BreadCrumb Trail Control v2.0 Component Specification

1. Design

Many websites include breadcrumb trails to help their users navigate complex hierarchical structures. This component provides an ASP.NET web control, which renders a breadcrumb trail from a bound data source, allowing web authors to add breadcrumb trails to their pages without additional coding.

The appearance of the breadcrumbs is controlled by several of the control's properties. The NodeStyle property is used as the style attribute for each normal node in the breadcrumb trail; RootNodeStyle and CurrentNodeStyle are used for those two special nodes. MouseOverNodeStyle is the CSS class to specify the css style when the mouse is over any node. PathSeparator is used to separate each node, and PathSeparatorStyle is its style. The configuration manager has been introduced to allow the application to setup the defaults over all pages (and eliminate the need to duplicate the information in every page).

Additionally, the user can control the processing of the breadcrumb trail by determining whether the nodes are represented from root to current or current to root. The disposition of the errors can be set to silently ignore errors, render the errors in place or throw the error to the IIS engine. The type of implementation can be set to allow the application to control the exact processing that will be used and can include either the included implementation or custom implementations that the application includes.

In addition to the above, the control can be used to dynamically adjust the included site map based on the usage patterns:

1. Query strings are now included in both the node matching (the site map XML document includes full regex matching capabilities in identifying the nodes) and generated trails.
2. The last query string for each node in the current path will be 'remembered' and rendered back into the trail – regardless of how the site map has been setup.
3. If a page has not explicitly been specified in the site map, it will automatically be appended to the end of the current trail.

This enhancement has separated the logic in this component into 3 basic areas:

1. *The current node discovery process*. This process provides the discovery process that the control uses to determine what the current page (otherwise known as the node). This component includes an implementation to discover the current node from the HttpRequest using the full Url and query strings.

2. *The path discovery process.* The path discovery process that will attempt to discover the path from the current node (as identified above) and the root node. Two implementations of this have been included:

    a. A dynamic path discovery process that will use the XML site map as a base and then include dynamic capability (as described above) on top of it.
    b. A static path discovery process that will mimic the behavior of the v1.0 component (with the exception that regex patterns can be specified also).

3. *The path writing process.* The path writing process allows the type of node rendering for the path to be fully pluggable to allow various ways of representing the path. This component provides 2 implementations of this:
    a. A simple html anchor implementation to generate html anchors.
    b. An order list implementation where each item is represented as a separate list item.

Please note that this is almost a complete rewrite of the V1.0 component and the class diagrams have not been marked up to reflect what has changed because everything has essentially changed.

1.1 Design Patterns

Strategy Pattern – is used extensively in this application to provide various implementations of functionality that can easily be swapped

1.2 Industry Standards

- HTML and CSS are used to render the breadcrumb trail and control its appearance. This component allows the web designer to specify the CSS style classes to be used. The related CSS style sheet will need to be included on the webpage outside of this component.

- The control can generate the breadcrumb trail from an XML document describing the website's layout. See the related ExampleBreadCrumbSource_2.0.xml and BreadCrumbTrailSchema_2.0.xsd in the docs directory for details. Please note that this file/schema is compatible and interchangeable with the Java Bread Crumb Trail v2.0 component

1.3 Control parameters:

The breadcrumb trail control can have most of its attributes defined in a configuration file for easy assigning of common attributes (see the configurable column). This control has the following attributes assignable to the control:

Attribute	Description	Required	Configurable
PathSeparator	The characters to use to separate nodes in the path	No	Yes
PathSeparatorStyle	The CSS style class of the separator	No	Yes
NodeStyle	The CSS style class of the non-root and non-current nodes	No	Yes
RootNodeStyle	The CSS style class of the root node	No	Yes
CurrentNodeStyle	The CSS style class of the current node	No	Yes
MouseOverStyle	The CSS style class when the mouse hovers over any node	No	Yes
SilentErrors	Whether exceptions should silently be ignored	No	Yes
PathDirection	The direction of the trail	No	Yes
NodeDiscoveryKey	The ObjectFactory key to use to create the INodeDiscovery implementation	No*	Yes*
PathDiscoveryKey	The ObjectFactory key to use to create the IPathDiscovery implementation	No*	Yes*
PathWriterKey	The ObjectFactory key to use to create the IPathWriter implementation	No*	Yes*
Title	The title of the current page	No	No
DataSource	The data source to use for the path.	No	No

