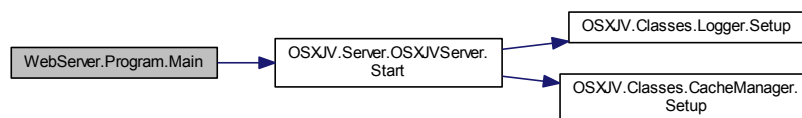
References [OSXJV.Server.OSXJVServer.Start\(\)](#).

```

00019     {
00020
00021         if (args.Length == 0)
00022         {
00023             Console.WriteLine("Using Default Cache Directory Path and Logger Directory Path");
00024             string dir = Directory.GetCurrentDirectory();
00025             Array.Resize(ref args, 2);
00026             args[0] = dir + "/Cache/";
00027             args[1] = dir + "/Logger/";
00028             if (!Directory.Exists(args[0]))
00029                 Directory.CreateDirectory(args[0]);
00030             if (!Directory.Exists(args[1]))
00031                 Directory.CreateDirectory(args[1]);
00032         }
00033
00034         if (args[0] == args[1])
00035         {
00036             Console.WriteLine("Cache location and Log location is the same. Please enter two different
locations");
00037         }
00038         else
00039         {
00040             try
00041             {
00042                 OSXJVServer s = new OSXJVServer();
00043                 s.Start(args[0], args[1]);
00044             }
00045             catch (Exception e)
00046             {
00047                 Console.WriteLine(e.Message);
00048                 Console.WriteLine("Press any key to exit");
00049                 Console.Read();
00050             }
00051         }
00052     }
00053 }

```

Here is the call graph for this function:



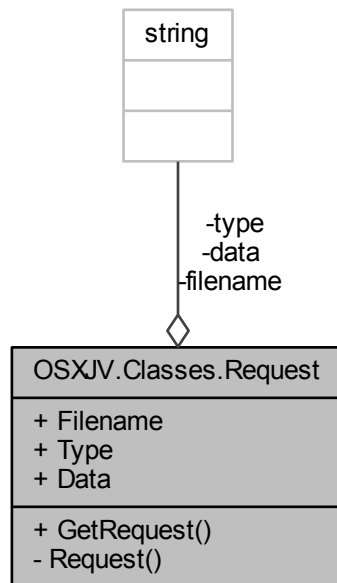
The documentation for this class was generated from the following file:

- [WebServiceCSharp/Program.cs](#)

## 5.9 OSXJV.Classes.Request Class Reference

A object containing the document to process, filename and type.

Collaboration diagram for OSXJV.Classes.Request:



### Static Public Member Functions

- static [Request GetRequest](#) (string [filename](#), string [type](#), string [data](#))  
Creates an instance of [Request](#).

### Properties

- string [Filename](#) [get, set]  
To retrieve the filename of the document
- string [Type](#) [get, set]  
To retrieve type of document
- string [Data](#) [get, set]  
To retrieve the document data

### Private Member Functions

- [Request](#) (string [filename](#), string [type](#), string [data](#))  
Initialises the [Request](#) object, can only be called from [GetRequest\(...\)](#).

### Private Attributes

- string [filename](#)  
Document Filename.
- string [type](#)  
Type of document.
- string [data](#)  
Contents of documents.

### 5.9.1 Detailed Description

A object containing the document to process, filename and type.

Definition at line 8 of file [Request.cs](#).

### 5.9.2 Constructor & Destructor Documentation

#### 5.9.2.1 Request()

```
OSXJV.Classes.Request.Request (
    string filename,
    string type,
    string data ) [private]
```

Initialises the [Request](#) object, can only be called from GetRequest(...).

##### Parameters

<i>filename</i>	The document filename e.g. Test
<i>type</i>	The document file type e.g. text/xml
<i>data</i>	The document data e.g. {"name":"bob","address":"123 Somewhere"}

Definition at line 31 of file [Request.cs](#).

```
00032     {
00033         this.filename = filename;
00034         this.type = type;
00035         this.data = data;
00036     }
```

### 5.9.3 Member Function Documentation

#### 5.9.3.1 GetRequest()

```
static Request OSXJV.Classes.Request.GetRequest (
    string filename,
    string type,
    string data ) [static]
```

Creates an instance of [Request](#).

##### Parameters

<i>filename</i>	The document filename e.g. Test
<i>type</i>	The document file type e.g. text/xml
<i>data</i>	The document data e.g. {"name":"bob","address":"123 Somewhere"}



## Returns

Object of [Request](#)

Definition at line 45 of file [Request.cs](#).

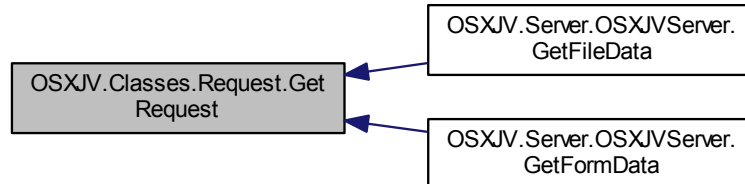
Referenced by [OSXJV.Server.OSXJVServer.GetFileData\(\)](#), and [OSXJV.Server.OSXJVServer.GetFormData\(\)](#).

```

00046     {
00047         string Type = "";
00048         if (string.IsNullOrEmpty(filename) || string.IsNullOrEmpty(
00049             type) || string.IsNullOrEmpty(data))
00050             throw new ArgumentException();
00051         if (type.Equals("text/xml") || type.Equals("application/xml"))
00052         {
00053             Type = "XML";
00054         }
00055         else if (type.Equals("text/html"))
00056         {
00057             Type = "HTML";
00058         }
00059         else if (type.Equals("application/json") || type.Equals("application/octet-stream"))
00060         {
00061             Type = "JSON";
00062         }
00063         return new Request(filename, Type, data);

```

Here is the caller graph for this function:



## 5.9.4 Member Data Documentation

### 5.9.4.1 data

```
string OSXJV.Classes.Request.data [private]
```

Contents of documents.

Definition at line 23 of file [Request.cs](#).

#### 5.9.4.2 filename

```
string OSXJV.Classes.Request.filename [private]
```

Document Filename.

Definition at line 13 of file [Request.cs](#).

#### 5.9.4.3 type

```
string OSXJV.Classes.Request.type [private]
```

Type of document.

Definition at line 18 of file [Request.cs](#).

### 5.9.5 Property Documentation

#### 5.9.5.1 Data

```
string OSXJV.Classes.Request.Data [get], [set]
```

To retrieve the document data

Definition at line 101 of file [Request.cs](#).

Referenced by [OSXJV.Server.OSXJVServer.HandlePost\(\)](#).

#### 5.9.5.2 Filename

```
string OSXJV.Classes.Request.Filename [get], [set]
```

To retrieve the filename of the document

Definition at line 69 of file [Request.cs](#).

## 5.9.5.3 Type

```
string OSXJV.Classes.Request.Type [get], [set]
```

To retrieve type of document

Definition at line 85 of file [Request.cs](#).

Referenced by [OSXJV.Server.OSXJVServer.HandlePost\(\)](#).

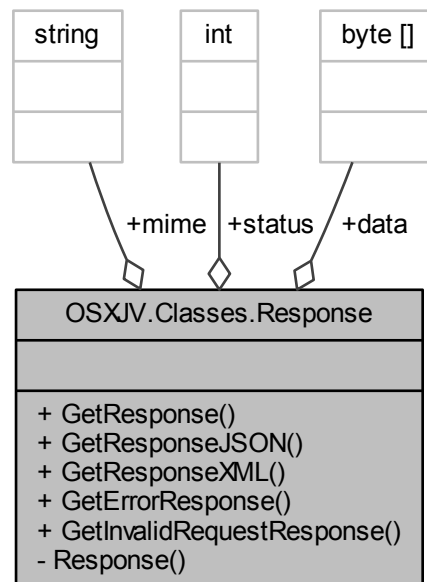
The documentation for this class was generated from the following file:

- [WebServiceCSharp/OSXJVClasses/Request.cs](#)

## 5.10 OSXJV.Classes.Response Class Reference

The Object containing data to send to the client

Collaboration diagram for OSXJV.Classes.Response:



## Static Public Member Functions

- static [Response GetResponse](#) (int [status](#), string [type](#), byte[] [data](#))  
*A custom response object*
- static [Response GetResponseJSON](#) (int [status](#), byte[] [data](#))  
*Return an application/json response*
- static [Response GetResponseXML](#) (int [status](#), byte[] [data](#))  
*Return an text/xml response*
- static [Response GetErrorResponse](#) (string [message](#))  
*Return a error response object*
- static [Response GetInvalidRequestResponse](#) ()  
*Returns an invalid response object*

## Public Attributes

- byte [] [data](#) = null  
*Data*
- int [status](#)  
*Status Code*
- string [mime](#)  
*Data type e.g. "application/json"*

## Private Member Functions

- [Response](#) (int [status](#), string [mime](#), byte[] [buffer](#))  
*Constructor*

### 5.10.1 Detailed Description

The Object containing data to send to the client

Definition at line 9 of file [Response.cs](#).

### 5.10.2 Constructor & Destructor Documentation

#### 5.10.2.1 Response()

```
OSXJV.Classes.Response.Response (
    int status,
    string mime,
    byte [] buffer ) [private]
```

#### Constructor

##### Parameters

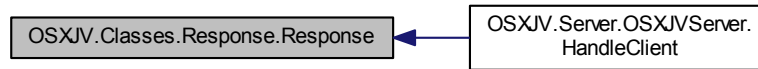
<i>status</i>	Status Code
<i>mime</i>	MIME type
<i>buffer</i>	Data

Definition at line 34 of file [Response.cs](#).

Referenced by [OSXJV.Server.OSXJVServer.HandleClient\(\)](#).

```
00035         {
00036             this.status = status;
00037             this.data = buffer;
00038             this.mime = mime;
00039         }
```

Here is the caller graph for this function:



### 5.10.3 Member Function Documentation

#### 5.10.3.1 GetErrorResponse()

```
static Response OSXJV.Classes.Response.GetErrorResponse (  
    string message ) [static]
```

Return a error response object

##### Returns

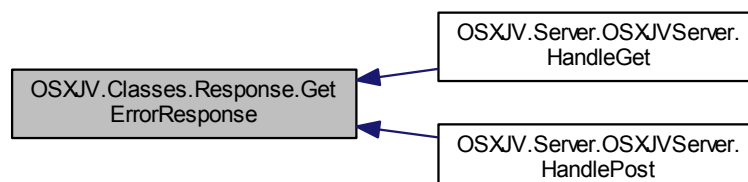
New response object

Definition at line 118 of file [Response.cs](#).

Referenced by [OSXJV.Server.OSXJVServer.HandleGet\(\)](#), and [OSXJV.Server.OSXJVServer.HandlePost\(\)](#).

```
00119     {  
00120         byte[] res = Encoding.UTF8.GetBytes(message);  
00121         return new Response(400, "text/html", res);  
00122     }
```

Here is the caller graph for this function:



### 5.10.3.2 GetInvalidRequestResponse()

```
static Response OSXJV.Classes.Response.GetInvalidRequestResponse ( ) [static]
```

Returns an invalid response object

#### Returns

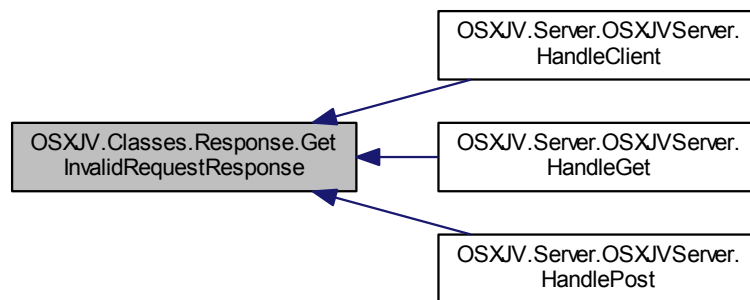
New response object

Definition at line 128 of file [Response.cs](#).

