

# I Prepping for Obsidian

What you can do to get ready for the  
new UI framework





# Daniel Hazelbaker

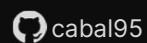
---



Software Developer

Working at Triumph Tech

20 Years Experience at Churches





# What is Obsidian

---

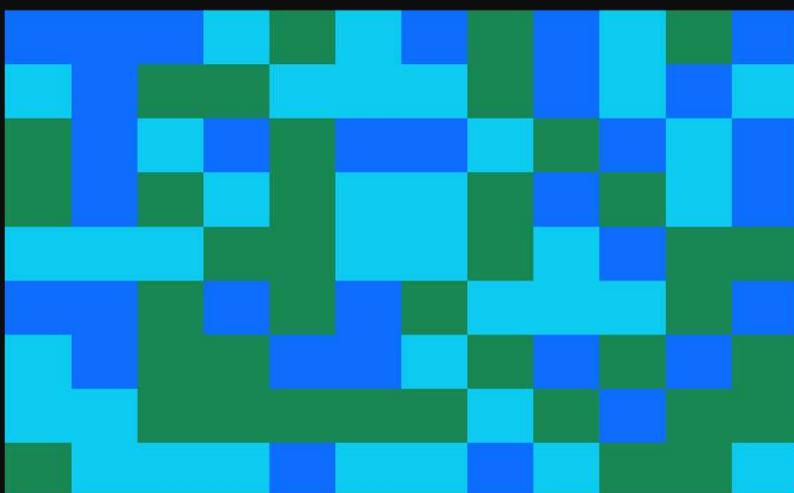


= + +



# WebForms vs Obsidian

WebForms Block



UI / Layout

Obsidian Block



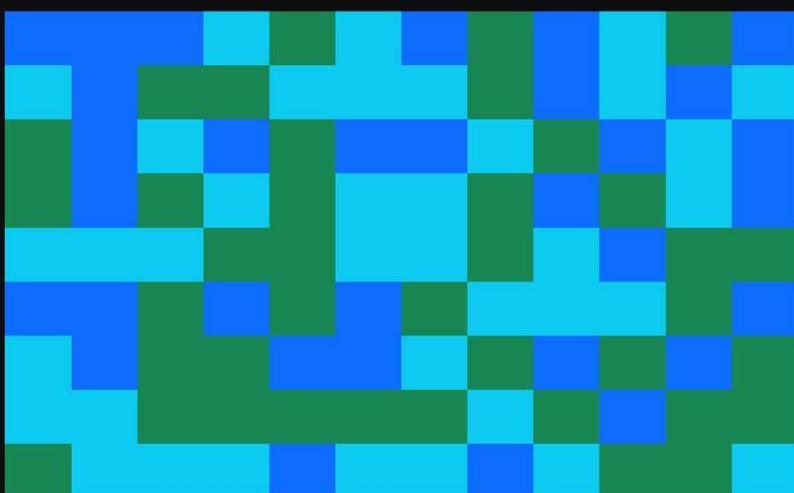
Logic