* Required if the configuration property was not set

The DataSource, when specified, is considered a path override and the normal path discovery process will **not** be done. The DataSource can be one of the following sources:

1. BreadCrumbNode[] – an array of bread crumb nodes to use for the specific page
2. IList of BreadCrumbNode(s) – a list of bread crumb nodes for the specific page
3. DataSet where the first DataTable matches the following
4. DataTable with a “Title” and “Url” column. BreadCrumbNode(s) will be defined from those columns and used as the path.

1.4 Required Algorithms

1.4.1 *The CSS Style*

The web page designer should include a CSS file defining the various styles that will be used by this component. An example file could look like:

```
.root { color : red }
.current { color : green }
.node { color : #10ABCD }
.hover { color : #AA0CBF }
.mouseout { color : #CCBA00 }
.path { color : #BBEBC0 } <span>
```

The various style(s), shown above, can then be specified on the control (or via the property sheet in VS.Net) like:

```
<brd:BreadCrumbTrailControl
     NodeStyle=”.node”
     CurrentNodeStyle=”.current”
      … etc …
>
</brd:BreadCrumbTrailControl>
```

1.4.2 *Query Strings*

The old component ignored the query string portion of the Url. This version will handle query strings in the node discovery, the path discovery and in the path writer.

The node discovery will simply create the node with a url that has the query string appended to it (i.e. “www.topcoder.com?id=1”). The node discovery will sort the query strings by the key to allow easier, consistent regex pattern matching creation.

The path discovery will can then use the query string portion as part of the matching (in other words, the regex used to match nodes can include patterns on the query string). The developer should note that if query strings are used in the site, the site map must include query string matching in the xml file (or a “.*” regex pattern on the end of the url to match all query strings for that url). Please also note that differing query strings (on the same url) will create different nodes on the trail if the node is a dynamic node (i.e. doesn’t match any of the regex pattern).

The node formatter will then include the query string as part of the links that are rendered.

The path writer process will then write out the url, with any included query string, as part of the path.

1.4.3 Overall process

The overall process that the BreadCrumbTrailControl will follow is:

1. Determine the current node.
2. Determine the current path to the root.
3. Write out the path

Each of these three steps will be completed by the various implementations. The BreadCrumbTrailControl relies on the Object Factory component to create the implementations at render time. The application can customize the implementation by specifying unique keys (see Object Factory documentation) to each of the implementations in the configuration file. The key to use (and thus the implementation to use) can then be specified as properties to the control

There are two enumerations that affect the processing:

1. PathDirection determines how to iterate the path – either from root to current or current to root (see the Path Writer discussion for more details)
2. ErrorDisposition determines how the control handles any exceptions. SilentlyIgnore will ignore any exceptions, RenderException will render any exceptions to the html document and ThrowException will cause the exception to be rethrown to the IIS engine.

1.4.3.1 Determining the current node

The current node is by using the object factory, create the INodeDiscovery class specified by the “NodeDiscoveryKey” property and call GetCurrentNode with the title (if specified) and the page context.

If the HttpRequestNodeDiscovery implementation will simply interrogate the Page context for the url and query strings, sort the query strings by name and then return a BreadCrumbNode with the passed title and the url (url + sorted query string).

1.4.3.2 Determining the current path to the root

We determine the current path to the root node by calling an IPathDiscovery implementation with the current node (defined in 1.5.2.1):

1. Using the object factory, create the IPathDiscovery class specified by the “PathDiscoveryKey” property.
2. Call GetPath with the current node
3. If a path is returned, skip back to step 3 of the overall process
4. If the path is null, simply return (couldn’t find a path and nothing should be rendered)
5. If an exception is thrown, handle the exception as described above.

Please note that this process will be ignored if the application specifies a DataSource for the page (which is a path override for the current page).

1.4.3.3 Writing out the path

This is where things get interesting for the control. The object factory will create the IPathWriter class specified by the "PathWriterKey" property and a call to the WritePath method will be done to write out the path.