Referenced by [OSXJV.Server.OSXJVServer.HandleClient\(\)](#), [OSXJV.Server.OSXJVServer.HandleGet\(\)](#), and [OSXJV.Server.OSXJVServer.HandlePost\(\)](#).

```
00129     {
00130         return new Response(405, "text/html", new byte[0]);
00131     }
```

Here is the caller graph for this function:



### 5.10.3.3 GetResponse()

```
static Response OSXJV.Classes.Response.GetResponse (
    int status,
    string type,
    byte [] data ) [static]
```

A custom response object

#### Parameters

<i>status</i>	The HTTP Code to send back e.g. 200 for success
<i>type</i>	Data type to send back e.g. application/json
<i>data</i>	The data to send

## Returns

Definition at line 48 of file [Response.cs](#).

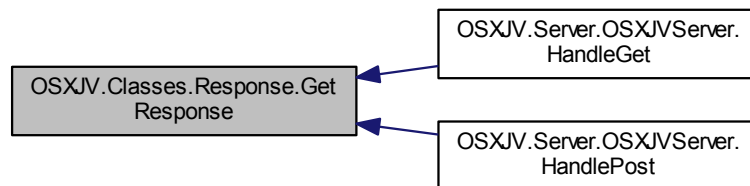
Referenced by [OSXJV.Server.OSXJVServer.HandleGet\(\)](#), and [OSXJV.Server.OSXJVServer.HandlePost\(\)](#).

```

00049     {
00050         if(string.IsNullOrEmpty(type))
00051             throw new ArgumentException("Type cannot be Null or empty");
00052
00053         if (status.Equals(null))
00054             throw new ArgumentException("Status cannot be Null");
00055         else
00056             if (status == 0)
00057                 throw new ArgumentException("Status cannot be 0");
00058
00059         if (data == null)
00060             throw new ArgumentException("Data cannot be null");
00061         else
00062             if (data.Length == 0)
00063                 throw new ArgumentException("No data, use invalid or error response");
00064
00065         return new Response(status, type, data);
00066     }

```

Here is the caller graph for this function:



#### 5.10.3.4 GetResponseJSON()

```

static Response OSXJV.Classes.Response.GetResponseJSON (
    int status,
    byte [] data ) [static]

```

Return an application/json response

## Parameters

<i>status</i>	The HTTP Code to send back e.g. 200 for success
<i>data</i>	The data to send

**Returns**

New response object

Definition at line 74 of file [Response.cs](#).

```

00075     {
00076         if (status.Equals(null))
00077             throw new ArgumentException("Status cannot be Null");
00078         else
00079             if (status == 0)
00080                 throw new ArgumentException("Status cannot be 0");
00081
00082         if (data == null)
00083             throw new ArgumentException("Data cannot be null");
00084         else
00085             if (data.Length == 0)
00086                 throw new ArgumentException("No data, use invalid or error response");
00087
00088         return new Response(status, "application/json", data);
00089     }

```

**5.10.3.5 GetResponseXML()**

```

static Response OSXJV.Classes.Response.GetResponseXML (
    int status,
    byte [] data ) [static]

```

Return an text/xml response

**Parameters**

<i>status</i>	The HTTP Code to send back e.g. 200 for success
<i>data</i>	The data to send

**Returns**

New response object

Definition at line 97 of file [Response.cs](#).

```

00098     {
00099         if (status.Equals(null))
00100             throw new ArgumentException("Status cannot be Null");
00101         else
00102             if (status == 0)
00103                 throw new ArgumentException("Status cannot be 0");
00104
00105         if (data == null)
00106             throw new ArgumentException("Data cannot be null");
00107         else
00108             if (data.Length == 0)
00109                 throw new ArgumentException("No data, use invalid or error response");
00110
00111         return new Response(status, "text/xml", data);
00112     }

```

**5.10.4 Member Data Documentation**



#### 5.10.4.1 data

```
byte [] OSXJV.Classes.Response.data = null
```

Data

Definition at line 14 of file [Response.cs](#).

Referenced by [OSXJV.Server.OSXJVServer.Post\(\)](#).

#### 5.10.4.2 mime

```
string OSXJV.Classes.Response.mime
```

Data type e.g. "application/json"

Definition at line 24 of file [Response.cs](#).

Referenced by [OSXJV.Server.OSXJVServer.Post\(\)](#).

#### 5.10.4.3 status

```
int OSXJV.Classes.Response.status
```

Status Code

Definition at line 19 of file [Response.cs](#).

Referenced by [OSXJV.Server.OSXJVServer.Post\(\)](#).

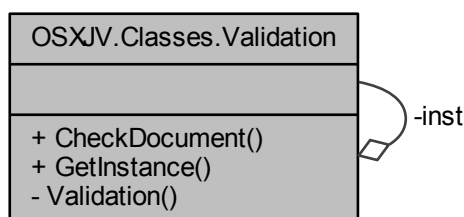
The documentation for this class was generated from the following file:

- [WebServiceCSharp/OSXJVClasses/Response.cs](#)

## 5.11 OSXJV.Classes.Validation Class Reference

Perform validation on document

Collaboration diagram for OSXJV.Classes.Validation:



## Public Member Functions

- bool [CheckDocument](#) (string data, string type)  
*Checks the document and if it is valid*

## Static Public Member Functions

- static [Validation GetInstance](#) ()  
*Gets the instance.*

## Private Member Functions

- [Validation](#) ()  
*Constructor*

## Static Private Attributes

- static [Validation inst](#)  
*The inst.*

### 5.11.1 Detailed Description

Perform validation on document

Definition at line 12 of file [Validation.cs](#).

### 5.11.2 Constructor & Destructor Documentation

#### 5.11.2.1 Validation()

```
OSXJV.Classes.Validation.Validation ( ) [private]
```

Constructor

Definition at line 22 of file [Validation.cs](#).

```
00022 {}
```

### 5.11.3 Member Function Documentation

#### 5.11.3.1 CheckDocument()

```
bool OSXJV.Classes.Validation.CheckDocument (
    string data,
    string type )
```

Checks the document and if it is valid

## Parameters

<i>data</i>	Document contents
<i>type</i>	Type of document

## Returns

True if valid, else false

## Exceptions

<i>ArgumentException</i>	Invalid data type or data and type cannot be null
<i>XmlException</i>	Invalid XML or HTML
<i>JsonReaderException</i>	Invalid JSON

Definition at line 44 of file [Validation.cs](#).

Referenced by [OSXJV.Server.OSXJVServer.HandlePost\(\)](#).

```

00045     {
00046         if(string.IsNullOrEmpty(data) || string.IsNullOrEmpty(type))
00047         {
00048             throw new ArgumentException("Data or Type cannot be Null");
00049         }
00050
00051         if (type.Equals("XML") || type.Equals("HTML"))
00052         {
00053             XmlReaderSettings settings = new XmlReaderSettings();
00054             settings.DtdProcessing = DtdProcessing.Parse;
00055             settings.MaxCharactersFromEntities = 2048;
00056             using (XmlReader xr = XmlReader.Create(new StringReader(data), settings))
00057             {
00058                 try
00059                 {
00060                     while (xr.Read()) { }
00061                     return true;
00062                 }
00063                 catch (XmlException ex)
00064                 {
00065                     throw ex;
00066                 }
00067             }
00068         }
00069         else if(type.Equals("JSON"))
00070         {
00071             try
00072             {
00073                 JToken.Parse(data);
00074                 return true;
00075             }
00076             catch (JsonReaderException ex)
00077             {
00078                 throw new JsonReaderException(ex.Message);
00079             }
00080         }
00081         throw new ArgumentException("Invalid data or type");
00082     }
00083 
```

Here is the caller graph for this function:



#### 5.11.3.2 GetInstance()

```
static Validation OSXJV.Classes.Validation.GetInstance ( ) [static]
```

Gets the instance.

##### Returns

The instance.

Definition at line 28 of file [Validation.cs](#).

Referenced by [OSXJV.Server.OSXJVServer.HandlePost\(\)](#).

```

00029      {
00030          if (inst != null)
00031              return inst;
00032          else
00033              return (inst = new Validation ());
00034      }
  
```

Here is the caller graph for this function:



### 5.11.4 Member Data Documentation

#### 5.11.4.1 inst

```
Validation OSXJV.Classes.Validation.inst [static], [private]
```

The inst.

Definition at line 17 of file [Validation.cs](#).

The documentation for this class was generated from the following file:

- [WebServiceCSharp/OSXJVClasses/Validation.cs](#)

## Chapter 6

# File Documentation

### 6.1 WebServiceCSharp/OSXJVClasses/Attribute.cs File Reference

#### Classes

- class [OSXJV.Classes.Attribute](#)

#### Namespaces

- namespace [OSXJV.Classes](#)

### 6.2 Attribute.cs

```
00001 namespace OSXJV.Classes
00002 {
00006     public class Attribute
00007     {
00008         private string name;
00009         private string value;
00010
00014         public string Name
00015         {
00016             get
00017             {
00018                 return name;
00019             }
00020
00021             set
00022             {
00023                 name = value;
00024             }
00025         }
00029         public string Value
00030         {
00031             get
00032             {
00033                 return value;
00034             }
00035
00036             set
00037             {
00038                 this.value = value;
00039             }
00040         }
00041     }
00042 }
```

## 6.3 WebServiceCSharp/OSXJVClasses/CacheManager.cs File Reference

### Classes

- class [OSXJV.Classes.CacheManager](#)  
*Manages Saving an Retrieving Processed Documents*

### Namespaces

- namespace [OSXJV.Classes](#)

## 6.4 CacheManager.cs

```

00001 using System;
00002 using System.IO;
00003
00004 namespace OSXJV.Classes
00005 {
00009     public class CacheManager
00010     {
00014         private static CacheManager Inst;
00015
00019         private static string path = null;
00020
00025         private CacheManager(string cachePath)
00026         {
00027             path = cachePath;
00028         }
00029
00034         public static bool Setup(string path)
00035         {
00036             if (string.IsNullOrEmpty(path))
00037                 throw new ArgumentException("Path cannot be empty");
00038
00039             if (!Directory.Exists(string.Format(@"{0}", path)))
00040                 throw new Exception("Path is not a valid cache directory");
00041
00042             return (Inst = new CacheManager(path)) != null ? true : false;
00043         }
00044
00050         public static CacheManager GetInstance()
00051         {
00052             if (Inst != null)
00053                 return Inst;
00054             else
00055                 throw new Exception("CacheManger has not been setup");
00056         }
00057
00063         public string getFile(string ID)
00064         {
00065             if (string.IsNullOrEmpty(ID))
00066                 throw new ArgumentException("ID cannot be null or empty");
00067
00068             string filePath = path + "/" + ID.Replace("/", "") + ".json";
00069             string output = "";
00070
00071             using (StreamReader sr = new StreamReader(filePath))
00072             {
00073                 output = sr.ReadToEnd();
00074             }
00075
00076             if (!string.IsNullOrEmpty(output))
00077                 return output;
00078             else
00079                 throw new Exception("Error Reading From File");
00080         }
00081
00087         public bool saveFile(string ID, string nodes)
00088         {
00089             if (string.IsNullOrEmpty(ID))
00090                 throw new ArgumentException("ID cannot be null or empty");
00091
00092             if (string.IsNullOrEmpty(nodes))

```

```

00093         throw new ArgumentException("Document cannot be null or empty");
00094
00095         string filePath = path + "/" + ID + ".json";
00096         try
00097         {
00098             using (StreamWriter sw = new StreamWriter(filePath))
00099             {
00100                 sw.WriteLine(nodes);
00101             }
00102         }
00103         catch
00104         {
00105             throw new Exception("Failed to save file");
00106         }
00107
00108         return true;
00109     }
00114     public static void Close()
00115     {
00116         if (Inst == null)
00117             throw new Exception("CacheManager Already Closed");
00118         else
00119         {
00120             path = null; //Clear static path
00121             Inst = null; //clear static instance
00122         }
00123     }
00124
00128     public static void ManageCache()
00129     {
00130         if (path != null)
00131         {
00132             string[] files = Directory.GetFiles(path);
00133             foreach (string file in files)
00134             {
00135                 if (File.GetLastAccessTime(file) < DateTime.Now.AddHours(-6.0))
00136                     File.Delete(file);
00137             }
00138         }
00139         else
00140             throw new Exception("CacheManger not setup");
00141     }
00142 }
00143 }

```

## 6.5 WebServiceCSharp/OSXJVClasses/Logger.cs File Reference

### Classes

- class [OSXJV.Classes.Logger](#)  
A simple class that writes errors to a single file.

