# BEA AquaLogic® .NET Portlet Toolkit

## AquaLogic Interaction (ALI) Development Guide

Version 1.1
Revised: March 2008

# Contents

# BEA Documentation and Resources

The following documentation and resources are available from BEA.

| Resource | Description |
|---|---|
| Installation Guide | This guide describes the prerequisites (such as required software) and procedures for installing and upgrading .NET Portlet Toolkit components.<br><br>It is available on *edocs.bea.com/alui/dotnetappaccelerator/docs11*. |
| Release Notes | The release notes provide information about new features, issues addressed, and known issues in the release.<br><br>They are available on *edocs.bea.com/alui/dotnetappaccelerator/docs11*. |
| Development Guides | The development guide provides detailed information on authoring portlets using the .NET Portlet Toolkit.<br><br>They are available on *edocs.bea.com/alui/dotnetappaccelerator/docs11*. |
| Additional Developer Guides, Articles, API Documentation, Blogs, Newsgroups, and Sample Code | These resources are provided for developers on the BEA dev2dev site (*dev2dev.bea.com*). They describe how to build custom applications using AquaLogic User Interaction and how to customize AquaLogic User Interaction products and features. |

| Resource | Description |
|---|---|
| AquaLogic User Interaction (ALUI) and AquaLogic Business Process Management (ALBPM) Support Center | The ALUI and ALBPM Support Center is a comprehensive repository for technical information on ALUI and ALBPM products. From the Support Center, you can access products and documentation, search knowledge base articles, read the latest news and information, participate in a support community, get training, and find tools to meet most of your ALUI and ALBPM-related needs. The Support Center encompasses the following communities:<br><br>**Technical Support**<br><br>Submit online service requests, check the status of your requests, search the knowledge base, access documentation, and download maintenance packs and hotfixes.<br><br>**User Group**<br><br>Participate in user groups; view webinars, presentations, the CustomerConnection newsletter, and the Upcoming Events calendar.<br><br>**Product Center**<br><br>Download product updates, maintenance packs, and patches; view the Product Interoperability matrix (supported third-party products and interoperability between products).<br><br>**Developer Center**<br><br>Download developer tools, view code samples, access technical articles, and participate in discussions.<br><br>**Education Services**<br><br>Review the available education options, then choose courses by role and delivery method (Live Studio, Public Classroom Training, Remote Classroom, Private Training, or Self-Paced eLearning).<br><br>**Profile Center**<br><br>Manage your implementation details, local user accounts, subscriptions, and more.<br><br>If you do not see the Support Center when you log in to *one.bea.com/support*, contact *ALUISupport@bea.com* or *ALBPMSupport@bea.com*for the appropriate access privileges. |

| Resource | Description |
|---|---|
| Technical Support | If you cannot resolve an issue using the above resources, BEA Technical Support is happy to assist. Our staff is available 24 hours a day, 7 days a week to handle all your technical support needs.<br>E-mail: *ALUISupport@bea.com* or *ALBPMSupport@bea.com*<br>Phone Numbers:<br>USA, Canada +1 866.262.7586 or +1 415.263.1696<br>EMEA +44 1494 559127<br>Asia Pacific +61 2.9931.7822<br>Australia/NZ +61 2.9923.4030<br>Singapore +1 800.1811.202 |

# Authoring ALI Portlets

## Creating an ALI Portlet Using the .NET Portlet Toolkit

To create a new ALI portlet, use the Microsoft Visual Studio 2005 templates provided with the .NET Portlet Toolkit. The templates are pre-configured to include references to required BEA assemblies, including the IDK, Portlet API libraries, the ALI PTSpy logging infrastructure and the IDK user profile provider.

A new portlet project includes a simple portlet page in the Default.aspx/Default.aspx.cs files. This page extends the `ALIPortletPage` base class, which provides access to preferences, user information and IDK resources such as the PortletRequest and PortletResponse.

Follow the steps below to create a new portlet using the `ALIPortletPage` template.

1. Create a new ALI portlet project in Microsoft Visual Studio 2005:
   a) To create a new project, click **File ➤ New ➤ Web Site** .
   b) Select the language option: C#.
   c) In the My Templates list, select ALI Portlet Project.
   d) Enter a name for the project (for example, "simpleALIWeb").