There are two ways of processing the path – backwards and forwards. We can either iterate the path at the 0th index position (processing root to current) or at the size-1 position (processing current to root). The processing will then either run forwards or backwards. Regardless of the direction of the path – each node will be formatted by the implementation. The anchor implementation will simply write out an html anchor for each node (using the correct style). The ordered list implementation will create a new list item for each node (each list item will be an anchor itself).

1.4.4 Dynamic Path Discovery

The dynamic path discovery is really the heart of this component. The class will attempt to discover the path of the current node to the root using a combination of a 'learned' path and a static site map (see 1.5.4 for information about the static site map).

The developer should note that the DynamicPathDiscovery class, when it needs to discover the underlying site map, will call the Object Factory with the ISiteDiscovery type – presumably getting back the XmlPathDiscovery class to read and parse the xml representing the static site map. The user can modify the Object Factory configuration to return a different ISiteDiscovery type if it wishes to customize the source of the underlying site information.

Both of those variables are kept in session variables that will be reused every time the class is called for a path.

The site map is simply a mapping of INodeMatcher (key) object to an IList (value) of INodeMatcher objects. The key is considered the parent of all the objects in the list (the children). Note: the developer is free to also create a reverse mapping (child to all parents) if it will help their implementation. The only restriction the developer has (in assigning new state variables) is to define a public static variable for that state information (like CURRENT_REVERSE_SITEMAP).

The current path is the path the class has 'learned' so far. Each element of the path is of NodePair type and contains the actual node (including any contextual information [like the query string]) and the INodeMatcher that the node matched (which will be used to access the site map).

The rules that we will be following for generating a path is:

1. If the current node matches any of the nodes on the current path (as defined by the node matcher), we take the path from the current root to the current node – discarding all nodes after that one.
2. If current node matches no nodes in the site map, it's considered a 'new' node that is linked from the last site. In other words, we append the node to the current path (if there is one, if not – we simply return null because we don't know the path).
3. Otherwise, we perform a shortest path search on the site map to a root and then try to match it up with as much of the current path as possible.

The underlying INodeMatcher for this implementation will be a regex matcher (which will be fairly slow on large sites). Because of that, we perform two quick checks that cover two common situations:

1. We iterate the current path backwards and see if the node matches the INodeMatcher for any element. If the element has a null matcher, compare the node's URL directly to determine if it matches. If a match is made and it is not the last NodePair on the chain, a new NodePair is generated using the current node and the node matcher (effectively discarding the old node's contextual information) and then discard all elements past the matching node. A new path from root to that node is returned. This will cover the situations where the user clicked on the trail (which should match somewhere on the trail) or pressed the back button (which should match the last node).

2. We take the last element in the path and see if the current node matches any of the children of that node. If it does, we simply append a new NodePair to the end of the path and return the new path. This covers the situation where they went to a place that is well defined (in the site map).

3. Failing those two – we will do a shortest path search on the sitemap for any INodeMatcher that matches the node. Example:

   We may iterate the sitemap and find 3 potential candidates (i.e. the current node matched 3 INodeMatcher keys in the site map). We then iterate all the parents of those 3 nodes (this is where a reverse site map may come in handy!). If any of those parents have no parents, then we have found a shortest path to a root node. If they all have parents, we then recursively search the parents of the parents until we have found a root node (or more than one). Note: the developer is free to implement whatever shortest path algorithm they choose – as long as it correctly identifies the shortest path (or paths if there are more than one).
4. We then take the shortest path(s) and try to determine which one has the most commonality with the current path starting with the root node forward. Example:

   Let's say our current path is A->B->C->D->E

Let’s say we found two paths to our current page M:
A->B->L->M
A->B->C->M
Because “A->B->C->M” had the 3 nodes in common with our current path, we choose that path. If we have more than one with the same commonality, choose the first one (where ‘first’ can be defined anyway you choose).

5. Calculate the new path, save it and return it.

6. If the no shortest path is found for the node and we have a current path, we assume this is a new ‘dynamic’ link and simply append the node onto the end of the current path.

7. If no shortest path was found for the node and we have no current path (i.e. the user probably went to this page directly and it’s not listed in the site map), return null indicating we have no idea what the path is.