### Namespaces

- namespace [OSXJV.Classes](#)

## 6.6 Logger.cs

```

00001 using System;
00002 using System.IO;
00003
00004 namespace OSXJV.Classes
00005 {
00009     public class Logger
00010     {
00014         private static Logger inst;
00015         private string location;
00016
00017         private Logger(string location)
00018         {

```

```

00019         this.location = location;
00020     }
00021
00026     public static bool Setup(string location)
00027     {
00028         if (string.IsNullOrEmpty(location))
00029             throw new ArgumentException("Location cannot be empty");
00030
00031         if (!Directory.Exists(string.Format(@"{0}", location)))
00032             throw new Exception("Location is not a valid logger directory");
00033
00034         return (inst = new Logger(location)) != null ? true:false;
00035     }
00036
00041     public static Logger GetInstance()
00042     {
00043         if (inst != null)
00044             return inst;
00045         else
00046             throw new Exception("Logger has not been setup");
00047     }
00048
00053     public void WriteError(string error)
00054     {
00055         try
00056         {
00057             if (!string.IsNullOrEmpty(error))
00058             {
00059                 string file = string.Format(@"{0}/Error-{1}.txt", location, DateTime.Now.ToString("
dd-MM-yy hh-MM-ss"));
00060                 StreamWriter sw = new StreamWriter(file);
00061                 sw.WriteLine(error);
00062                 sw.WriteLine();
00063                 sw.Close();
00064             }
00065         }
00066         catch (IOException e)
00067         {
00068             throw e;
00069         }
00070     }
00071
00072     public static void Close()
00073     {
00074         if (inst == null)
00075             throw new Exception("Logger Already Closed");
00076         else
00077             inst = null;
00078     }
00079 }
00080 }

```

## 6.7 WebServiceCSharp/OSXJVClasses/Node.cs File Reference

### Classes

- class [OSXJV.Classes.Node](#)  
*Contain Processed Document Information*

### Namespaces

- namespace [OSXJV.Classes](#)

## 6.8 Node.cs

```

00001 using System.Collections.Generic;
00002 using Newtonsoft.Json.Serialization;
00003 using Newtonsoft.Json;
00004
00005 namespace OSXJV.Classes

```



```

00006 {
00010     public class Node
00011     {
00012         private string name;
00013         private List<Attribute> attributes;
00014         private string value;
00015         private List<Node> children;
00016         private int number;
00017         private bool visited;
00018         private List<string> comments;
00019
00023         public Node()
00024         {
00025             Attributes = new List<Attribute>();
00026             Children = new List<Node>();
00027             Comments = new List<string>();
00028             number = 0;
00029             visited = false;
00030         }
00031
00035         public int Number
00036         {
00037             get
00038             {
00039                 return number;
00040             }
00041
00042             set
00043             {
00044                 number = value;
00045             }
00046         }
00047
00051         public string Name
00052         {
00053             get
00054             {
00055                 return name;
00056             }
00057
00058             set
00059             {
00060                 name = value;
00061             }
00062         }
00063
00067         [JsonProperty(NullValueHandling = NullValueHandling.Ignore)]
00068         public string Value
00069         {
00070             get
00071             {
00072                 return value;
00073             }
00074
00075             set
00076             {
00077                 this.value = value;
00078             }
00079         }
00080
00084         public List<string> Comments
00085         {
00086             get
00087             {
00088                 return comments;
00089             }
00090
00091             set
00092             {
00093                 comments = value;
00094             }
00095         }
00096
00100         [JsonProperty()]
00101         public List<Attribute> Attributes
00102         {
00103             get
00104             {
00105                 return attributes;
00106             }
00107
00108             set
00109             {
00110                 attributes = value;
00111             }
00112         }
00113

```

```

00117         [JsonProperty()]
00118         public List<Node> Children
00119         {
00120             get
00121             {
00122                 return children;
00123             }
00124             set
00125             {
00126                 children = value;
00127             }
00128         }
00129     }
00130
00131     [Newtonsoft.Json.JsonIgnore]
00132     public bool Visited
00133     {
00134         get
00135         {
00136             return visited;
00137         }
00138         set
00139         {
00140             visited = value;
00141         }
00142     }
00143 }
00144
00145 }
00146
00147 }
00148 }

```

## 6.9 WebServiceCSharp/OSXJVClasses/Output.cs File Reference

### Classes

- class [OSXJV.Classes.Output](#)  
*Creates the [Output](#) for the web page to display.*

### Namespaces

- namespace [OSXJV.Classes](#)

## 6.10 Output.cs

```

00001 using Newtonsoft.Json.Linq;
00002 using System;
00003 using System.Collections.Generic;
00004 using System.Threading;
00005
00006 namespace OSXJV.Classes
00007 {
00011     public class Output
00012     {
00016         private int left = 100, top = 130;
00017
00021         private Node nodes;
00022
00026         private bool GotParent = false;
00027
00031         private int Parent = 0;
00032
00036         private List<Tuple<int, string>> cNodes = new List<Tuple<int, string>>();
00037
00042         public Output(Node nodes)
00043         {
00044             if (nodes == null)
00045                 throw new ArgumentException();
00046             this.nodes = nodes;
00047         }
00048
00053         public JObject CreateGrid()

```

```

00054     {
00055         JObject obj = new JObject();
00056         obj.Add("text", nodes.Name);
00057         obj.Add("id", nodes.Number);
00058         obj.Add("state", new JObject(new JProperty("selected", true)));
00059
00060         if(nodes.Children.Count > 0)
00061         {
00062             JArray array = new JArray();
00063             foreach (Node n2 in nodes.Children)
00064             {
00065                 array.Add(GridGetChidren(n2));
00066             }
00067             obj.Add("children", array);
00068         }
00069         return obj;
00070     }
00071
00072 private JObject GridGetChidren(Node n)
00073 {
00074     JObject child = new JObject();
00075     child.Add("id", n.Number);
00076     child.Add("text", n.Name);
00077
00078     if (n.Children.Count > 0)
00079     {
00080         JArray array = new JArray();
00081         foreach (Node n2 in n.Children)
00082         {
00083             array.Add(GridGetChidren(n2));
00084         }
00085         child.Add("children", array);
00086     }
00087     return child;
00088 }
00089
00090 public string CreateViewSingle(int node, int nodeStart = 0)
00091 {
00092     string output = "<div class='text-center ui-layout-center ui-layout-pane
00093 ui-layout-pane-center'><div style ='display:inline-block' class='ui-selectable ui-droppable'>";
00094
00095     if (nodes.Number.Equals(node))
00096     {
00097         int count = 0;
00098         output += CreateNodeView(nodes, "node");
00099
00100         foreach (Node n in nodes.Children)
00101         {
00102             if (nodeStart > 0)
00103             {
00104                 if (count != nodeStart)
00105                     continue;
00106             }
00107             count++;
00108             output += CreateNodeView(n, "node-child"); //Child(Nodes) Thread
00109
00110             if ((count-nodeStart) == 200)
00111             {
00112                 output += CreateExtraNode("node-child", count);
00113                 break;
00114             }
00115         }
00116     }
00117     else
00118     {
00119         GetParent(nodes, node);
00120         string temp = "";
00121         if (GotParent)
00122         {
00123             if (nodes.Number == Parent)
00124             {
00125                 output += CreateNodeView(nodes, "node-parent");
00126             }
00127             foreach (Node n2 in nodes.Children)
00128             {
00129                 if (GotParent)
00130                 {
00131                     if (n2.Number == Parent)
00132                     {
00133                         output += CreateNodeView(n2, "node-parent");
00134                     }
00135                 }
00136                 temp += CheckChildren(n2, node);
00137             }
00138             if (!string.IsNullOrEmpty(temp))
00139         }
00140     }
00141 }

```

```

00152         output += temp;
00153     }
00154     output += "</div></div>"; //Close out divs
00155     return output;
00156 }
00157
00164 private string CreateExtraNode(string type,int id)
00165 {
00166     string node = "";
00167
00168     if (type == "node")
00169     {
00170         if (GotParent)
00171         {
00172             left = left + 400;
00173         }
00174     }
00175     if (type == "node-child")
00176     {
00177         left = left + 400;
00178     }
00179
00180     node += "<div class='node-child type ui-draggable ui-selectee' style='left:" + left + "px; top:"
00181 " + top + "px;margin-bottom:50px;'>";
00182     node += "<div class='head'><span><button class='nameBtn' onclick='GetMoreNodes(" + id + "
00183 )'>Show Lower</button></span></div>";
00184     node += "</div></div>";
00185     return node;
00186 }
00187
00194 private string CreatePreviousNode(string type, int leftVal, int topVal, int id)
00195 {
00196     string node = "";
00197
00198     if (type == "node")
00199     {
00200         if (GotParent)
00201         {
00202             leftVal = leftVal + 400;
00203         }
00204     }
00205     if (type == "node-child")
00206     {
00207         leftVal = leftVal + 400;
00208     }
00209     node += "<div class='node-child type ui-draggable ui-selectee' style='left:" + leftVal + "px;
00210 top:" + topVal + "px;'>";
00211     node += "<div class='head'><span><button class='nameBtn' onclick='GetMoreNodes(" + id + "
00212 )'>Show Higher</button></span></div>";
00213     node += "</div></div>";
00214     return node;
00215 }
00216
00223 private string CreateExtraNode(string type, int leftVal, int topVal,int id)
00224 {
00225     string node = "";
00226
00227     if (type == "node")
00228     {
00229         if (GotParent)
00230         {
00231             leftVal = leftVal + 400;
00232         }
00233     }
00234     if (type == "node-child")
00235     {
00236         leftVal = leftVal + 400;
00237     }
00238     node += "<div class='node-child type ui-draggable ui-selectee' style='left:" + leftVal + "px;
00239 top:" + topVal + "px;margin-bottom:50px;'>";
00240     node += "<div class='head'><span><button class='nameBtn' onclick='GetMoreNodes(" + id + "
00241 )'>Show Lower</button></span></div>";
00242     node += "</div></div>";
00243     return node;
00244 }
00245
00252 private void CreateNodeChildViewsParallel(List<Node> job,int start,
00253 bool showHigher,int next,int previous)
00254 {
00255     int threadID = int.Parse(Thread.CurrentThread.Name);
00256     string type = "node-child";
00257     string output = "";
00258
00259     if(start == 0 && showHigher)
00260     {
00261         output += CreatePreviousNode(type, left, top, previous);
00262     }