Service



# WebForms vs Obsidian

WebForms Block



UI / Layout

Logic

Service

Obsidian Block

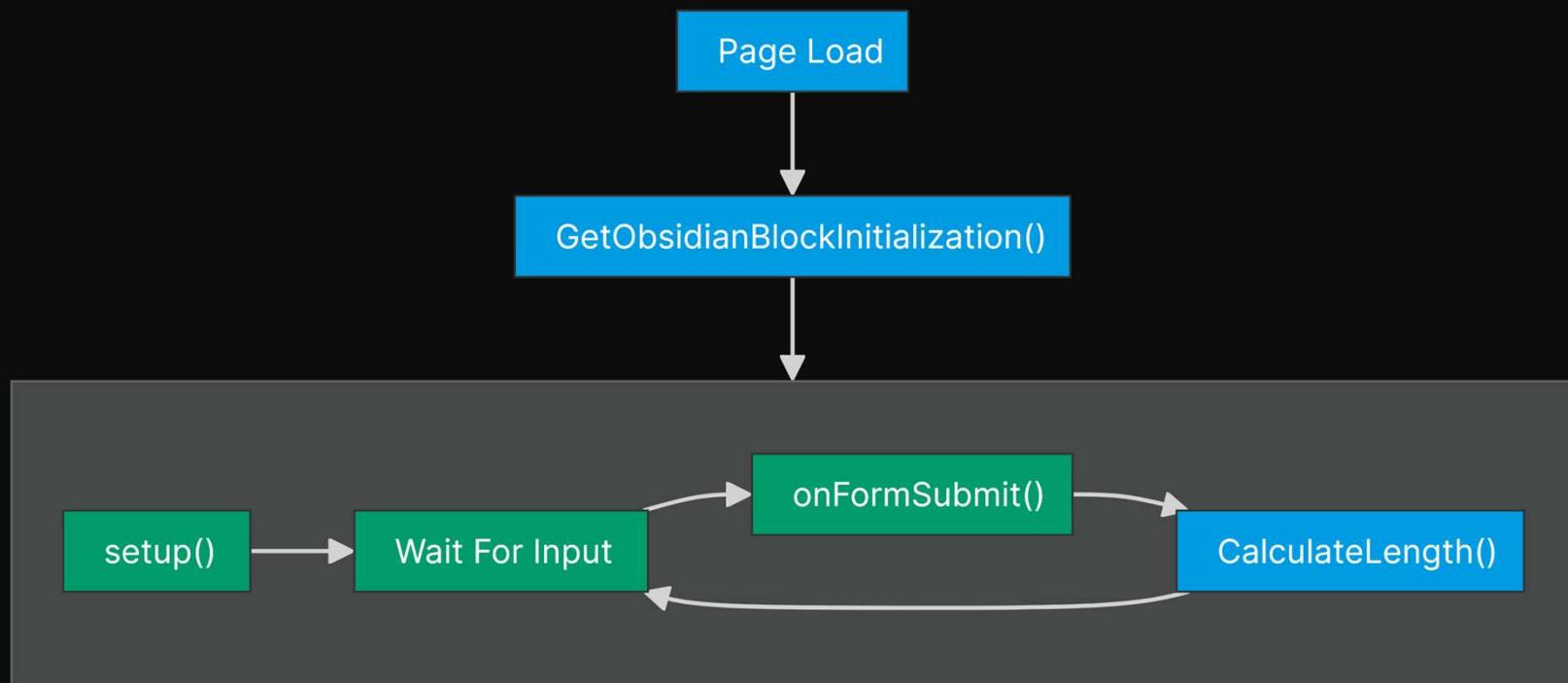
campusDetail.obs

CampusDetail.cs

CampusService.cs



# How Obsidian Blocks Work

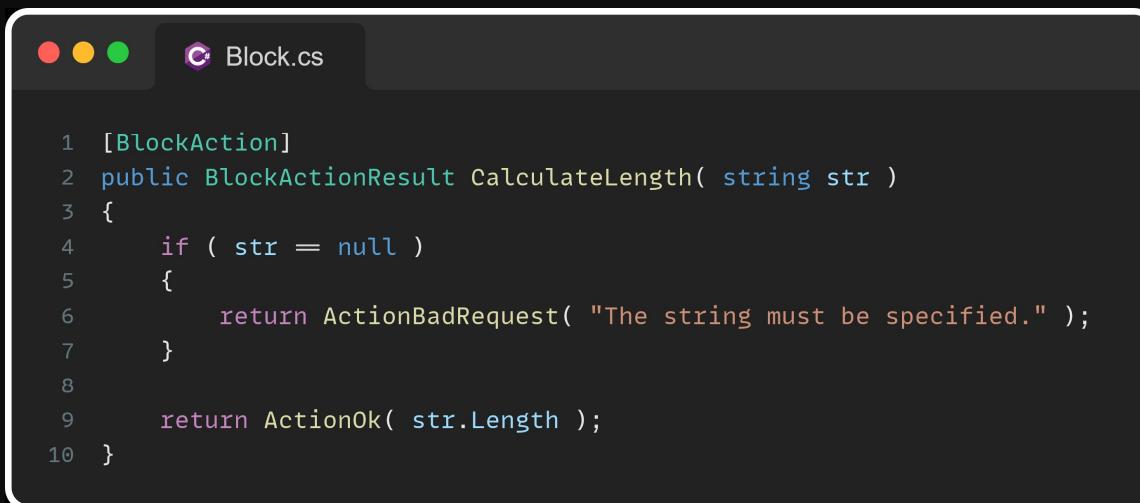




# Block Actions

A call to block action CalculateLength becomes a URL request.

```
POST https://www.rockrms.com/api/v2/<pageguid>/<blockguid>/CalculateLength
```