1.4.5 XML Path Discovery

The Xml Path Discovery object has two functions, it can be used as a static map path discovery (i.e. not dynamic capable) and it can serve as a source for the static site information to other IPathDiscovery implementations (like the Dynamic Path Discovery).

Every node found in the Xml document should appear as a key in the site map with a value of an IList. That list will then hold references to any children that node may have (or be empty if it’s a leaf node).

Let’s take an example file:

```
<bc_node title="MyMain" url="/" pattern="/">
    <bc_node title="Forums" url="/forums" pattern="/forums">
       <bc_node title="Help" url="/forums" pattern="/forums?id.*"/>
    </bc_node>
    <bc_node title="Topics" url="/topics" pattern="/topics"/>
</bc_node>

<bc_node title="AltMain" url="/main" pattern="/">
    <bc_node title="Help" url="/forums" pattern="/forums?id.*"/>
</bc_node>
```

Here we have a “Help” node that has two parents. The resulting site map would look like (using only the title to represent each node):

Key	**Value List**
MyMain	Forums, Topics
Forums	Help
Topics	{ empty }
Help	{ empty }
AltMain	Help

1.4.6 Session Data

The bread trail crumb control will make use of Session data to track the path the user has taken and to store the site map.

The path to the current node (i.e. an array of NodePair[s]) is stored in the Session data by the DynamicPathDiscovery class, as each node has been discovered. This allows the DynamicPathDiscovery class the ability to:

- Provides a quick lookup on the current path if the current node is already on the path (thus preventing any expensive path searching for most cases where the user selects a node on the path or goes backwards on the path.
- Allows the class to build a dynamic path outside of the static site map and remember the trail

Both the DynamicPathDiscovery and XmlPathDiscovery will store the static site map in the Session data to improve performance by only reading and parsing the static information once.

1.5 Component Class Overview

BreadCrumbTrailControl:

This is the main control class that application will use. This control will render the current node to the root node (or vice-versa) using a path separator and specific styles. The application can setup the separator and the various styles either by specifying default values in the configuration manager or by specifying attributes on the string. The object factory will be used to discover the node, the path to the root and the formatting of the path using the key(s) specified on the control.

BreadCrumbTrailControlDesigner:

This class is the control designer for the BreadCrumbTrailControl. The control designer will interact with the IDE to provide a custom designer environment. Currently, this class simply interacts with the BreadCrumbTrailControl to display the Html that it renders.

ErrorDisposition:

This enumeration describes the values that the various error dispositions can be. The enumeration will be used by the breadcrumb trail control to determine how to handle any exceptions that occur. The exceptions can be ignored, can be rendered or can be thrown.

PathDisposition:

This enumeration describes the values the path direction can allow. The path direction is used by the breadcrumb trail control to determine the direction to render the path in (the node is the first node to the last node)

INodeDiscovery:

This interface defines the contract for classes wishing to discover the current node. The implementation should discover, in its own way, the current node when GetCurrentNode is called and return the node representation of that url.

IPathDiscovery:

This interface defines the contract for classes wishing to discover the path from current node to the root node. Implementations of this interface will be called with the current node and the current page context. The implementation is then supposed to discover the complete path from the current node to the root or return null when it can't discover the current path.

BreadCrumbNode:

This class represents the structure for holding the url and the page title of a specific node.

IPathWriter:

This interface defines the contract for classes wishing to write out the discovered path. Implementations of this interface will be called with a writer, the path to write and the breadcrumb trail control itself. The implementation is then responsible for formatting the path in its own way - writing the path out to the HtmlTextWriter

AbstractPathWriter:

This abstract implementation of the IPathWriter will provide some of the common functionality when writing a path. This abstract implementation will provide the following functionality:

HttpRequestNodeDiscovery:

This implementation of the INodeDiscovery will discover the current node from the HttpRequest for the given page. This implementation will return a valid Url (with query string if specified) in the GetCurrentNode method. The query string will be appended to the Url in a sorted order.

RegexNodeMatcher:

This implementation of an INodeMatcher will provide node-matching services given a specific regex pattern. This class will return true when the passed node's url matches the specified pattern.

NodePair:

This class is a typical pair class that holds a paired association of a node matcher to the matching node. The node describes the actual node that matched the node matcher and contains the contextual specific information.