```

```

00262         bool hadCommentsPrev = false;
00263         int numCommentsPrevious = 0;
00264
00265         foreach(Node n in job)
00266         {
00267             int extra = showHigher ? 130 * (start + 1) : 130 * start;
00268
00269             if (hadCommentsPrev)
00270                 extra += (numCommentsPrevious * 25);
00271
00272             if (n.Comments.Count > 0)
00273             {
00274                 hadCommentsPrev = true;
00275                 numCommentsPrevious = n.Comments.Count;
00276             }
00277             else
00278                 hadCommentsPrev = false;
00279
00280             output += CreateNodeView(n, type, left, top + extra);
00281             start++;
00282             if (start == 200)
00283             {
00284                 output += CreateExtraNode(type, left, top + extra + 130, next);
00285                 break;
00286             }
00287         }
00288
00289         cNodes.Add(new Tuple<int, string>(threadID, output));
00290     }
00291 }
00292
00300 public string CreateView(int node = 1, int pCount = 4, int nodeStart = 0) //Setting
Defaults
00301 {
00302
00303     List<Thread> threadList = new List<Thread>();
00304
00305     string output = "<div class='text-center ui-layout-center ui-layout-pane
ui-layout-pane-center'><div style ='display:inline-block' class='ui-selectable ui-droppable'>";
00306     if (nodes.Number.Equals(node))
00307     {
00308         int childCount = 0;
00309
00310         if (nodes.Children.Count < 200)
00311             childCount = nodes.Children.Count;
00312         else
00313         {
00314             childCount = 200;
00315         }
00316
00317         if (childCount < pCount * 2)
00318         {
00319             output += CreateNodeView(nodes, "node", left, top);
00320             foreach(Node n2 in nodes.Children)
00321             {
00322                 output += CreateNodeView(n2, "node-child");
00323             }
00324         }
00325         else
00326         {
00327             int spread = (int)Math.Ceiling((double)childCount / (double)pCount);
00328
00329             output += CreateNodeView(nodes, "node", left, top); //Parent (Node) Thread
00330
00331             for (int i = 0; i < pCount; i++)
00332             {
00333                 int neg = 0;
00334                 if ((spread * (i + 1)) > childCount)
00335                 {
00336                     neg = childCount - (spread * (i + 1));
00337                 }
00338                 int start = (spread * i);
00339                 int rangeStart = (spread * i) + nodeStart;
00340                 bool showHigher = nodeStart != 0 ? true : false;
00341
00342                 List<Node> NodesToProcess = nodes.Children.GetRange(rangeStart, spread +
neg);
00343                 Thread threadJob = new Thread(() => CreateNodeChildViewsParallel(NodesToProcess,
start, showHigher, childCount + nodeStart, nodeStart - childCount));
00344                 threadJob.Name = i.ToString();
00345                 threadJob.Start();
00346                 threadList.Add(threadJob);
00347             }
00348             foreach(Thread t in threadList)
00349             {
00350                 t.Join();
00351             }

```

```

00352         cNodes.Sort((x, y) => x.Item1.CompareTo(y.Item1));
00353
00354         foreach(Tuple<int,string> tup in cNodes)
00355         {
00356             output += tup.Item2;
00357         }
00358     }
00359 }
00360
00361 else
00362 {
00363     GetParent(nodes, node);
00364     string temp = "";
00365     if (GotParent)
00366     {
00367         if (nodes.Number == Parent)
00368         {
00369             output += CreateNodeView(nodes, "node-parent");
00370         }
00371     }
00372     bool found = false;
00373     foreach (Node n2 in nodes.Children)
00374     {
00375         if (GotParent)
00376         {
00377             if (n2.Number == Parent)
00378             {
00379                 output += CreateNodeView(n2, "node-parent");
00380             }
00381             temp += CheckChildren(n2, node, pCount, nodeStart, ref found);
00382             if (found)
00383                 break;
00384         }
00385     }
00386     if (!string.IsNullOrEmpty(temp))
00387         output += temp;
00388 }
00389 output += "</div></div>";
00390 return output;
00391 }
00392
00399 private string CheckChildren(Node n, int number)
00400 {
00401     string output = "";
00402     if (CheckNodeNumber(n, number))
00403     {
00404         int count = 0;
00405         output += CreateNodeView(n, "node");
00406         foreach (Node n2 in n.Children)
00407         {
00408             count++;
00409             output += CreateNodeView(n2, "node-child");
00410         }
00411     }
00412 }
00413 else if (n.Children.Count > 0)
00414 {
00415     foreach (Node n2 in n.Children)
00416     {
00417         if (GotParent)
00418         {
00419             if (n2.Number == Parent)
00420             {
00421                 output += CreateNodeView(n2, "node-parent");
00422             }
00423             output += CheckChildren(n2, number);
00424         }
00425     }
00426 }
00427 return output;
00428 }
00429 }
00430
00440 private string CheckChildren(Node n, int number, int pCount, int nodeStart, ref
bool found)
00441 {
00442     string output = "";
00443     if (CheckNodeNumber(n, number))
00444     {
00445         found = true;
00446         List<Thread> threadList = new List<Thread>();
00447
00448         int count = 0;
00449         output += CreateNodeView(n, "node");
00450         count++;
00451         //output += CreateNodeView(n2, "node-child");
00452         int childCount = 0;

```

```

00453
00454         if (n.Children.Count < 200)
00455             childCount = n.Children.Count;
00456         else
00457         {
00458             childCount = 200;
00459         }
00460         if (childCount < pCount * 2)
00461         {
00462             foreach (Node n2 in n.Children)
00463             {
00464                 output += CreateNodeView(n2, "node-child");
00465             }
00466         }
00467         else
00468         {
00469             int spread = (int)Math.Ceiling((double)childCount / (double)pCount);
00470
00471             if (childCount > 0)
00472             {
00473                 for (int i = 0; i < pCount; i++)
00474                 {
00475                     int neg = 0;
00476                     if ((spread * (i + 1)) > childCount)
00477                     {
00478                         neg = childCount - (spread * (i + 1));
00479                     }
00480                     int start = (spread * i);
00481                     int rangeStart = (spread * i) + nodeStart;
00482                     bool showHigher = nodeStart != 0 ? true : false;
00483
00484                     List<Node> NodesToProcess = n.Children.GetRange(rangeStart, spread +
neg);
00485
00486                     if (NodesToProcess.Count > 0)
00487                     {
00488                         Thread threadJob = new Thread(() => CreateNodeChildViewsParallel(
NodesToProcess, start, showHigher, childCount + nodeStart, nodeStart - childCount));
00489                         threadJob.Name = i.ToString();
00490                         threadJob.Start();
00491                         threadList.Add(threadJob);
00492                     }
00493                 }
00494                 foreach (Thread t in threadList)
00495                 {
00496                     t.Join();
00497                 }
00498                 cNodes.Sort((x, y) => x.Item1.CompareTo(y.Item1));
00499
00500                 foreach (Tuple<int, string> tup in cNodes)
00501                 {
00502                     output += tup.Item2;
00503                 }
00504             }
00505         }
00506     }
00507     else if (n.Children.Count > 0)
00508     {
00509         foreach (Node n2 in n.Children)
00510         {
00511             if (GotParent)
00512             {
00513                 if (n2.Number == Parent)
00514                 {
00515                     output += CreateNodeView(n2, "node-parent");
00516                 }
00517             }
00518             output += CheckChildren(n2, number, pCount, nodeStart, ref found);
00519         }
00520     }
00521
00522     return output;
00523 }
00524
00530 private void GetParent(Node node, int number)
00531 {
00532     if (!CheckNodeNumber(node, number))
00533     {
00534         foreach (Node n in node.Children)
00535         {
00536             if (CheckNodeNumber(n, number))
00537             {
00538                 Parent = node.Number;
00539                 GotParent = true;
00540             }
00541             else
00542             {

```

```

00543         GetParent(n, number);
00544     }
00545     }
00546 }
00547 }
00548
00557 private string CreateNodeView(Node n, string type,int leftVal,int topVal)
00558 {
00559     string node = "";
00560
00561     if(type == "node")
00562     {
00563         if(GotParent)
00564         {
00565             leftVal = leftVal + 400;
00566         }
00567     }
00568     if(type == "node-child")
00569     {
00570         leftVal = leftVal + 400;
00571     }
00572     node += "<div id='" + n.Number + "' class='" + type + " type ui-draggable ui-selectee'
style='left:" + leftVal + "px; top:" + topVal + "px;'>";
00573     node += "<div class='head'><span><button class='nameBtn' onclick='GetNode(\"+n.
Number+\")'>" + n.Name + "</button></span></div>";
00574     if (!string.IsNullOrEmpty(n.Value))
00575     {
00576         node += string.Format("<div class='blockR'><p>Value</p></div><div
class=comment><span>{0}</span></div>", n.Value);
00577     }
00578     if(n.Comments.Count > 0)
00579     {
00580         node += "<div><p class='text-center'>Comments</p></div>";
00581
00582         foreach(string com in n.Comments)
00583         {
00584             node += "<div class='comment'>" + com + "</div>";
00585         }
00586     }
00587     if (n.Attributes.Count > 0)
00588     {
00589         node += "<div class='attribute'><div class='aHeader'><p><button><i class='fa
fa-plus'></i></button>Attributes</p></div><div class='options'>";
00590         foreach (Attribute a in n.Attributes)
00591         {
00592             node += string.Format("<div class='blockR'><p>{0}</p></div><div
class='comment'><p>{1}</p></div>", a.Name, a.Value);
00593         }
00594         node += "</div>";
00595     }
00596     node += "</div></div>";
00597     return node;
00598 }
00599
00606 private string CreateNodeView(Node n, string type)
00607 {
00608     string node = "";
00609     int leftVal = left;
00610     if (type == "node")
00611     {
00612         if (GotParent)
00613         {
00614             left = left + 400;
00615             leftVal = left;
00616         }
00617     }
00618     if (type == "node-child")
00619     {
00620         leftVal = leftVal + 400;
00621     }
00622     node += "<div id='" + n.Number + "' class='" + type + " type ui-draggable ui-selectee'
style='left:" + leftVal + "px; top:" + top + "px;'>";
00623     node += "<div class='head'><span><button class='nameBtn' onclick='GetNode(" + n.
Number + ")'>" + n.Name + "</button></span></div>";
00624     if (!string.IsNullOrEmpty(n.Value))
00625     {
00626         node += string.Format("<div class='blockR'><p>Value</p></div><div
class=comment><span>{0}</span></div>", n.Value);
00627     }
00628     if (n.Comments.Count > 0)
00629     {
00630         node += "<div><p class='text-center'>Comments</p></div>";
00631
00632         foreach (string com in n.Comments)
00633         {
00634             node += "<div class='comment'>" + com + "</div>";
00635         }

```



```

00636         }
00637         if (n.Attributes.Count > 0)
00638         {
00639             node += "<div class='attribute'><div class='aHeader'><p><button><i class='fa
fa-plus'></i></button>Attributes</p></div><div class='options'>";
00640             foreach (Attribute a in n.Attributes)
00641             {
00642                 node += string.Format("<div class='blockR'><p>{0}</p></div><div
class='comment'><p>{1}</p></div>", a.Name, a.Value);
00643             }
00644             node += "</div>";
00645         }
00646         node += "</div></div>";
00647
00648         if (type == "node-child")
00649         {
00650             top = top + 130;
00651         }
00652         return node;
00653     }
00654
00661     private bool CheckNodeNumber(Node n, int number)
00662     {
00663         return n.Number.Equals(number);
00664     }
00665 }
00666 }