2. Create a new portlet page:
   a) In the **Solution Explorer**, right-click on the root of the project and select Add New Item….
   b) In the **My Templates** list, select ALI Portlet Page.
3. Add the necessary functionality to the portlet page:
   a) *Manipulating ALI Preferences Using the .NET Portlet Toolkit* on page 10
   b) *Accessing User Profile Information Using the .NET Portlet Toolkit* on page 11
   c) *Accessing Additional Portlet Information Using the .NET Portlet Toolkit* on page 12

## Manipulating ALI Preferences Using the .NET Portlet Toolkit

To access and update ALI preferences using the .NET Portlet Toolkit, use preference attributes or a preference collection.

**Preference attributes** provide type specific read-write binding of an ALI preference to a page member variable (field). One of the following access modifiers is required on all local fields that are bound to a preference using a preference attribute: [protected | internal | protected internal | public]. The modifier "private" is not supported. Since this is the default modifier for a field if none is specified, the modifier must not be omitted. Preference attributes can be bound to fields with types of String, int, float, bool, or DateTime. Values for fields marked with preference attributes are set after the OnLoad() page event; any changes are persisted during the OnUnload() page event.

`ALIPortletPage`exposes a preference attribute for each type of preference: `PortletPreferenceAttribute`, `CommunityPreferenceAttribute`, `CommunityPortletPreferenceAttribute`, `AdminPreferenceAttribute`, `UserPreferenceAttribute`, and `SessionPreferenceAttribute`. All preference attributes have two optional properties:

- Key: A string that provides the key name of the property to which the member variable is bound. If you do not include this property, the name of the preference defaults to the name of the variable. Key values must be unique within a page.
- DefaultValue: A string that provides the value to use when the preference is unavailable or has not yet been set.

**Note:** Private variables (including those with no permissions modifier) cannot be seen by reflection and are not supported. You must include a permissions modifier of `public`, `protected` or `internal`.

```
[PortletPreference]
protected int myIntPref;

[PortletPreference(Key = "lastAccessed", DefaultValue =
"2008-01-01")]
protected DateTime lastAccessedDateTime;
```

**Preference collections** allow you to enumerate all of the preference values in the request using the PreferenceCollection.Keys property. `ALIPortletPage` provides a preference collection for each type of preference: `PortletPreferences`, `CommunityPreferences`, `CommunityPortletPreferences`, `AdminPreferences`, `UserPreferences`, and `SessionPreferences`.

```
String myPrefValue = this.PortletPreferences["MyPrefName"];
this.PortletPreferences["MyPrefName"] = myPrefValue;
```

The examples above use `PortletAttribute` and `PortletPreferences`, but could be replaced with Admin*, User*, Community*, CommunityPortlet* or Session* to read preferences of the corresponding type. (Most preferences can also be set from a portlet page, but Admin preferences must be set from an preference page.)

For a full description of the API, see the complete class documentation for the ALI section of the Portlet API, installed into the Microsoft Visual Studio 2005 Integrated Help System.

For details on the types of preferences available in ALI, see *Portlet Settings and Preferences* in the ALI 6.0 development documentation.

## Accessing User Profile Information Using the .NET Portlet Toolkit

To access user profile information using the .NET Portlet Toolkit, add the properties to the Web.config file and reference them by name in the portlet page.

ASP.NET 2.0 introduced the Provider model as a design pattern for plugging data providers into the framework. The .NET Portlet Toolkit Portlet API uses the ProfilerProvider API to expose user profile information sent by ALI. For more information about the ProfileProvider model, see the *MSDN documentation*.

The Web.config file for a .NET portlet project contains a `<profile>` element that defines the user profile properties supported by a Provider. In order for user profile properties from the source application to be accessible from the ASP.NET Profile object, they must be registered in the Web.config file.

**Note:** User Profile information is read-only and can only be modified through the source application.

1. Add the user profile properties to a `<properties>` element within the `<profile>` element in the Web.config file for the portlet project. Each property must be defined with a name and a type. If you provide a default value, it will be used if the property is not available.