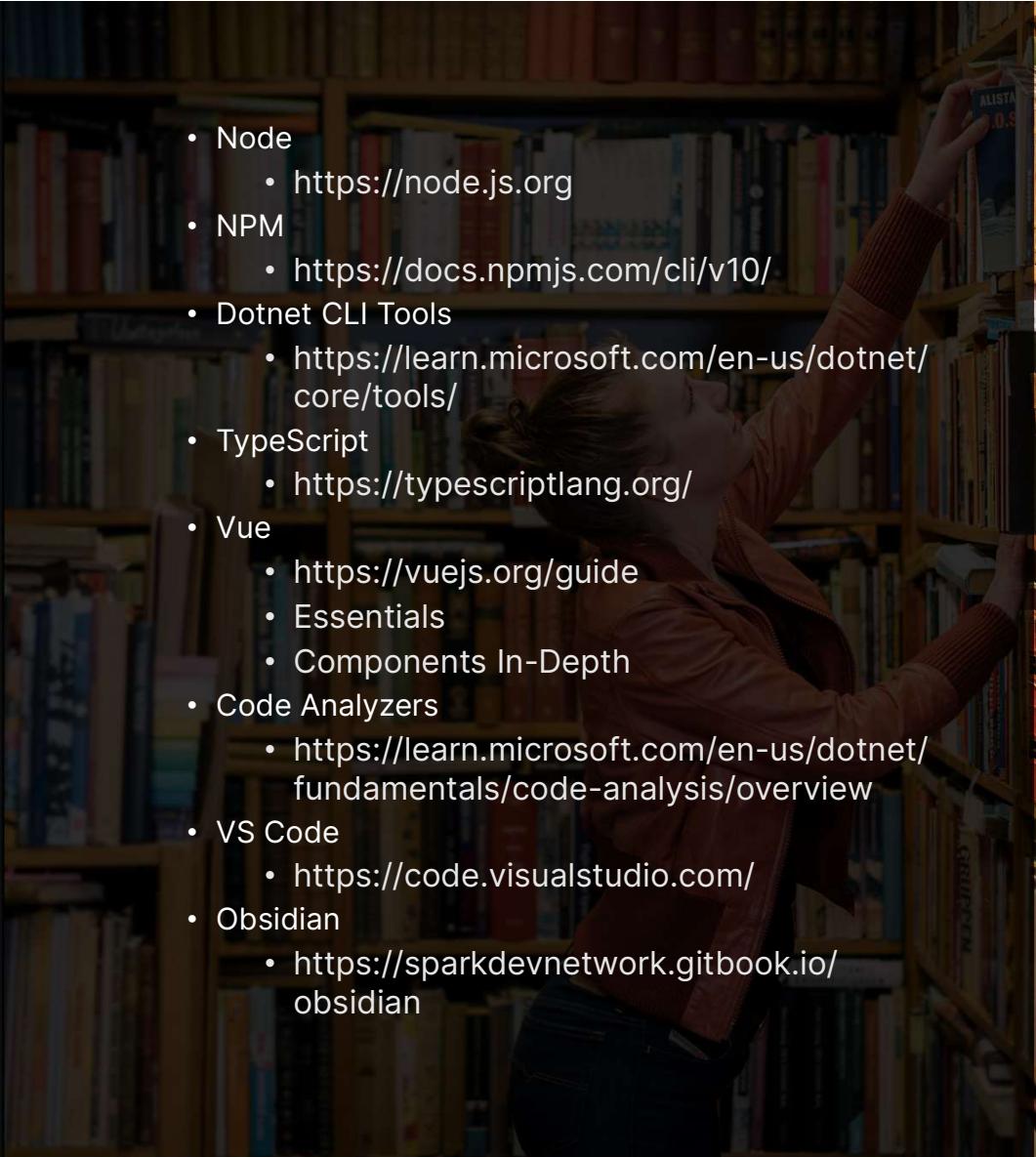
The image shows a screenshot of a Mac OS X desktop environment. A window titled "Block.cs" is open, displaying a C# code listing. The code defines a class with a single method, CalculateLength, which takes a string parameter and returns its length. The code includes basic validation for null input.

```
1 [BlockAction]
2 public BlockActionResult CalculateLength( string str )
3 {
4     if ( str == null )
5     {
6         return ActionBadRequest( "The string must be specified." );
7     }
8
9     return ActionOk( str.Length );
10 }
```