DynamicPathDiscovery:

This implementation of the IPathDiscovery will attempt to create a static site map and then apply dynamic information to that site map. The dynamic information can either be in matching nodes with different query strings or in new nodes that will be assumed to map to the last known node.

XmlPathDiscovery:

This implementation of the IPathDiscovery <strong>and </strong>ISiteDiscovery will attempt to create a static site map from either a filename or string source (both of which is an XML document). This class will parse the document to create a static map of regex matchers (where each node can provide a regex matching). If used as an IPathDiscovery, this provides a static view of the site (i.e. no dynamic capabilities).

INodeMatcher:

Defines the contract for a node matcher. Implementations need to provide three functions:

- A matching function that will take a node and determine if it matches.
- A default title string used for those nodes that match.
- A default url to use if no contextual information is available.

AbstractNodeMatcher:

This abstract implementation of the INodeMatcher interface provides title and url type services to subclasses. This implementation will provide a title and url holder variable and getters.

ISiteDiscovery:

Defines the contract for a data source for the site. Implementations of this interface should create and return a site map (in an IDictionary implementation) consisting of INodeMatcher nodes that describe the site map. The returns IDictionary should have each unique INodeMatcher listed as the key and an IList of INodeMatcher that describe the children of that key.

AnchorPathWriter:

This path writer will write the node and title using the html anchor pattern of <a href='node.Url'>node.Title</a>

OrderedListPathWriter:

This path writer will write the node and title using the ordered list pattern where each rendered node is a separate line item.

1.6 Component Exception Definitions

BreadCrumbException:

Exception thrown in all cases where the BreadCrumbTrailControl cannot correctly parse, format and write breadcrumb trail.

1.7 Thread Safety

The IIS Engine will call the control(s) in a thread safe manner and therefore thread safety isn’t an issue. However, many classes are immutable or have no state information and will naturally be thread-safe.

The only concern would be in the Session data. Session data can be accessed in multiple threads simultaneously if a user has opened up two browser windows under the same session. This component makes no efforts to detect or prevent this situation. However, this should prevent no difficulties to the component. There are two things that are stored in the session:

1. Static site information
2. Current node path

The static site information is a non-changeable property under most circumstances. The only time it is writable is when it is first created. If two threads attempt to create the session variable at the same time, the resulting data will be the same (since both threads would source from the same file) and the last thread 'wins'. The only affect this has would be a slight slowdown on both threads (and only on the first access).

The current node path is more problematic since it's a read/write property. However, according to a PM post in the java version forum (of the Bread Crumb Trail Tag) – we should not be concerned about multiple, same session issues. What will occur if multiple threads read/write the current node path is that the last thread 'wins' again – in other words, the thread that finishes last will be the one setting the new current node path.

2. Environment Requirements

2.1 Environment

- .NET CLR version 1.1;
- C#.NET compiler
- IIS
- Visual Studio.NET, if you want to test the integration with the ASP.NET Designer.

2.2 TopCoder Software Components

- Object Factory v1.0 –is used by the component to create the various implementations in the component.
- Configuration Manager v2.0 - will provide default configuration of the breadcrumb trail control.

NOTE: The default location for TopCoder Software component jars *is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.*

2.3 Third Party Components

None needed

3. Installation and Configuration

3.1 Package Name

TopCoder.Web.UI.WebControl.BreadCrumb

3.2 Configuration Parameters]

The following configuration parameters are available to the BreadCrumbTrailControl under the TopCoder.Web.UI.WebControl.BreadCrumb.BreadCrumbTrailControl namespace

Parameter	**Description**	**Values**
pathSeparator	The path separator that will be used	String, optional defaults to ":"
pathSeparatorStyle	The css style used when rendering the path separator	String, optional, defaults to nothing
NodeStyle	The non-root, non-current node css style	String, optional, defaults to nothing
rootNodeStyle	The root node css style	String, optional, defaults to nothing
currentNodeStyle	The current node css style	String, optional, defaults to nothing
mouseOverStyle	The mouse over css style for all nodes	String, optional, defaults to nothing
pathDirection	The direction of the path between the root node and the current node	Integer, optional, defaults to RootToCurrent
silentErrors	Whether errors should be silently ignored	Integer, optional, defaults to SilentlyIgnore
nodeDiscoveryKey	The object factory key to use when creating the INodeDiscovery implementation	String, required (optional if specified on the control)
pathDiscoveryKey	The object factory key to use when creating the IPathDiscovery implementation	String, required (optional if specified on the control)

pathWriterKey	The object factory key to use when creating the IPathWriter implementation	String, required (optional if specified on the control)

The following classes can have object factory specified constructor parameters (please see the Object Factory documentation on how to set it up).