```

## 6.11 WebServiceCSharp/OSXJVClasses/ProcessDocument.cs File Reference

### Classes

- class [OSXJV.Classes.ProcessDocument](#)  
*Class the Processes the document*

### Namespaces

- namespace [OSXJV.Classes](#)

## 6.12 ProcessDocument.cs

```

00001 using Newtonsoft.Json;
00002 using System;
00003 using System.Collections.Generic;
00004 using System.Linq;
00005 using System.Text.RegularExpressions;
00006 using System.Threading;
00007 using System.Xml.Linq;
00008
00009 namespace OSXJV.Classes
00010 {
00011
00015     public class ProcessDocument
00016     {
00020         private XDocument document;
00021
00025         private Node node = new Node();
00026
00030         private List<Tuple<Node, int>> ProcessedElements = new List<Tuple<Node, int>>();
00031
00035         private List<Thread> ThreadList = new List<Thread>();
00036
00040         private string type;
00041         private Thread th;
00042
00046         int count;
00047
00053         private ProcessDocument(XDocument doc, string type)
00054         {
00055             document = doc;

```

```

00056         this.type = type;
00057     }
00058
00064     private void ProcessComment(XComment e, Node node)
00065     {
00066         string s = "";
00067         s = Regex.Replace(e.Value, @"^[^\\w\\s\\.@-]", "");
00068         node.Comments.Add(s);
00069     }
00070
00077     public static ProcessDocument GetProcess(string data, string type)
00078     {
00079         if (string.IsNullOrEmpty(data) || string.IsNullOrEmpty(type))
00080         {
00081             throw new ArgumentException();
00082         }
00083         try
00084         {
00085             XDocument doc = null;
00086             doc = Prepare(data, type);
00087             return new ProcessDocument(doc, type);
00088         }
00089         catch (System.Xml.XmlException e)
00090         {
00091             throw e;
00092         }
00093     }
00094
00100     private void ProcessText(XText e, Node n)
00101     {
00102         n.Value = e.Value;
00103     }
00104
00111     private static XDocument Prepare(string data, string type)
00112     {
00113
00114         if (type.Equals("JSON"))
00115             return new XDocument(JsonConvert.DeserializeXmlNode(data, "Root", false).Root.FirstNode);
00116         else if (type.Equals("XML") || type.Equals("HTML"))
00117             return XDocument.Parse(data);
00118
00119         return null;
00120     }
00121
00127     public Node Process()
00128     {
00129         if (document.Nodes() != null)
00130         {
00131             foreach (XNode n in document.Nodes())
00132             {
00133                 switch (n.NodeType)
00134                 {
00135                     case System.Xml.XmlNodeType.Element:
00136                         count++;
00137                         ProcessElement(XElement.Parse(n.ToString()), node);
00138                         break;
00139                     case System.Xml.XmlNodeType.Comment:
00140                         ProcessComment(n as XComment, node);
00141                         break;
00142                     case System.Xml.XmlNodeType.Text:
00143                         ProcessText(n as XText, node);
00144                         break;
00145                     case System.Xml.XmlNodeType.Notations:
00146                         break;
00147                     case System.Xml.XmlNodeType.EndElement:
00148                         break;
00149                     default:
00150                         break;
00151                 }
00152             }
00153         }
00154         //SortArray(ref node);
00155         document = null;
00156         return node;
00157     }
00158
00165     private Node ProcessElement(XElement e, Node node)
00166     {
00167         if (node.Number == 0)
00168         {
00169             node.Number = count;
00170         }
00171         if (!node.Visited)
00172         {
00173
00174             node.Name = e.Name.LocalName;

```

```

00175         foreach (XmlAttribute ax in e.Attributes())
00176         {
00177             if (ax.Name == "id")
00178             {
00179                 node.Name = node.Name + " #" + ax.Value;
00180             }
00181
00182             if (type == "HTML")
00183             {
00184                 if (ax.IsNamespaceDeclaration)
00185                     continue;
00186
00187                 Attribute att = new Attribute();
00188                 att.Name = ax.Name.LocalName;
00189                 att.Value = ax.Value;
00190                 node.Attributes.Add(att);
00191             }
00192         }
00193
00194         if (e.Nodes() != null)
00195         {
00196             foreach (XmlNode n in e.Nodes())
00197             {
00198                 switch (n.NodeType)
00199                 {
00200                     case System.Xml.XmlNodeType.EndElement:
00201                         break;
00202                     case System.Xml.XmlNodeType.Element:
00203                         count++;
00204                         Node n2 = new Node();
00205                         node.Children.Add(ProcessElement(XElement.Parse(n.ToString()), n2));
00206                         break;
00207                     case System.Xml.XmlNodeType.Comment:
00208                         ProcessComment(n as XComment, node);
00209                         break;
00210                     case System.Xml.XmlNodeType.Text:
00211                         ProcessText(n as XText, node);
00212                         break;
00213                     case System.Xml.XmlNodeType.Notation:
00214                         break;
00215
00216                     default:
00217                         break;
00218                 }
00219             }
00220         }
00221         node.Visited = true;
00222         return node;
00223     }
00224
00232 private Node ProcessElement(XElement e, Node node, ref int nodeNumber)
00233 {
00234     if (!node.Visited)
00235     {
00236         if (node.Number == 0)
00237         {
00238             node.Number = nodeNumber;
00239         }
00240         if (!node.Visited)
00241         {
00242
00243             node.Name = e.Name.LocalName;
00244             foreach (XmlAttribute ax in e.Attributes())
00245             {
00246                 if (ax.Name == "id")
00247                 {
00248                     node.Name = node.Name + " #" + ax.Value;
00249                 }
00250
00251                 if (type == "HTML")
00252                 {
00253                     if (ax.IsNamespaceDeclaration)
00254                         continue;
00255
00256                     Attribute att = new Attribute();
00257                     att.Name = ax.Name.LocalName;
00258                     att.Value = ax.Value;
00259                     node.Attributes.Add(att);
00260                 }
00261             }
00262
00263             if (e.Nodes() != null)
00264             {
00265                 foreach (XmlNode n in e.Nodes())
00266                 {
00267                     switch (n.NodeType)
00268                     {

```

```

00269             case System.Xml.XmlNodeType.EndElement:
00270                 break;
00271             case System.Xml.XmlNodeType.Element:
00272                 nodeNumber++;
00273                 Node n2 = new Node();
00274                 node.Children.Add(ProcessElement(XElement.Parse(n.ToString()), n2,
ref nodeNumber));
00275                 break;
00276             case System.Xml.XmlNodeType.Comment:
00277                 ProcessComment(n as XComment, node);
00278                 break;
00279             case System.Xml.XmlNodeType.Text:
00280                 ProcessText(n as XText, node);
00281                 break;
00282             case System.Xml.XmlNodeType.Notations:
00283                 break;
00284             default:
00285                 break;
00286         }
00287     }
00288 }
00289 }
00290 node.Visited = true;
00291 }
00292 return node;
00293 }
00294
00301 private Node ProcessRoot(XElement e, Node node)
00302 {
00303     node.Number = 1;
00304
00305     if (!node.Visited)
00306     {
00307         node.Name = e.Name.LocalName;
00308         foreach (XAttribute ax in e.Attributes())
00309         {
00310             if (ax.Name == "id")
00311             {
00312                 node.Name = node.Name + " #" + ax.Value;
00313             }
00314
00315             if (type == "HTML")
00316             {
00317                 if (ax.IsNamespaceDeclaration)
00318                     continue;
00319             }
00320             Attribute att = new Attribute();
00321             att.Name = ax.Name.LocalName;
00322             att.Value = ax.Value;
00323             node.Attributes.Add(att);
00324         }
00325     }
00326     node.Visited = true;
00327     return node;
00328 }
00329
00330
00336 private void ProcessDocumentParallelInit(XDocument doc,int start)
00337 {
00338     int nodeNum = start;
00339
00340     Node node = new Node();
00341     if (doc.Root.Nodes() != null)
00342     {
00343         List<XNode> list = doc.Root.Nodes().ToList();
00344         foreach (XNode n in doc.Root.Nodes())
00345         {
00346             switch (n.NodeType)
00347             {
00348                 case System.Xml.XmlNodeType.Element:
00349                     nodeNum++;
00350                     Node n2 = new Node();
00351                     node.Children.Add(ProcessElement(XElement.Parse(n.ToString()), n2, ref
nodeNum));
00352                     break;
00353                 case System.Xml.XmlNodeType.Comment:
00354                     ProcessComment(n as XComment, node);
00355                     break;
00356                 case System.Xml.XmlNodeType.Text:
00357                     ProcessText(n as XText, node);
00358                     break;
00359                 case System.Xml.XmlNodeType.Notations:
00360                     break;
00361                 case System.Xml.XmlNodeType.EndElement:
00362                     break;
00363                 default:
00364                     break;

```

```

00365         }
00366     }
00367 }
00368     document = null;
00369     ProcessedElements.Add(new Tuple<Node, int>(node, start));
00370 }
00371
00372 public Node ProcessParallel(int pCount = 4)
00373 {
00374     node = ProcessRoot(document.Root, node);
00375
00376     int nodeCount = document.Root.Nodes().Count();
00377
00378     if (nodeCount <= pCount)
00379     {
00380         return Process();
00381     }
00382     else if (nodeCount > pCount)
00383     {
00384         List<XNode> List = document.Root.Nodes().ToList();
00385         int spread = 0;
00386
00387         spread = (int)Math.Ceiling((double)nodeCount / (double)pCount);
00388
00389         int totalNodes = 1;
00390
00391         for (int i = 0; i < pCount; i++)
00392         {
00393             int neg = 0;
00394             int start = totalNodes;
00395             if ((spread * (i+1)) > nodeCount)
00396             {
00397                 neg = nodeCount - (spread * (i + 1));
00398             }
00399
00400             List<XNode> list = List.GetRange((spread * i), spread + neg);
00401             XElement root = new XElement("Root", list);
00402             XDocument doc = new XDocument(root);
00403
00404             (th = new Thread(() => ProcessDocumentParallelInit(doc, start))).Start();
00405
00406             ThreadList.Add(th); //Add to Threads list to keep recored of threads running
00407             totalNodes += root.Descendants().Count(); //Increment start position.
00408         }
00409         document = null;
00410         foreach (Thread t in ThreadList)
00411         {
00412             t.Join(); //Wait for threads to join
00413         }
00414         ProcessedElements.Sort((x, y) => x.Item2.CompareTo(y.Item2)); //Sort List by start index so
00415         they are in order.
00416
00417         foreach (Tuple<Node, int> tup in ProcessedElements)
00418         {
00419             foreach (Node n in tup.Item1.Children)
00420             {
00421                 node.Children.Add(n);
00422             }
00423         }
00424         return node;
00425     }
00426 }
00427 }
00428 }
00429 }
00430 }
00431 }
00432 }
00433 }
00434 }

```

## 6.13 WebServiceCSharp/OSXJVClasses/Request.cs File Reference

### Classes

- class [OSXJV.Classes.Request](#)  
A object containing the document to process, filename and type.