# What Can You Learn Now

- Node
  - <https://nodejs.org>
- NPM
  - <https://docs.npmjs.com/cli/v10/>
- Dotnet CLI Tools
  - <https://learn.microsoft.com/en-us/dotnet/core/tools/>
- TypeScript
  - <https://typescriptlang.org/>
- Vue
  - <https://vuejs.org/guide>
  - Essentials
  - Components In-Depth
- Code Analyzers
  - <https://learn.microsoft.com/en-us/dotnet/fundamentals/code-analysis/overview>
- VS Code
  - <https://code.visualstudio.com/>
- Obsidian
  - <https://sparkdevnetwork.gitbook.io/obsidian>

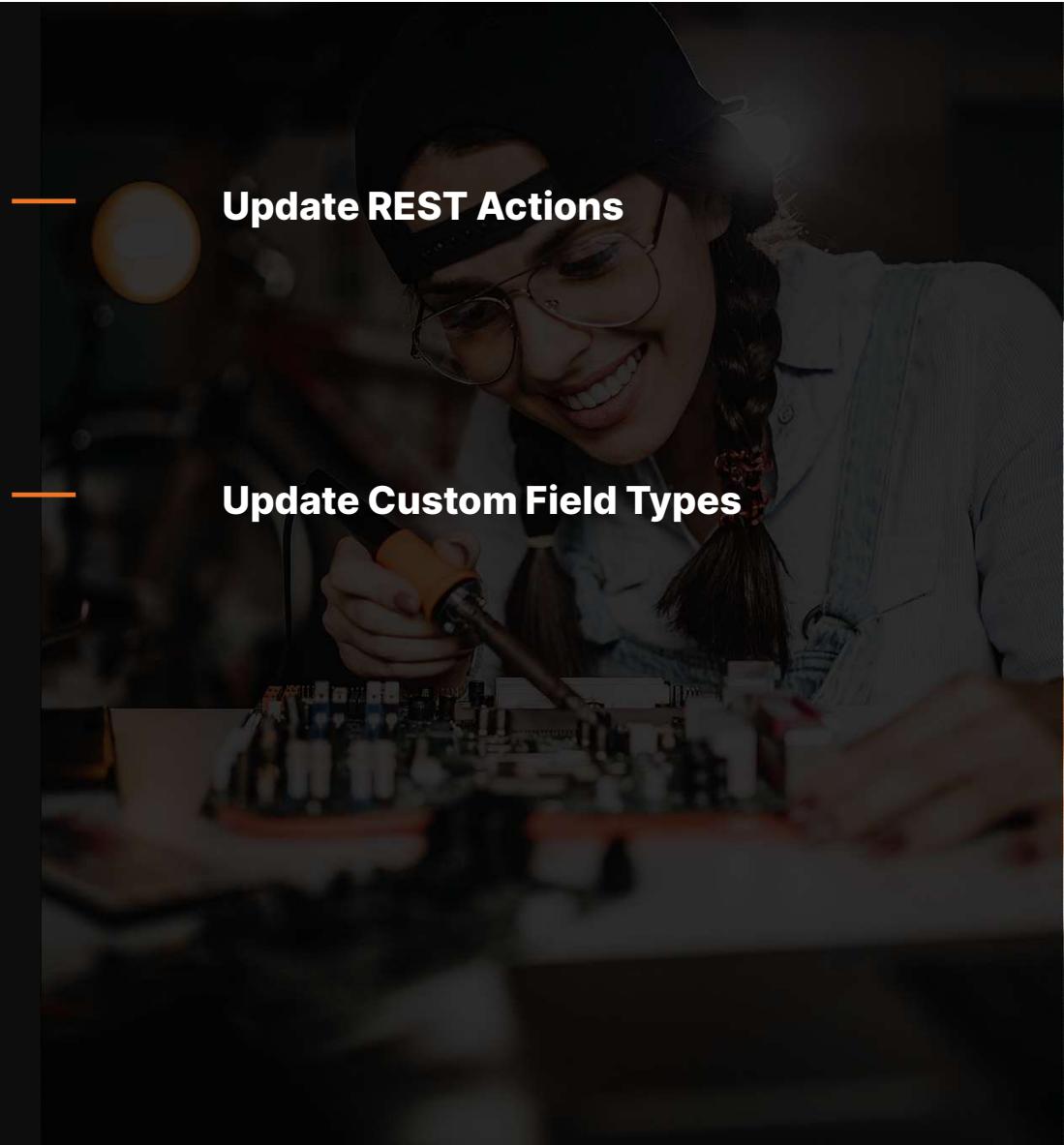




# What Can Do Now

— **Update REST Actions**

— **Update Custom Field Types**





# REST Actions

A screenshot of a code editor window titled "PluginController.cs". The code is written in C# and defines a class named "PluginController" that inherits from "ApiController". The class contains a single method