   **Note:** The name of the property **must** match the name sent by the source application.

   ```
   <properties>
   <add name="CustomPropertiesTitle" type="string" defaultValue="No
    Title"/>
   <add name="MyProfileName" type="String" defaultValue="Guest"/>
   <add name="MyProfileAge" type="Int32"/>
   </properties>
   ```

2. Reference the property in the portlet page by name using the Profile object.

   ```
   string title = Profile.CustomPropertiesTitle
   string name = Profile.MyProfileName;
   int age = Profile.MyProfileAge;
   ```

## Accessing Additional Portlet Information Using the .NET Portlet Toolkit

To access portlet-specific information from ALI, use the .NET Portlet Toolkit objects in the `ALIPortletPage` base class.

In addition to preferences and user profile information, a variety of other portlet-specific information is available from a portlet page that extends `ALIPortletPage`. This page provides examples. For a complete list, see the API documentation.

- To access the current portal session (IRemoteSession), use the following code:

  ```
  IRemoteSession session = this.RemotePortalSession;
  ```

- To access the current user's name from ALI, use the following code:

  ```
  String name = User.Name;
  ```

- To access the host page URI, use the following code:

  ```
  URI hostPage = HostPageURI;
  ```

## Creating ALI Preference Pages Using the .NET Portlet Toolkit

To create a preference page, use the VisualStudio.NET templates provided with the .NET Portlet Toolkit. The templates are pre-configured to include references to required BEA assemblies, including the IDK, Portlet API libraries, the ALI PTSpy logging infrastructure and the IDK user profile provider.

The ALIPreferencePage template provides the same functionality as an ALIPortletPage and includes additional features for rendering and editing a preference page. By default, the page renders a title bar with "Finish" and "Cancel" buttons that match the standard ALI preference page style. The appearance and functionality of the title bar can be changed by overriding the `DrawTitleBar()` method from the base class.

To complete the preference page, you must implement four methods:

| Method Name | Description |
|---|---|
| `GetTitleBarText()` | Read-only property to get the text for the title bar. |
| `GetFinishButtonText()` | Read-only property to get the text for the finish button. |
| `GetCancelButtonText()` | Read-only property to get the text for the cancel button. |

| Method Name | Description |
|---|---|
| `SavePreferences()` | Method implemented to update preference values for any preferences changed on the page. |

## Logging .NET Application Accelerator Activity

To view logs of .NET Application Accelerator activity, use ALI Logging Spy.

ALI Logging Spy is automatically installed with ALI or installed on a remote server as part of the ALI Logging Utilities, a stand-alone package available for download on bea.dev2dev. ALI Logging Spy listens for log messages transmitted using network broadcast. By configuring a list of message senders, you can listen in on log traffic from multiple BEA applications in a single console.

The .NET Application Accelerator includes four logging message senders:

| Logging Message Sender | Description |
|---|---|
| .NET WSRP Producer | The stand-alone web application designed to serve as a WSRP producer for WLP. This name is not configurable and transmits all messages related to the WSRP Producer. The WSRP producer "loggingName" entry in Web.config is not currently used. |
| ALIPortletProject | Transmits log messages from the portlet ASP.NET web application. The name of this message sender is defined in the Web.config file of any ASP.NET portlet project built using the .NET Portlet Toolkit. The name of the sender defaults to the project name entered when the project was created, but can be changed by modifying the value attribute of the following entry: `<add key="ptedk.LoggingApplicationName" value="ALIPortletProject"/>` |

| Logging Message Sender | Description |
|---|---|
| BEA Portlet API | Transmits log messages from the .NET Portlet API. |
| ALI .NET Application Accelerator | Reserved for future use. No logging traffic is sent by this sender. |

To add a .NET Application Accelerator message sender to your Spy configuration, follow the instructions below.

1. Open ALI Logging Spy.
2. Select **View** ➤ **Set Filters** .
3. In the **Filter Settings** dialog, click **Edit** ➤ **Add Message Sender** and choose a message sender (see the table above).

For more information on configuring ALI Logging Spy, see *ALI Logging Utilities* in the ALI 6.0 development documentation.

# Using the Web Control Consumer (WCC)

## About the AquaLogic Web Control Consumer (WCC)

The AquaLogic Web Control Consumer provides support for using standard .NET Web Controls in portlet development.