### Namespaces

- namespace [OSXJV.Classes](#)

## 6.14 Request.cs

```
00001 using System;
00002
00003 namespace OSXJV.Classes
00004 {
00005     public class Request
00006     {
00007         private string filename;
00008
00009         private string type;
00010
00011         private string data;
00012
00013         private Request(string filename, string type, string data)
00014         {
00015             this.filename = filename;
00016             this.type = type;
00017             this.data = data;
00018         }
00019
00020         public static Request GetRequest(string filename, string type, string data)
00021         {
00022             string Type = "";
00023             if (string.IsNullOrEmpty(filename) || string.IsNullOrEmpty(type) || string.IsNullOrEmpty(data))
00024                 throw new ArgumentException();
00025             if (type.Equals("text/xml") || type.Equals("application/xml"))
00026             {
00027                 Type = "XML";
00028             }
00029             else if (type.Equals("text/html"))
00030             {
00031                 Type = "HTML";
00032             }
00033             else if (type.Equals("application/json") || type.Equals("application/octet-stream"))
00034             {
00035                 Type = "JSON";
00036             }
00037             return new Request(filename, Type, data);
00038         }
00039
00040         public string Filename
00041         {
00042             get
00043             {
00044                 return filename;
00045             }
00046             set
00047             {
00048                 filename = value;
00049             }
00050         }
00051
00052         public string Type
00053         {
00054             get
00055             {
00056                 return type;
00057             }
00058             set
00059             {
00060                 type = value;
00061             }
00062         }
00063
00064         public string Data
00065         {
00066             get
00067             {
00068                 return data;
00069             }
00070             set
00071             {
00072                 data = value;
00073             }
00074         }
00075     }
00076 }
```

## 6.15 WebServiceSharp/OSXJVClasses/Response.cs File Reference

### Classes

- class [OSXJV.Classes.Response](#)  
*The Object containing data to send to the client*

### Namespaces

- namespace [OSXJV.Classes](#)

## 6.16 Response.cs

```

00001 using System;
00002 using System.Text;
00003
00004 namespace OSXJV.Classes
00005 {
00009     public class Response
00010     {
00014         public byte[] data = null;
00015
00019         public int status;
00020
00024         public string mime;
00025         //static string format = "yyyy-MM-dd HH:mm:ss";
00026
00027
00034         private Response(int status,string mime,byte[] buffer)
00035         {
00036             this.status = status;
00037             this.data = buffer;
00038             this.mime = mime;
00039         }
00040
00048         public static Response GetResponse(int status,string type,byte[] data)
00049         {
00050             if(string.IsNullOrEmpty(type))
00051                 throw new ArgumentException("Type cannot be Null or empty");
00052
00053             if (status.Equals(null))
00054                 throw new ArgumentException("Status cannot be Null");
00055             else
00056                 if (status == 0)
00057                     throw new ArgumentException("Status cannot be 0");
00058
00059             if (data == null)
00060                 throw new ArgumentException("Data cannot be null");
00061             else
00062                 if (data.Length == 0)
00063                     throw new ArgumentException("No data, use invalid or error response");
00064
00065             return new Response(status, type, data);
00066         }
00067
00074         public static Response GetResponseJSON(int status,byte[] data)
00075         {
00076             if (status.Equals(null))
00077                 throw new ArgumentException("Status cannot be Null");
00078             else
00079                 if (status == 0)
00080                     throw new ArgumentException("Status cannot be 0");
00081
00082             if (data == null)
00083                 throw new ArgumentException("Data cannot be null");
00084             else
00085                 if (data.Length == 0)
00086                     throw new ArgumentException("No data, use invalid or error response");
00087
00088             return new Response(status,"application/json", data);
00089         }
00090
00097         public static Response GetResponseXML(int status, byte[] data)

```

```

00098     {
00099         if (status.Equals(null))
00100             throw new ArgumentException("Status cannot be Null");
00101         else
00102             if(status == 0)
00103                 throw new ArgumentException("Status cannot be 0");
00104
00105         if (data == null)
00106             throw new ArgumentException("Data cannot be null");
00107         else
00108             if (data.Length == 0)
00109                 throw new ArgumentException("No data, use invalid or error response");
00110
00111         return new Response(status, "text/xml", data);
00112     }
00113
00118     public static Response GetErrorResponse(string message)
00119     {
00120         byte[] res = Encoding.UTF8.GetBytes(message);
00121         return new Response(400, "text/html", res);
00122     }
00123
00128     public static Response GetInvalidRequestResponse()
00129     {
00130         return new Response(405, "text/html", new byte[0]);
00131     }
00132 }
00133 }

```

## 6.17 WebServiceCSharp/OSXJVClasses/Validation.cs File Reference

### Classes

- class [OSXJV.Classes.Validation](#)  
*Perform validation on document*

### Namespaces

- namespace [OSXJV.Classes](#)

## 6.18 Validation.cs

```

00001 using Newtonsoft.Json;
00002 using Newtonsoft.Json.Linq;
00003 using System;
00004 using System.IO;
00005 using System.Xml;
00006
00007 namespace OSXJV.Classes
00008 {
00012     public class Validation
00013     {
00017         private static Validation inst;
00018
00022         private Validation(){}
00023
00028         public static Validation GetInstance()
00029         {
00030             if (inst != null)
00031                 return inst;
00032             else
00033                 return (inst = new Validation ());
00034         }
00044         public bool CheckDocument(string data, string type)
00045         {
00046             if(string.IsNullOrEmpty(data) || string.IsNullOrEmpty(type))
00047             {
00048                 throw new ArgumentException("Data or Type cannot be Null");
00049             }
00050

```



```

00051         if (type.Equals("XML") || type.Equals("HTML"))
00052         {
00053             XmlReaderSettings settings = new XmlReaderSettings();
00054             settings.DtdProcessing = DtdProcessing.Parse;
00055             settings.MaxCharactersFromEntities = 2048;
00056             using (XmlReader xr = XmlReader.Create(new StringReader(data), settings))
00057             {
00058                 try
00059                 {
00060                     while (xr.Read()) { }
00061                     return true;
00062                 }
00063                 catch (XmlException ex)
00064                 {
00065                     throw ex;
00066                 }
00067             }
00068         }
00069         else if (type.Equals("JSON"))
00070         {
00071             try
00072             {
00073                 JToken.Parse(data);
00074                 return true;
00075             }
00076             catch (JsonReaderException ex)
00077             {
00078                 throw new JsonReaderException(ex.Message);
00079             }
00080         }
00081         throw new ArgumentException("Invalid data or type");
00082     }
00083 }
00084 }
00085 }

```

## 6.19 WebServiceCSharp/OSXJVServer.cs File Reference

### Classes

- class [OSXJV.Server.OSXJVServer](#)  
*HTTPServer that process the incoming requests.*

### Namespaces

- namespace [OSXJV.Server](#)

## 6.20 OSXJVServer.cs

```

00001 using System;
00002 using System.Text;
00003 using System.Net;
00004 using System.Threading;
00005 using System.IO;
00006 using HttpMultipartParser;
00007 using Newtonsoft.Json.Linq;
00008 using Newtonsoft.Json;
00009 using OSXJV.Classes;
00010
00011 namespace OSXJV.Server
00012 {
00013     public class OSXJVServer
00014     {
00015         private int port = 8082;
00016
00017         public static bool running = false; //sets if the server is currently running
00018
00019         private HttpListener listener;
00020
00021         private Thread serverThread = null;
00022     }
00023 }
00024
00025
00026
00027
00028
00029
00030
00031
00032
00033

```

```

00034
00038     private Thread cacheThread = null;
00039
00043     public OSXJVServer()
00044     {
00045         listener = new HttpListener();
00046         listener.Prefixes.Add("http://localhost:" + port + "/"); //change if need be
00047     }
00048
00054     public bool Start(string cachePath, string loggerPath)
00055     {
00056         bool success = false;
00057
00058         success = CacheManager.Setup(cachePath);
00059         success = Logger.Setup(loggerPath);
00060
00061         serverThread = new Thread(new ThreadStart(Run)); //Server thread
00062         cacheThread = new Thread(new ThreadStart(ManageCache)); //Cache manage thread
00063         try
00064         {
00065             serverThread.Start();
00066             cacheThread.Start();
00067         }
00068         catch (Exception e)
00069         {
00070             throw e;
00071         }
00072
00073         success = cacheThread.IsAlive;
00074         success = serverThread.IsAlive;
00075
00076         return success;
00077     }
00078
00082     public bool Stop()
00083     {
00084         if (listener != null)
00085             if (listener.IsListening)
00086                 listener.Abort();
00087
00088         if (serverThread != null)
00089         {
00090             serverThread.Join();
00091             serverThread = null;
00092         }
00093
00094         return serverThread == null ? true : false;
00095     }
00096
00100     public void Run()
00101     {
00102         running = true;
00103         listener.Start();
00104
00105         while (listener.IsListening)
00106         {
00107
00108             Console.WriteLine("Waiting");
00109
00110             //Wait for Listener
00111             IAsyncResult result = listener.BeginGetContext(new AsyncCallback(ListenerCallback),
00112 listener);
00113             result.AsyncWaitHandle.WaitOne();
00114
00115             if (result.CompletedSynchronously)
00116                 Console.WriteLine("Completed Synchronously");
00117         }
00118     }
00119
00120     //Asynchronous Handler
00125     private void ListenerCallback(IAsyncResult result)
00126     {
00127         HttpListener listener = (HttpListener)result.AsyncState;
00128         HttpListenerContext context = listener.EndGetContext(result);
00129         try
00130         {
00131             HandleClient(context);
00132         }
00133         catch (Exception e)
00134         {
00135             Logger.GetInstance().WriteError(e.Message);
00136             context.Response.StatusCode = 500;
00137             context.Response.Close();
00138         }
00139     }
00140

```

```

00141
00142 //Handles the client request
00147 private void HandleClient(HttpListenerContext c)
00148 {
00149     switch(c.Request.HttpMethod)
00150     {
00151         case "POST":
00152             Post(HandlePost(c.Request),c.Response);
00153             break;
00154         case "GET":
00155             Post(HandleGet(c.Request), c.Response);
00156             break;
00157         case "OPTIONS":
00158             HandleOptions(c.Response);
00159             c.Response.Close();
00160             break;
00161         default:
00162             Post(Response.GetInvalidRequestResponse(), c.
Response);
00163             break;
00164     }
00165 }
00166
00171 private void HandleOptions(HttpListenerResponse response)
00172 {
00173     response.AddHeader("Access-Control-Allow-Headers", "Content-Type, Accept, X-Requested-With");
00174     response.AddHeader("Access-Control-Allow-Methods", "POST");
00175     response.AddHeader("Access-Control-Allow-Methods", "GET");
00176     response.AddHeader("Access-Control-Max-Age", "1728000");
00177     response.AppendHeader("Access-Control-Allow-Origin", "*");
00178 }
00179
00186 public Request GetFormData(Stream input)
00187 {
00188     string request = "";
00189     MultipartFormDataParser parser = new MultipartFormDataParser(input);
00190     if (parser.Files.Count > 0)
00191     {
00192         using (StreamReader ms = new StreamReader(parser.Files[0].Data))
00193         {
00194             request = ms.ReadToEnd();
00195         }
00196     }
00197     else
00198     {
00199         throw new InvalidOperationException();
00200     }
00201     return Request.GetRequest(parser.Files[0].FileName, parser.Files[0].
ContentType, request);
00202 }
00203
00210 private Request GetFileData(Stream input,string type)
00211 {
00212     string request = "";
00213     using (StreamReader ms = new StreamReader(input))
00214     {
00215         request = ms.ReadToEnd();
00216     }
00217     string filename = "temp";
00218
00219     if (type == "text/xml")
00220         filename += ".xml";
00221     else if(type == "application/json")
00222         filename += ".json";
00223     else
00224         filename += ".html";
00225
00226     return Request.GetRequest(filename,type, request);
00227 }
00228
00234 private Response HandlePost(HttpListenerRequest req)
00235 {
00236
00237     JObject eRes = new JObject();
00238
00239     if (SegmentNormalize(req.RawUrl).Equals("Process"))
00240     {
00241         if (req.HasEntityBody)
00242         {
00243
00244
00245             Request r = null;
00246             try
00247             {
00248                 r = GetData(req);
00249                 if (r == null)
00250                     return Response.GetInvalidRequestResponse();