named "Sum" which takes two integer parameters "a" and "b", and returns their sum. The code uses attribute-based routing and action selection via GUIDs.

```
1 [RestControllerGuid( "03edaa8b-f746-4bfc-8ee9-d828bbf9edcc" )]
2 public class PluginController : ApiController
3 {
4     [HttpGet]
5     [Route( "sum" )]
6     [RestActionGuid( "36acb31d-6251-4f76-809c-38986b1ab0e4" )]
7     public int Sum( int a, int b )
8     {
9         return a + b;
10    }
11 }
```



# REST Actions



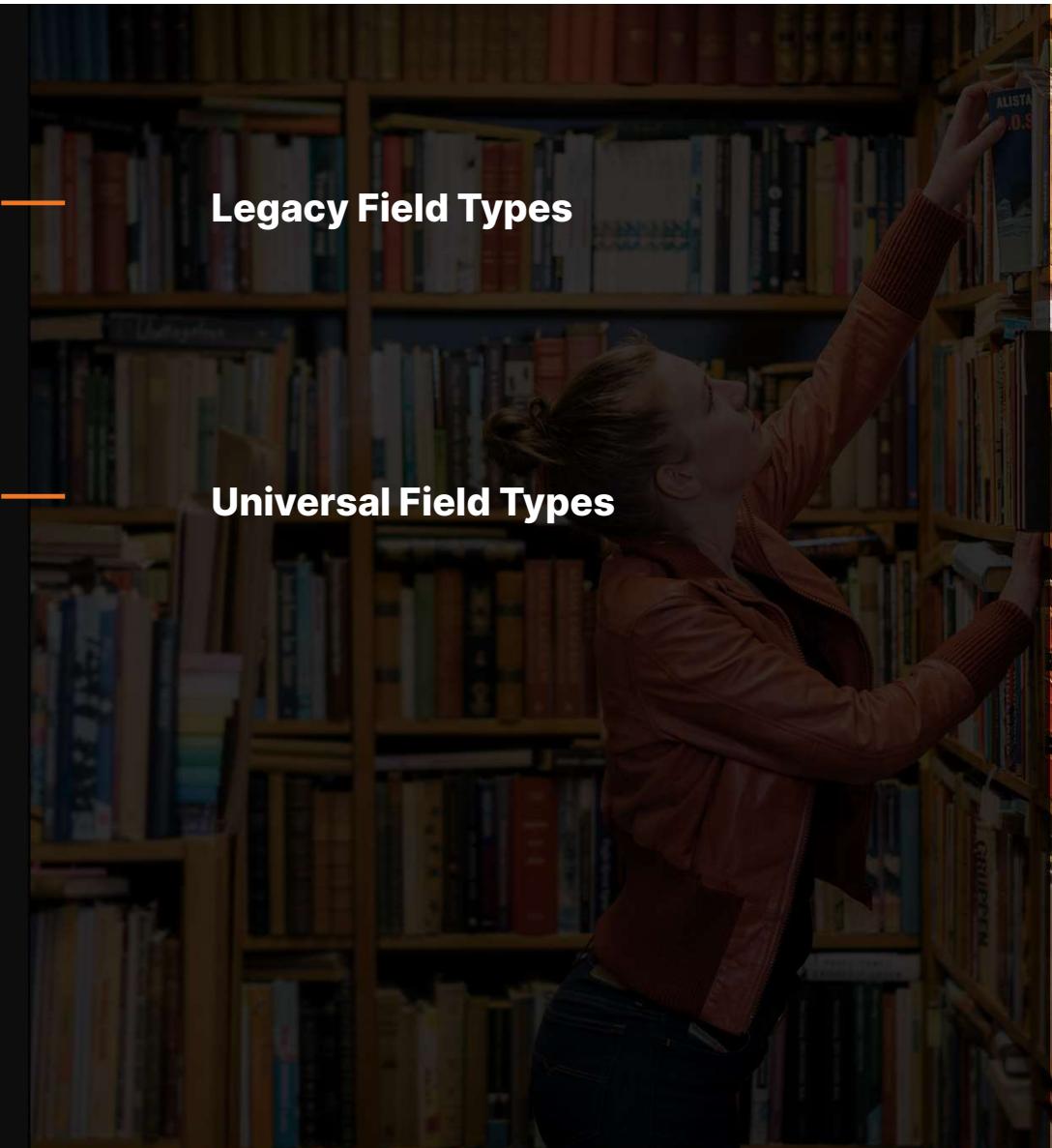
```
1 [RestControllerGuid( "03edaa8b-f746-4bfc-8ee9-d828bbf9edcc" )]
2 public class PluginController : ApiController
3 {
4     [HttpGet]
5     [HttpPost]
6     [Route( "sum" )]
7     [RestActionGuid( "36acb31d-6251-4f76-809c-38986b1ab0e4" )]
8     public int Sum( int a, int b )
9     {
10         return a + b;
11     }
12 }
```