Two problems arise with using .NET Web Controls out of the box for portlet development:

- Client-side scripting problems due to naming collisions: .NET assumes it occupies the whole page, and all specific functions are always named the same. These problems occur primarily with validator controls. Validator controls define a page scope variable (`Page_Validators`), a page scope function (`ValidatorOnSubmit()`), and also write some inline JavaScript code.
- Postback problems cause naming collisions in addition to posting back to the incorrect page: The preferred behavior for Web Control is generally an in-place refresh. For example, when a user selects the next month in the Calendar control, it posts back to the originating aspx page. Even though the post goes through the gateway, it is not sent to the portal page that hosted it, and the user loses the portal experience.

The .NET Web Control Consumer package overcomes these problems and provides a simplified experience for portlet developers. The core of the package is a .NET HttpModule that modifies the outgoing HTML before it reaches the portal server. The module solves the problems described above by: a) individuating all .NET provided function names to avoid collisions, and b) modifying all postback functions to use JavaScript instead of posting directly to the server.

## Configuring ALI Portlets in the Web Control Consumer

To configure an existing .NET Web application to be used in ALI, you must make a new entry in the web application's Web.config file.

Once you add an entry to the Web.config file, all gatewayed requests to the web application are transformed for use with AquaLogic Interaction. You can still access your web application directly; the module is bypassed for any request that does not originate from a portal server. For example templates, see the following folders in the installation directory: \deploy\Web.config and \deploy\web.config.node (node only). Your IIS Web Application Server must have direct Web access to the Plumtree Image Service.

**Note:** Any .NET portlet page that performs postbacks (including auto-postback) must be gatewayed. For details on configuring the gateway, see the AquaLogic Interaction Development Documentation or the portal online help.

- In the /configuration/system.web node of the Web.config file, add a new node that specifies the HttpModule class and containing assembly, as shown in the example below:

  ```
  <httpModules>
   <add type="Com.Plumtree.Remote.Loader.TransformerProxy,
  Aqualogic.WCLoader, Version=3.1.0.0, Culture=neutral,
  PublicKeyToken=d0e882dd51ca12c5" name="PTWCFilterHttpModule"
  />
  </httpModules>
  ```

## Configuring Image Server Access

If the URL used to access the portal Image Service internally is different from the URL used for external access, and the portlet server is hosted internally, you must configure the Image Service address.

Some network configurations require that the portal Image Service be accessed through a different URL internally and externally. This can be a problem for portlet servers that are hosted internally because AquaLogic Interaction always sends the external Image Service URL (the portlet must contact the Image Service to retrieve auxiliary JavaScript files). If this is the case, set up a mapping to allow the portlet server to determine the internal Image Service address. This can be achieved in two ways.

The simplest way is to map the external image server to the internal URL by following the steps below:

1. Open the imageserver.mapping.xml file in a text editor. This file is located in the .NET Web Controls installation folder, under \settings\config.
2. Add a new entry for the internal Image Service, as shown below:

   ```
   <mapping find="http://www.external-servername.com/ptimages/"
   replace="http://internal-servername:8080/ptimages/"/>
   ```

   You must include the whole URL; sections of the URL will not be replaced. To replace sections, you must use a regular expression mapper, as shown below:

   ```
   <mapping regex="true" find="www.external-(\w+).com"
   replace="internal-$1:8080" />
   ```

   (Both mappings do the same thing in this case. For details on syntax, see .NET regular expressions.)
3. Save the imageserver.mapping.xml file.
4. Restart IIS or re-save the config.xml file located in the same directory (this instructs the Web Controls to reload all configuration settings).

The second option is to set an alternative Image Service URL to override the external URL, using an Administrative preference. The default setting name is `PTWC.Mapping.Override` (this name can be changed in the HttpPipe.xml file). You must create an administrative preferences page to set the preference. This granularity of configuration is not generally necessary except for specific remote portlets that access the Image Service through a specific unique URL.

## Configuring .NET Framework Support for the Web Control Consumer

To add support for any .NET Framework version that uses JavaScript WebUIValidation, use the JSGenerator application.

The .NET Web Control Consumer is shipped with support for a few explicit versions of the .NET Framework. You can add support for any .NET Framework version that uses JavaScript.WebUIValidation (version 125) using the JSGenerator application. To generate a new version of JavaScript files, follow the steps below:

1. Start the JSGenerator application by running JSGenerator.exe located in the \bin directory of the WCC installation directory.
2. Select the new Framework version from the list, or enter the version in the format X.X.X.X. (The list includes all versions available on the local machine.)
3. Click Generate.
4. Copy the generated files. (Once generation is complete, the application will provide instructions on where to copy the new files to the Image Service.)