```

```

00251         }
00252         catch
00253         {
00254             return Response.GetInvalidRequestResponse();
00255         }
00256
00257
00258
00259         try
00260         {
00261             Validation.GetInstance().
CheckDocument(r.Data, r.Type);
00262         }
00263         catch (Exception e)
00264         {
00265             eRes.Add("Error", e.Message);
00266             return Response.GetErrorResponse(eRes.ToString());
00267         }
00268
00269         string id = Guid.NewGuid().ToString();
00270         ProcessDocument pro = ProcessDocument.
GetProcess(r.Data, r.Type);
00271         Node n = pro.ProcessParallel();
00272         Output o = new Output(n); //new output object
00273         try
00274         {
00275             CacheManager.GetInstance().
saveFile(id, JsonConvert.SerializeObject(n));
00276             JObject response = new JObject();
00277
00278             n = null; //remove node as its completed;
00279
00280             response.Add("filename", id);
00281             response.Add("grid", o.CreateGrid());
00282             response.Add("view", o.CreateView());
00283
00284
00285
00286             byte[] bytes = Encoding.UTF8.GetBytes(response.ToString());
00287             return Response.GetResponse(200, "application/json", bytes);
00288         }
00289         catch (Exception e)
00290         {
00291             Logger.GetInstance().WriteError(e.Message);
00292             eRes.Add("Error", "Error Creating Response");
00293             return Response.GetErrorResponse(eRes.ToString());
00294         }
00295
00296     }
00297     eRes.Add("Error", "No File Recieved By Server");
00298     return Response.GetErrorResponse(eRes.ToString());
00299 }
00300 else if (req.RawUrl.Equals("/Output"))
00301 {
00302     return Response.GetInvalidRequestResponse();
00303 }
00304 else
00305     return Response.GetInvalidRequestResponse();
00306 }
00307
00313 private Response HandleGet(HttpListenerRequest req)
00314 {
00315     if (SegmentNormalize(req.Url.Segments[1]).Equals("Process"))
00316     {
00317         if (req.Url.Segments.Length == 4)
00318         {
00319
00320             Node cached;
00321             try
00322             {
00323                 cached = JsonConvert.DeserializeObject<Node>(
CacheManager.GetInstance().getFile(req.Url.Segments[2]));
00324             }
00325             catch (Exception e)
00326             {
00327                 Logger.GetInstance().WriteError(e.Message);
00328                 JObject eRes = new JObject();
00329                 eRes.Add("Error", "Error Creating Response");
00330                 return Response.GetErrorResponse(eRes.ToString());
00331             }
00332             Output o = new Output(cached);
00333             JObject response = new JObject();
00334             response.Add("view", o.CreateView(int.Parse(req.Url.Segments[3])));
00335             byte[] bytes = Encoding.UTF8.GetBytes(response.ToString());
00336             return Response.GetResponse(200, "application/json", bytes);
00337
00337         }

```

```

00338         else if (req.Url.Segments.Length == 5)
00339         {
00340             Node cached;
00341             try
00342             {
00343                 {
00344                     cached = JsonConvert.DeserializeObject<Node>(
CacheManager.GetInstance().getFile(req.Url.Segments[2]));
00345                 }
00346                 catch (Exception e)
00347                 {
00348                     Logger.GetInstance().WriteError(e.Message);
00349                     JObject eRes = new JObject();
00350                     eRes.Add("Error", "Error Creating Response");
00351                     return Response.GetErrorResponse(eRes.ToString());
00352                 }
00353                 Output o = new Output(cached);
00354                 JObject response = new JObject();
00355                 response.Add("view", o.CreateView(int.Parse(SegmentNormalize(req.Url.Segments
[3])), 4, int.Parse(SegmentNormalize(req.Url.Segments[4]))));
00356                 byte[] bytes = Encoding.UTF8.GetBytes(response.ToString());
00357                 return Response.GetResponse(200, "application/json", bytes);
00358             }
00359             else
00360                 return Response.GetInvalidRequestResponse();
00361         }
00362         //If it got here its an invalid response.
00363         return Response.GetInvalidRequestResponse();
00364     }
00365
00372     private void SaveFile(string id, Node nodes)
00373     {
00374         if(nodes == null || string.IsNullOrEmpty(id))
00375         {
00376             throw new ArgumentException();
00377         }
00378         try
00379         {
00380             {
00381                 CacheManager.GetInstance().saveFile(id, JsonConvert.
SerializeObject(nodes));
00382             }
00383             catch (Exception e)
00384             {
00385                 Logger.GetInstance().WriteError(e.Message);
00386             }
00387         }
00388
00395     private void Post(Response res,HttpListenerResponse stream)
00396     {
00397         if (res == null || stream == null)
00398             throw new ArgumentException("Response or Client Stream cannot be NULL");
00399
00400         HandleOptions(stream);
00401         stream.ProtocolVersion = new Version(1, 1);
00402         stream.StatusCode = res.status;
00403         stream.ContentType = res.mime;
00404         stream.ContentLength64 = res.data.Length;
00405         stream.OutputStream.Write(res.data, 0, res.data.Length);
00406         stream.Close();
00407     }
00408
00414     private Request GetData(HttpListenerRequest req)
00415     {
00416         Request r = null;
00417
00418         if (req.ContentType.Contains("application/x-www-form-urlencoded"))
00419         {
00420             r = GetFormData(req.InputStream);
00421         }
00422         else if (req.ContentType.Contains("application/json") || req.ContentType.Contains("
application/oclet-stream"))
00423         {
00424             r = GetFileData(req.InputStream, "application/json");
00425         }
00426         else if (req.ContentType.Contains("application/xml") || req.ContentType.Contains("text/xml"))
00427         {
00428             r = GetFileData(req.InputStream, "text/xml");
00429         }
00430         return r;
00431     }
00432
00438     private string SegmentNormalize(string input)
00439     {
00440         return input.Replace("/", "");
00441     }

```

```

00442
00446     private void ManageCache()
00447     {
00448         while (true)
00449         {
00450             Thread.Sleep(3600000);
00451             try
00452             {
00453                 CacheManager.ManageCache();
00454             }
00455             catch (Exception e)
00456             {
00457                 try
00458                 {
00459                     Logger.GetInstance().WriteError(e.Message);
00460                 }
00461                 catch
00462                 {
00463                     Console.WriteLine("Logger and Cache Manager not setup");
00464                 }
00465             }
00466         }
00467     }
00468 }
00469 }
00470 }

```

## 6.21 WebServiceCSharp/Program.cs File Reference

### Classes

- class [WebServer.Program](#)  
*The Initialiser*

### Namespaces

- namespace [WebServer](#)

## 6.22 Program.cs

```

00001 using System;
00002 using System.Threading;
00003 using System.IO;
00004 using OSXJV.Classes;
00005 using OSXJV.Server;
00006
00007 namespace WebServer
00008 {
00012     class Program
00013     {
00018         static void Main(string[] args)
00019         {
00020
00021             if (args.Length == 0)
00022             {
00023                 Console.WriteLine("Using Default Cache Directory Path and Logger Directory Path");
00024                 string dir = Directory.GetCurrentDirectory();
00025                 Array.Resize(ref args, 2);
00026                 args[0] = dir + "/Cache/";
00027                 args[1] = dir + "/Logger/";
00028                 if (!Directory.Exists(args[0]))
00029                     Directory.CreateDirectory(args[0]);
00030                 if (!Directory.Exists(args[1]))
00031                     Directory.CreateDirectory(args[1]);
00032             }
00033
00034             if (args[0] == args[1])
00035             {
00036                 Console.WriteLine("Cache location and Log location is the same. Please enter two different
locations");

```

```

00037         }
00038         else
00039         {
00040             try
00041             {
00042                 OSXJVServer s = new OSXJVServer();
00043                 s.Start(args[0], args[1]);
00044             }
00045             catch (Exception e)
00046             {
00047                 Console.WriteLine(e.Message);
00048                 Console.WriteLine("Press any key to exit");
00049                 Console.Read();
00050             }
00051         }
00052     }
00053 }
00054 }
00055 }

```

## 6.23 WebServiceCSharp/Properties/AssemblyInfo.cs File Reference

### 6.24 AssemblyInfo.cs

```

00001 using System.Reflection;
00002 using System.Runtime.CompilerServices;
00003 using System.Runtime.InteropServices;
00004
00005 // General Information about an assembly is controlled through the following
00006 // set of attributes. Change these attribute values to modify the information
00007 // associated with an assembly.
00008 [assembly: AssemblyTitle("WebServiceCSharp")]
00009 [assembly: AssemblyDescription("")]
00010 [assembly: AssemblyConfiguration("")]
00011 [assembly: AssemblyCompany("")]
00012 [assembly: AssemblyProduct("WebServiceCSharp")]
00013 [assembly: AssemblyCopyright("Copyright © 2016")]
00014 [assembly: AssemblyTrademark("")]
00015 [assembly: AssemblyCulture("")]
00016
00017 // Setting ComVisible to false makes the types in this assembly not visible
00018 // to COM components. If you need to access a type in this assembly from
00019 // COM, set the ComVisible attribute to true on that type.
00020 [assembly: ComVisible(false)]
00021
00022 // The following GUID is for the ID of the typelib if this project is exposed to COM
00023 [assembly: Guid("a57034df-dc0f-44ce-bb8a-cddafe37db17")]
00024
00025 // Version information for an assembly consists of the following four values:
00026 //
00027 //      Major Version
00028 //      Minor Version
00029 //      Build Number
00030 //      Revision
00031 //
00032 // You can specify all the values or you can default the Build and Revision Numbers
00033 // by using the '*' as shown below:
00034 // [assembly: AssemblyVersion("1.0.*")]
00035 [assembly: AssemblyVersion("1.0.0.0")]
00036 [assembly: AssemblyFileVersion("1.0.0.0")]