# Universal Field Types

—  
**Legacy Field Types**

—  
**Universal Field Types**





# Field Types

---

Sample

Red

Green

Blue

Sample

Red

Green

Blue

Sample

RedGreenBlue▼

Red

Green

Blue

Sample

Red ×Green ×▼

Red ✓

Green ✓

Blue



# Field Types

Sample

A screenshot of a dropdown menu titled "Sample". It contains three items: "Red" (with a red key icon), "Green" (with a green key icon), and "Blue" (with a blue key icon). The menu has a standard Windows-style appearance with a dropdown arrow icon.

Select Cancel

Sample

A screenshot of a detailed view window titled "Sample". It shows a single item: "Main Campus". To the right of the item name is a blue rectangular button labeled "Physical". Below the item name is a descriptive text: "This is the main campus for the organization." Underneath that is the status "Status Open". At the bottom of the window are "Select" and "Cancel" buttons.

Physical

Main Campus

This is the main campus for the organization.

Status Open

Select Cancel



# Item Field Type

```
 1 [FieldTypeGuid( "30EE533F-C11E-4846-AA79-1FCD96B37ACE" )]
 2 public class TestFieldType : UniversalItemPickerFieldType
 3 {
 4     protected override bool IsMultipleSelection => false;
 5
 6     protected override UniversalItemValuePickerDisplayStyle GetDisplayStyle(
 7         Dictionary<string, string> privateConfigurationValues );
 8
 9     protected override List<ListItemBag> GetListItems(
10         Dictionary<string, string> privateConfigurationValues );
11
12     protected override List<ListItemBag> GetItemBags(
13         IEnumerable<string> values,
14         Dictionary<string, string> privateConfigurationValues );
15 }
```