Once you have copied the files, the new version of the .NET Framework will be supported.

## Web Control Consumer Development Best Practices

The following tips and best practices should be considered for all portlets that use the Web Control Consumer.

Follow the basic guidelines below in any portlet that uses the Plumtree .NET Web Control Consumer:

- Name all elements uniquely.
- Do not assume your code is in a particular location in the HTML DOM.
- Do not rely on back/forward buttons or JavaScript commands.
- Do not access validator spans directly.
- Use FlowLayout instead of GridLayout.

- Always include the target page in the gateway space.

For more information and additional best practices, see the sections that follow.

### Using PostBacks

The .NET Web Control Consumer module enables the use of .NET Web Controls and their associated AutoPostBack function. All automatic postbacks are caught and used to repaint the individual portlet. The module makes a number of modifications to the HTML; below is an overview of the modifications you should be aware of:

- All POSTs are handled by JavaScript, so they do not become part of the browser history. This means that JavaScript functions will not work; the Back and Forward buttons of the browser will skip any clicks made on Web Controls that are handled in this way.
- All requests from a portal server are transformed; navigating to the portlet directly does not yield the same HTML that is returned to the portal server. This allows the ASPX page to be accessed directly if required (the modified page would not function outside the portal environment). To examine the HTML going to the portal server, use a trace tool between the portal and the portlet server to intercept the HTML.
- All client-side validation script is transformed, but since ASP.NET only generates client-side script for specific versions of Internet Explorer (IE), nothing is altered for browsers other than IE.
- All .NET portlets are wrapped in an HTML `<span>`. Do not assume that the portlet is at any specific point in the HTML structure of the page; portlet parent elements are usually HTML table cells, but not in this case.
- The `_doPostBack` method is removed (a similar function is included in an additional JavaScript file). Beware of hooking into this .NET-provided function directly; both the function name and argument list are modified. Any direct calls to this function should also be modified automatically, however any non-standard calling conventions could potentially not be recognized by the module. To programmatically perform a postback from JavaScript, make the call `_doPostBack('','');`.
- Always use FlowLayout because GridLayout uses absolute positioning.
- The names for HTML spans (for validators) and forms (required for all pages that perform a postback) are appended with a unique ID (portlet ID), to avoid naming collisions on a portal page. Do not refer to these names directly in JavaScript. (If it is absolutely required, you can assume that they will be named <original-name>_<portlet-id>.)

### Using Unique Naming

Generally all HTML elements should be named uniquely, which normally involves appending the portlet ID to the element name. This can be problematic with ASP.NET, because the Web Control names and IDs cannot be generated at runtime in the normal ASPX syntax. For example the following code will not work:

```
<asp:Button id="Execute_<%= portlet_id %>" runat="server"
Text="Test"></asp:Button>
```

The best way to append the portlet ID to the name is programmatically. To do this, declare the control in the ASPX page with a standard name, and then modify the ID property from the code behind page. You can access the actual controls using the `System.Web.UI.Page Controls` property or the `System.Web.UI.Page FindControl(string ControlName)` method.

For example, to append the portlet ID to a control named "MyButton," the following code could be employed:

```
protected override void OnPreRender(EventArgs e)
{
 base.OnPreRender (e);
 FindControl("MyControl").ID += portletID;
}
```

**Note:** It is simplest to append the ID using the page's PreRender event so that the standard ID is used until the last possible point.

### Using Submit Buttons

In most cases, submit buttons will be modified to remain in the portal context. This was not the case in the 2.1.x release; in that release, they required an additional `ptrender` attribute. This now defaults to true, although it can be disabled by adding a `ptrender=false` attribute to the button.

**Note:** As in the 2.0.x release, it is still possible to place a `ptrender` tag on the form itself. This means that anything attempting to submit the form will cause it to be posted back in-page. However, we recommend using the tag on the buttons, unless it is specifically required (for example, a third-party control is attempting to submit the form directly).

The standard method of posting to the target page and redirecting back to the portal page using the IDK is also possible, however this is generally slower (performs a post and a redirect), less aesthetically pleasing (refreshes the whole page), and will lose the state of the page if handled by the client (__VIEWSTATE will be lost).