```





# Index

- Attributes
  - OSXJV::Classes::Node, [25](#)
- attributes
  - OSXJV::Classes::Node, [24](#)
- cNodes
  - OSXJV::Classes::Output, [61](#)
- CacheManager
  - OSXJV::Classes::CacheManager, [12](#)
- cacheThread
  - OSXJV::Server::OSXJVServer, [44](#)
- CheckChildren
  - OSXJV::Classes::Output, [48](#)
- CheckDocument
  - OSXJV::Classes::Validation, [88](#)
- CheckNodeNumber
  - OSXJV::Classes::Output, [50](#)
- Children
  - OSXJV::Classes::Node, [26](#)
- children
  - OSXJV::Classes::Node, [24](#)
- Close
  - OSXJV::Classes::CacheManager, [12](#)
  - OSXJV::Classes::Logger, [19](#)
- Comments
  - OSXJV::Classes::Node, [26](#)
- comments
  - OSXJV::Classes::Node, [24](#)
- count
  - OSXJV::Classes::ProcessDocument, [73](#)
- CreateExtraNode
  - OSXJV::Classes::Output, [50](#), [51](#)
- CreateGrid
  - OSXJV::Classes::Output, [52](#)
- CreateNodeChildViewsParallel
  - OSXJV::Classes::Output, [53](#)
- CreateNodeView
  - OSXJV::Classes::Output, [54](#), [55](#)
- CreatePreviousNode
  - OSXJV::Classes::Output, [56](#)
- CreateView
  - OSXJV::Classes::Output, [57](#)
- CreateViewSingle
  - OSXJV::Classes::Output, [59](#)
- Data
  - OSXJV::Classes::Request, [80](#)
- data
  - OSXJV::Classes::Request, [79](#)
  - OSXJV::Classes::Response, [86](#)
- document
  - OSXJV::Classes::ProcessDocument, [73](#)
- Filename
  - OSXJV::Classes::Request, [80](#)
- filename
  - OSXJV::Classes::Request, [79](#)
- GetData
  - OSXJV::Server::OSXJVServer, [30](#)
- GetErrorResponse
  - OSXJV::Classes::Response, [83](#)
- getFile
  - OSXJV::Classes::CacheManager, [13](#)
- GetFileData
  - OSXJV::Server::OSXJVServer, [31](#)
- GetFormData
  - OSXJV::Server::OSXJVServer, [32](#)
- GetInstance
  - OSXJV::Classes::CacheManager, [14](#)
  - OSXJV::Classes::Logger, [19](#)
  - OSXJV::Classes::Validation, [90](#)
- GetInvalidRequestResponse
  - OSXJV::Classes::Response, [83](#)
- GetParent
  - OSXJV::Classes::Output, [60](#)
- GetProcess
  - OSXJV::Classes::ProcessDocument, [64](#)
- GetRequest
  - OSXJV::Classes::Request, [78](#)
- GetResponse
  - OSXJV::Classes::Response, [84](#)
- GetResponseJSON
  - OSXJV::Classes::Response, [85](#)
- GetResponseXML
  - OSXJV::Classes::Response, [86](#)
- GotParent
  - OSXJV::Classes::Output, [61](#)
- GridGetChidren
  - OSXJV::Classes::Output, [60](#)
- HandleClient
  - OSXJV::Server::OSXJVServer, [32](#)
- HandleGet
  - OSXJV::Server::OSXJVServer, [33](#)
- HandleOptions
  - OSXJV::Server::OSXJVServer, [35](#)
- HandlePost
  - OSXJV::Server::OSXJVServer, [35](#)
- Inst

- OSXJV::Classes::CacheManager, 17
- inst
  - OSXJV::Classes::Logger, 22
  - OSXJV::Classes::Validation, 90
- left
  - OSXJV::Classes::Output, 61
- listener
  - OSXJV::Server::OSXJVServer, 45
- ListenerCallback
  - OSXJV::Server::OSXJVServer, 38
- location
  - OSXJV::Classes::Logger, 22
- Logger
  - OSXJV::Classes::Logger, 19
- Main
  - WebServer::Program, 75
- ManageCache
  - OSXJV::Classes::CacheManager, 15
  - OSXJV::Server::OSXJVServer, 39
- mime
  - OSXJV::Classes::Response, 87
- Name
  - OSXJV::Classes::Attribute, 10
  - OSXJV::Classes::Node, 26
- name
  - OSXJV::Classes::Attribute, 10
  - OSXJV::Classes::Node, 25
- Node
  - OSXJV::Classes::Node, 24
- node
  - OSXJV::Classes::ProcessDocument, 73
- nodes
  - OSXJV::Classes::Output, 62
- Number
  - OSXJV::Classes::Node, 26
- number
  - OSXJV::Classes::Node, 25
- OSXJV.Classes, 7
- OSXJV.Classes.Attribute, 9
- OSXJV.Classes.CacheManager, 11
- OSXJV.Classes.Logger, 18
- OSXJV.Classes.Node, 23
- OSXJV.Classes.Output, 46
- OSXJV.Classes.ProcessDocument, 62
- OSXJV.Classes.Request, 76
- OSXJV.Classes.Response, 81
- OSXJV.Classes.Validation, 87
- OSXJV.Server, 7
- OSXJV.Server.OSXJVServer, 28
- OSXJV::Classes::Attribute
  - Name, 10
  - name, 10
  - Value, 10
  - value, 10
- OSXJV::Classes::CacheManager
  - CacheManager, 12
  - Close, 12
  - getFile, 13
  - GetInstance, 14
  - Inst, 17
  - ManageCache, 15
  - path, 17
  - saveFile, 15
  - Setup, 16
- OSXJV::Classes::Logger
  - Close, 19
  - GetInstance, 19
  - inst, 22
  - location, 22
  - Logger, 19
  - Setup, 20
  - WriteError, 21
- OSXJV::Classes::Node
  - Attributes, 25
  - attributes, 24
  - Children, 26
  - children, 24
  - Comments, 26
  - comments, 24
  - Name, 26
  - name, 25
  - Node, 24
  - Number, 26
  - number, 25
  - Value, 27
  - value, 25
  - Visited, 27
  - visited, 25
- OSXJV::Classes::Output
  - cNodes, 61
  - CheckChildren, 48
  - CheckNodeNumber, 50
  - CreateExtraNode, 50, 51
  - CreateGrid, 52
  - CreateNodeChildViewsParallel, 53
  - CreateNodeView, 54, 55
  - CreatePreviousNode, 56
  - CreateView, 57
  - CreateViewSingle, 59
  - GetParent, 60
  - GotParent, 61
  - GridGetChildren, 60
  - left, 61
  - nodes, 62
  - Output, 47
  - Parent, 62
  - top, 62
- OSXJV::Classes::ProcessDocument
  - count, 73
  - document, 73
  - GetProcess, 64
  - node, 73
  - Prepare, 65

- Process, [66](#)
- ProcessComment, [66](#)
- ProcessDocument, [64](#)
- ProcessDocumentParallelInit, [67](#)
- ProcessElement, [68](#), [69](#)
- ProcessParallel, [70](#)
- ProcessRoot, [71](#)
- ProcessText, [72](#)
- ProcessedElements, [73](#)
- th, [74](#)
- ThreadList, [74](#)
- type, [74](#)
- OSXJV::Classes::Request
  - Data, [80](#)
  - data, [79](#)
  - Filename, [80](#)
  - filename, [79](#)
  - GetRequest, [78](#)
  - Request, [78](#)
  - Type, [80](#)
  - type, [80](#)
- OSXJV::Classes::Response
  - data, [86](#)
  - GetErrorResponse, [83](#)
  - GetInvalidRequestResponse, [83](#)
  - GetResponse, [84](#)
  - GetResponseJSON, [85](#)
  - GetResponseXML, [86](#)
  - mime, [87](#)
  - Response, [82](#)
  - status, [87](#)
- OSXJV::Classes::Validation
  - CheckDocument, [88](#)
  - GetInstance, [90](#)
  - inst, [90](#)
  - Validation, [88](#)
- OSXJV::Server::OSXJVServer
  - cacheThread, [44](#)
  - GetData, [30](#)
  - GetFileData, [31](#)
  - GetFormData, [32](#)
  - HandleClient, [32](#)
  - HandleGet, [33](#)
  - HandleOptions, [35](#)
  - HandlePost, [35](#)
  - listener, [45](#)
  - ListenerCallback, [38](#)
  - ManageCache, [39](#)
  - OSXJVServer, [30](#)
  - port, [45](#)
  - Post, [40](#)
  - Run, [41](#)
  - running, [45](#)
  - SaveFile, [41](#)
  - SegmentNormalize, [42](#)
  - serverThread, [45](#)
  - Start, [43](#)
  - Stop, [44](#)
  - OSXJVServer
    - OSXJV::Server::OSXJVServer, [30](#)
  - OSXJV, [7](#)
  - Output
    - OSXJV::Classes::Output, [47](#)
  - Parent
    - OSXJV::Classes::Output, [62](#)
  - path
    - OSXJV::Classes::CacheManager, [17](#)
  - port
    - OSXJV::Server::OSXJVServer, [45](#)
  - Post
    - OSXJV::Server::OSXJVServer, [40](#)
  - Prepare
    - OSXJV::Classes::ProcessDocument, [65](#)
  - Process
    - OSXJV::Classes::ProcessDocument, [66](#)
  - ProcessComment
    - OSXJV::Classes::ProcessDocument, [66](#)
  - ProcessDocument
    - OSXJV::Classes::ProcessDocument, [64](#)
  - ProcessDocumentParallelInit
    - OSXJV::Classes::ProcessDocument, [67](#)
  - ProcessElement
    - OSXJV::Classes::ProcessDocument, [68](#), [69](#)
  - ProcessParallel
    - OSXJV::Classes::ProcessDocument, [70](#)
  - ProcessRoot
    - OSXJV::Classes::ProcessDocument, [71](#)
  - ProcessText
    - OSXJV::Classes::ProcessDocument, [72](#)
  - ProcessedElements
    - OSXJV::Classes::ProcessDocument, [73](#)
  - Request
    - OSXJV::Classes::Request, [78](#)
  - Response
    - OSXJV::Classes::Response, [82](#)
  - Run
    - OSXJV::Server::OSXJVServer, [41](#)
  - running
    - OSXJV::Server::OSXJVServer, [45](#)
  - SaveFile
    - OSXJV::Server::OSXJVServer, [41](#)
  - saveFile
    - OSXJV::Classes::CacheManager, [15](#)
  - SegmentNormalize
    - OSXJV::Server::OSXJVServer, [42](#)
  - serverThread
    - OSXJV::Server::OSXJVServer, [45](#)
  - Setup
    - OSXJV::Classes::CacheManager, [16](#)
    - OSXJV::Classes::Logger, [20](#)
  - Start
    - OSXJV::Server::OSXJVServer, [43](#)
  - status
    - OSXJV::Classes::Response, [87](#)

- Stop
  - OSXJV::Server::OSXJVServer, [44](#)
- th
  - OSXJV::Classes::ProcessDocument, [74](#)
- ThreadList
  - OSXJV::Classes::ProcessDocument, [74](#)
- top
  - OSXJV::Classes::Output, [62](#)
- Type
  - OSXJV::Classes::Request, [80](#)
- type
  - OSXJV::Classes::ProcessDocument, [74](#)
  - OSXJV::Classes::Request, [80](#)
- Validation
  - OSXJV::Classes::Validation, [88](#)
- Value
  - OSXJV::Classes::Attribute, [10](#)
  - OSXJV::Classes::Node, [27](#)
- value
  - OSXJV::Classes::Attribute, [10](#)
  - OSXJV::Classes::Node, [25](#)
- Visited
  - OSXJV::Classes::Node, [27](#)
- visited
  - OSXJV::Classes::Node, [25](#)
- WebServer, [8](#)
- WebServer.Program, [75](#)
- WebServer::Program
  - Main, [75](#)
- WebServiceCSharp/OSXJVClasses/Attribute.cs, [91](#)
- WebServiceCSharp/OSXJVClasses/CacheManager.cs,  
[92](#)
- WebServiceCSharp/OSXJVClasses/Logger.cs, [93](#)
- WebServiceCSharp/OSXJVClasses/Node.cs, [94](#)
- WebServiceCSharp/OSXJVClasses/Output.cs, [96](#)
- WebServiceCSharp/OSXJVClasses/ProcessDocument.  
cs, [103](#)
- WebServiceCSharp/OSXJVClasses/Request.cs, [107](#),  
[108](#)
- WebServiceCSharp/OSXJVClasses/Response.cs, [109](#)
- WebServiceCSharp/OSXJVClasses/Validation.cs, [110](#)
- WebServiceCSharp/OSXJVServer.cs, [111](#)
- WebServiceCSharp/Program.cs, [116](#)
- WebServiceCSharp/Properties/AssemblyInfo.cs, [117](#)
- WriteError
  - OSXJV::Classes::Logger, [21](#)