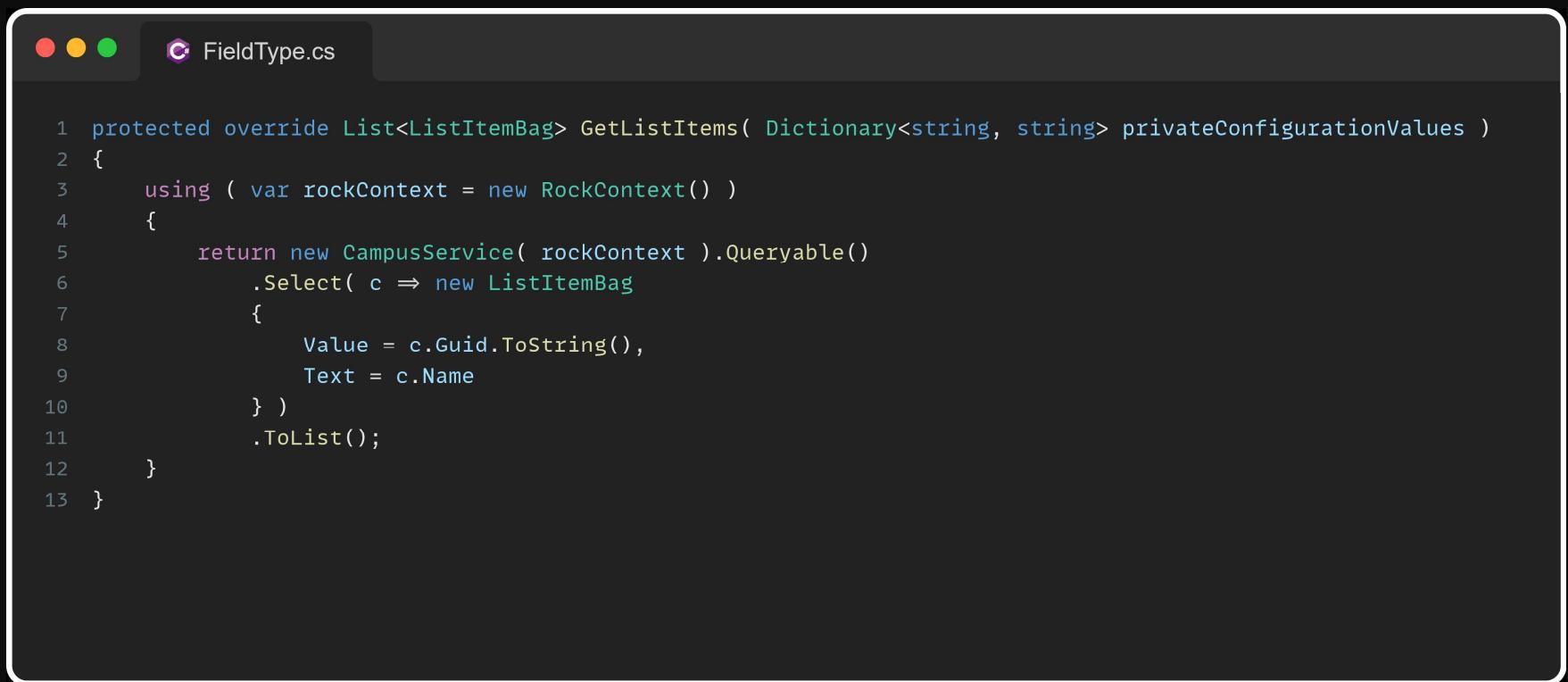


# Item Field Type

```
1 protected override List<ListItemBag> GetItemBags( IEnumerable<string> values,
2     Dictionary<string, string> privateConfigurationValues )
3 {
4     using ( var rockContext = new RockContext() )
5     {
6         return new CampusService( rockContext ).Queryable()
7             .Where( c => values.Contains( c.Guid.ToString() ) )
8             .Select( c => new ListItemBag
9             {
10                 Value = c.Guid.ToString(),
11                 Text = c.Name
12             } )
13             .ToList();
14     }
15 }
```



# Item Field Type



```
1 protected override List<ListItemBag> GetListItems( Dictionary<string, string> privateConfigurationValues )
2 {
3     using ( var rockContext = new RockContext() )
4     {
5         return new CampusService( rockContext ).Queryable()
6             .Select( c => new ListItemBag
7             {
8                 Value = c.Guid.ToString(),
9                 Text = c.Name
10            } )
11            .ToList();
12    }
13 }
```



# Item Field Type

---

Sample

Red

Green

Blue

Sample

Red

Green

Blue

Sample

RedGreenBlue▼

Red

Green

Blue

Sample

Red ×Green ×▼

Red ✓

Green ✓

Blue



# Search Field Type

---

```
 1 [FieldTypeGuid( "7F6A5898-4484-4021-A9BB-BF20A130B08B" )]
 2 public class TestFieldType : UniversalItemSearchPickerFieldType
 3 {
 4     protected override List<ListItemBag> GetItemBags( IEnumerable<string> values,
 5             Dictionary<string, string> privateConfigurationValues );
 6
 7     protected override string GetSearchUrl( Dictionary<string, string> privateConfigurationValues );
 8
 9     protected override bool AreDetailsAlwaysVisible( Dictionary<string, string> privateConfigurationValues );
10
11     protected override string GetItemIconCssClass( Dictionary<string, string> privateConfigurationValues );
12
13     protected override bool IsIncludeInactiveVisible( Dictionary<string, string> privateConfigurationValues );
14 }
```