### Using Custom JavaScript

Each page refresh dynamically writes the HTML to re-render the portlet, so custom JavaScript can pose a problem. Because any JavaScript is now executing as the portlet is rendering, there are some functions that will not work correctly. Any `document.write(..)` calls will not work. To call a script each time the page is refreshed (or every time the portlet is re-rendered), use the rerenderPortletID event in the ALI Scripting Framework. This event is available only to .NET portlets and is named using the current portlet ID. For example, to create a JavaScript alert each time the portlet is rendered, use the code below.

**Note:** JavaScript is executed for each postback; make sure that the script to register for the event is only included in the first page; otherwise the function will be registered multiple times.

```
<pt:namespace pt:token="$$PORTLET_ID$$"
xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'/>
function alertMe_$$PORTLET_ID$$, () {
      alert("Portlet Rendering!");
}
alertMe_$$PORTLET_ID$$;//call the first time the portlet is loaded
```

Then register for the event in the code behind page:

```
private void Page_Load(object sender, System.EventArgs e)
{
 if(!IsPostBack)
  RegisterClientScriptBlock("rerender", "<script
language=\"javascript\">document.PCC.RegisterForWindowEvent('rerender.$$PORTLET_ID$$',
 alertMe_$$PORTLET_ID$$)</script>");
}
```

In the example above, $$PORTLET_ID$$ is replaced by the portlet ID using the `pt:token` tag. You can also use the IDK to extract the portlet ID from the request. See the AquaLogic Interaction Developer Documentation for more information on adaptive tags.

### Disabling Filters

Under some circumstances, it may be desirable to disable HTML filtering. Pages opening in popup windows or operating in gatewayed mode will probably not want any HTML modification. Filtering can be disabled per-request by making the following call:

```
Context.Items["PTWC:EnableFilter"] = false;
```

This will disable filtering for the current request only.

## Web Control Consumer Frequently Asked Questions

The following questions address common issues in portlet development using the Web Control Consumer (WCC).

1. **Where are the WCC web controls in VS.NET?** The WCC does not provide any additional ASP.NET controls; it offers support for the existing ASP.NET controls available with VS.NET to operate as a portlet within a portal enviroment.
2. **Which dlls do I need to add as references to my project in Visual Studio?** None. All the assembly loading is handled automatically when the HttpModule line is included in your Web.config file.
3. **Do I need to use the `ReturnToPortal` call to go back to the portal page?** Usually, no. If your portlet performs all its logic on the main portal page, then you will never leave the page, so you don't need to return to it. The exception is if you use gatewayed or hosted mode, in which case you can use this call to return to the aggregated page.
4. **When do I need to include the `GetPostBackEventReference(this)` call?** This call is no longer required (version 2.2 and above).
5. **Why was the stylesheet link replaced with some JavaScript?** The stylesheet must be appended to the main HTML DOM programmatically, otherwise it will be unloaded upon postback and omitted upon refresh. Instead of including your stylesheets in the standard way (`<link type="text/css" rel="stylesheet" href="http://portal-img.plumtree.com/ptimages/plumtree/common/public/css/mainstyle-en.css"/>`), the filter rewrites the link and appends it to the page using JavaScript.
6. **How do I call a custom JavaScript function as soon as the portlet refreshes itself?** Call the function from an inline piece of script.
7. **How do I run a custom JavaScript function as soon as another portlet refreshes itself?** Register the function for the portlet's rerender event. This can be done with the following JavaScript call, where $$PORTLET_ID$$ is the ID for the portlet for which to listen: `document.PCC.RegisterForEvent(document.PCC.WindowEventURN, "rerender.$$PORTLET_ID$$", myCustomFunction);`
8. **How do I make my portlet refresh periodically to keep its data up to date?** Call the postback function from a timeout. For example, the code below specifies a 10msec timeout: `<script language="javascript">setTimeout(10, "__doPostBack('','')");<script>`

9. **Why can't I upload files from a form?** File upload cannot be done within the page, so in-page refresh is disabled for any multi-part forms. For these forms, you must redirect back to the aggregated page manually or perform the upload in a popup window.
10. **How do I disable the filter on a specific page?** Add the following call to your code to disable the filter for the current request: `Context.Items["PTWC:EnableFilter"] = false;`