XmlPathDiscovery

Parameter	**Description**	**Values**
filename	The location of the file that contains the XML site map	String, required

XmlPathDiscovery

Parameter	**Description**	**Values**
source	A string containing either a filename or a direct XML document	String, required
isFile	True if the source is a file,	Boolean, required

3.3 Dependencies Configuration

Set up the web server to serve ASP.NET pages and point it at the build/asp directory.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Copy the content of test_files/iis to your IIS web root.
- Execute ‘nant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

In an ASP.NET page:

```
<%@Register TagPrefix=”tc”
  Namespace="TopCoder.Web.UI.WebControl.BreadCrumb"
  Assembly="TopCoder.Web.UI.WebControl.BreadCrumb”%>
<tc:BreadCrumbTrailControl runat="server" />
```

4.3 Demo

There are several demonstrations of this component that can be done.

For the following demonstrations, assume the users have visited the following pages before the demonstration page:

(Main) http://www.xyz.com

(Forums) http://www.xyz.com/forums

(Help) http://www.xyz.com/forums?id=1

The static site map defines them like:

```
<?xml version="1.0" encoding="UTF-8"?>

<BreadCrumb xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation="BreadCrumbTrailSchema.xsd"
            version="v2">

  <bc_node title="Main" url="/" pattern="/">
        <bc_node title="Forums" url="/forums" pattern="/forums">
           <bc_node title="Help" url="/forums?id=1"
                                 pattern="/forums?id.*"/>
        </bc_node>
    </bc_node>
</BreadCrumb>
```

4.3.1 *Normal demonstration*

The following page "http://www.xyz.com/forums?thread=1" defines the following

```
<%@Register TagPrefix=”tc”
  Namespace="TopCoder.Web.UI.WebControl.BreadCrumb"
  Assembly="TopCoder.Web.UI.WebControl.BreadCrumb”%>

<hmtl><body>
My portable hole stopped working on the floor!

<tc:BreadCrumbTrailControl
   Title=”portable hole”
   runat=”server”/>

// etc etc
</body></html>
```

Would create a bread trail of:
"Main : Forums : Help : portable hole where:
    "Main" would be:
        <a href="/">Main</a>
    "Forums" would be:
        <a href="/forums">Forums</a>
    "Help" would be:

<a href=”/forums?id=1”>Help</a>
“portable hole” would be:
<a href=”/forums?thread=1”>portable hole</a>

4.3.2 Using styles

Assume we have a style sheet called “bread.css” that is defined as:

```
.root { color : red }
.current { color : green }
.node { color : #10ABCD }
.mouseover { color : #CCBA00 }
```

The following page “http://www.xyz.com/forums?thread=1” defines the following

```
<%@Register TagPrefix=”tc”
  Namespace="TopCoder.Web.UI.WebControl.BreadCrumb"
  Assembly="TopCoder.Web.UI.WebControl.BreadCrumb”%>

<hmtl>
<head>
<link rel="stylesheet" type="text/css" href="bread.css">
</head>
<body>
My portable hole stopped working on the floor!

<tc:BreadCrumbTrailControl
   Title=”portable hole”
   NodeStyle=”.node”
   CurrentNodeStyle=”.current”
   RootNodeStyle=”.root”
   MouseOverStyle=”.mouseover”
   PathSeparator=”>>>”
   runat=”server”/>

// etc etc
</body></html>
```

Would create a bread trail of:
“Main >>> Forums >>> Help >>> portable hole where:
“Main” would be:
<a href=”/” class=”.root”
onmouseover=”this.class=’.mouseover’”
onmouseout=”this.class=’.root’”>Main</a>
“Forums” would be:
<a href=”/forums” class=”.node”
onmouseover=”this.class=’.mouseover’”
onmouseout=”this.class=’.node’”>Forums</a>
“Help” would be:
<a href=”/forums?id=1” class=”.node”
onmouseover=”this.class=’.mouseover’”

onmouseout=”this.class=’.node’”>Help</a>
<a href=”/forums?id=1”>Help</a>
“portable hole” would be:
<a href=”/forums?thread=1” class=”.current”
onmouseover=”this.class=’.mouseover’”
onmouseout=”this.class=’.current’”>portable hole</a>

4.3.3 *Ordered List demonstration*

Assume the object factory has been setup to create an instance of OrderedListPathWriter for a key of “orderedlist”.

The following page “http://www.xyz.com/forums?thread=1” defines the following

```
<%@Register TagPrefix=”tc”
  Namespace="TopCoder.Web.UI.WebControl.BreadCrumb"
  Assembly="TopCoder.Web.UI.WebControl.BreadCrumb”%>

<hmtl><body>
My portable hole stopped working on the floor!

<tc:BreadCrumbTrailControl
   Title=”portable hole”
   PathWriterKey=”orderedlist”
   runat=”server”/>

// etc etc
</body></html>
```

Would create a bread trail of:

1. Main
2. Forums
3. Help
4. portable hole

Where:

“Main” would be:
<a href=”/”>Main</a>
“Forums” would be:
<a href=”/forums”>Forums</a>
“Help” would be:
<a href=”/forums?id=1”>Help</a>
“portable hole” would be:
<a href=”/forums?thread=1”>portable hole</a>

4.3.4 *Path Override*

The following demonstrates using a path override to specify a specific path. The following would be the page form:

```
<%@ Register TagPrefix="tc"
Namespace="TopCoder.Web.UI.WebControl.BreadCrumb"
Assembly="BreadCrumbControl" %>

<%@ Page language="c#" Src="ArrayWebForm.aspx.cs"%>
<HTML>
       <body MS_POSITIONING="GridLayout">
           <p>This web page contains a BreadCrumbControl binding to an
Array object </p>
              <form id="Form1" method="post" runat="server">
                     <tc:BreadCrumbTrailControl id="bcc" runat="server"/>
              </form>
       </body>
</HTML>
```

The code will then look like (binding it to a simple array):

```
public class ArrayDocWebForm : System.Web.UI.Page
{
   protected BreadCrumbTrailControl bcc;
   private void Page_Load(object sender, System.EventArgs e)
   {
      BreadCrumbNode[] nodes = new BreadCrumbNode[3];
      nodes[0] = new BreadCrumbNode("Main",
                           "http://www.xyz.com");
      nodes[1] = new BreadCrumbNode("Forums",
                           "http://www.xyz.com/forums");
      nodes[2] = new BreadCrumbNode("Help",
                           "http://www.xyz.com/forums/&id=1");

      bcc.DataSource = nodes;
      bcc.DataBind();
  }

  override protected void OnInit(EventArgs e)
  {
    InitializeComponent();
    base.OnInit(e);
  }

  private void InitializeComponent()
  {
    this.Load += new System.EventHandler(this.Page_Load);
  }
}
```

The resulting path should look similar to the path shown in 4.3.1

4.3.5 *Multiple parents*

Assume the static site map is defined like:

```
<?xml version="1.0" encoding="UTF-8"?>

<BreadCrumb xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation="BreadCrumbTrailSchema.xsd"
            version="v2">

  <bc_node title="MyMain" url="/" pattern="/">
        <bc_node title="Forums" url="/forums" pattern="/forums">
```

```
            <bc_node title="Help" url="/forums" pattern="/forums?id.*"/>
        </bc_node>
    </bc_node>

    <bc_node title="AltMain" url="/main" pattern="/">
        <bc_node title="Help" url="/forums" pattern="/forums?id.*"/>
    </bc_node>

</BreadCrumb>
```

As you can see – the help forums has two parents: “MyMain” and “AltMain”

If the user types in “http://www.xyz.com/forums?id=1”, the component will find the shortest path back to a root and select the following trail:

“AltMain : Help”

5. Future Enhancements

- Provide different node discovery implementations (absolute url versus relative url)
- Provide more path discovery implementations (database, config manager, etc)
- Provide more path writers (table driven, indenting, etc)