# Search Field Type

---

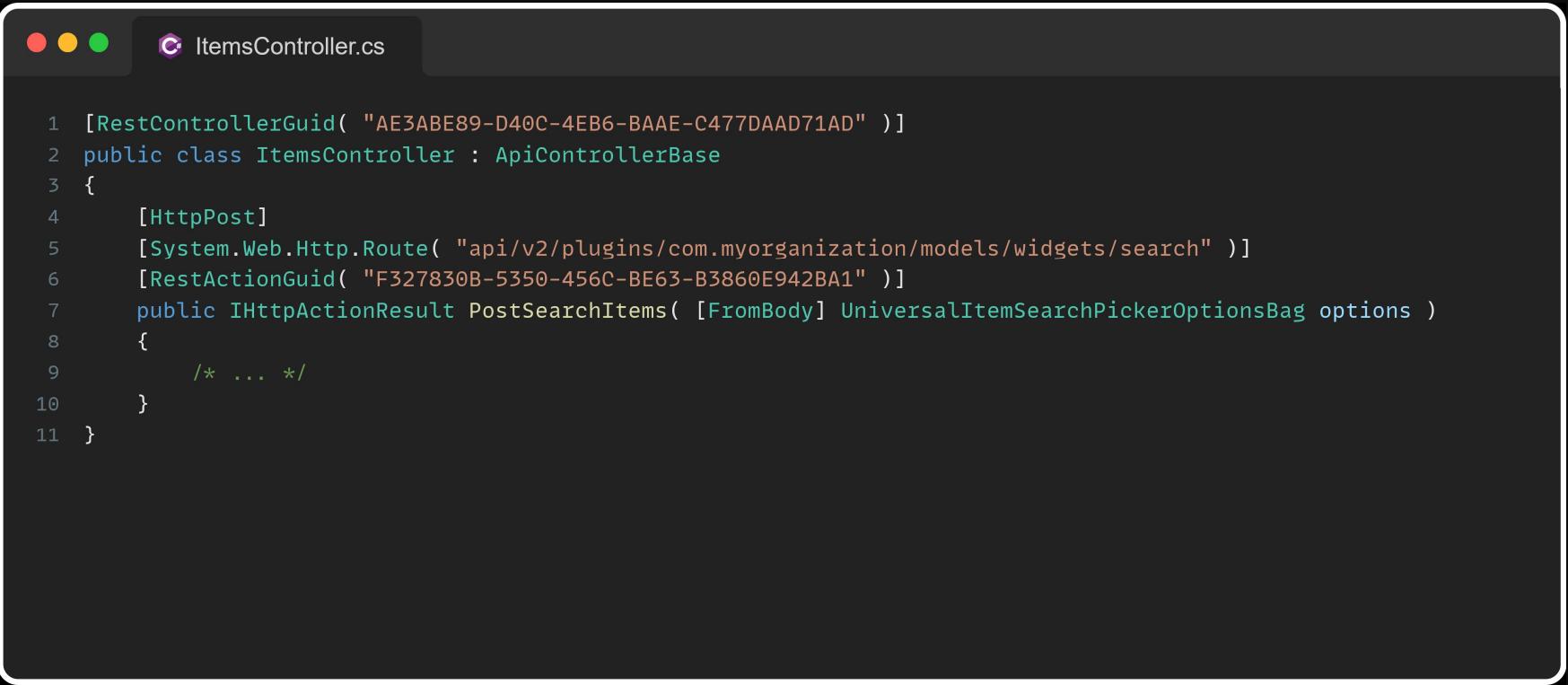


```
1 protected override string GetSearchUrl( Dictionary<string, string> privateConfigurationValues )
2 {
3     return "/api/v2/plugins/com.myorganization/models/widgets/search";
4 }
```



# Search Field Type

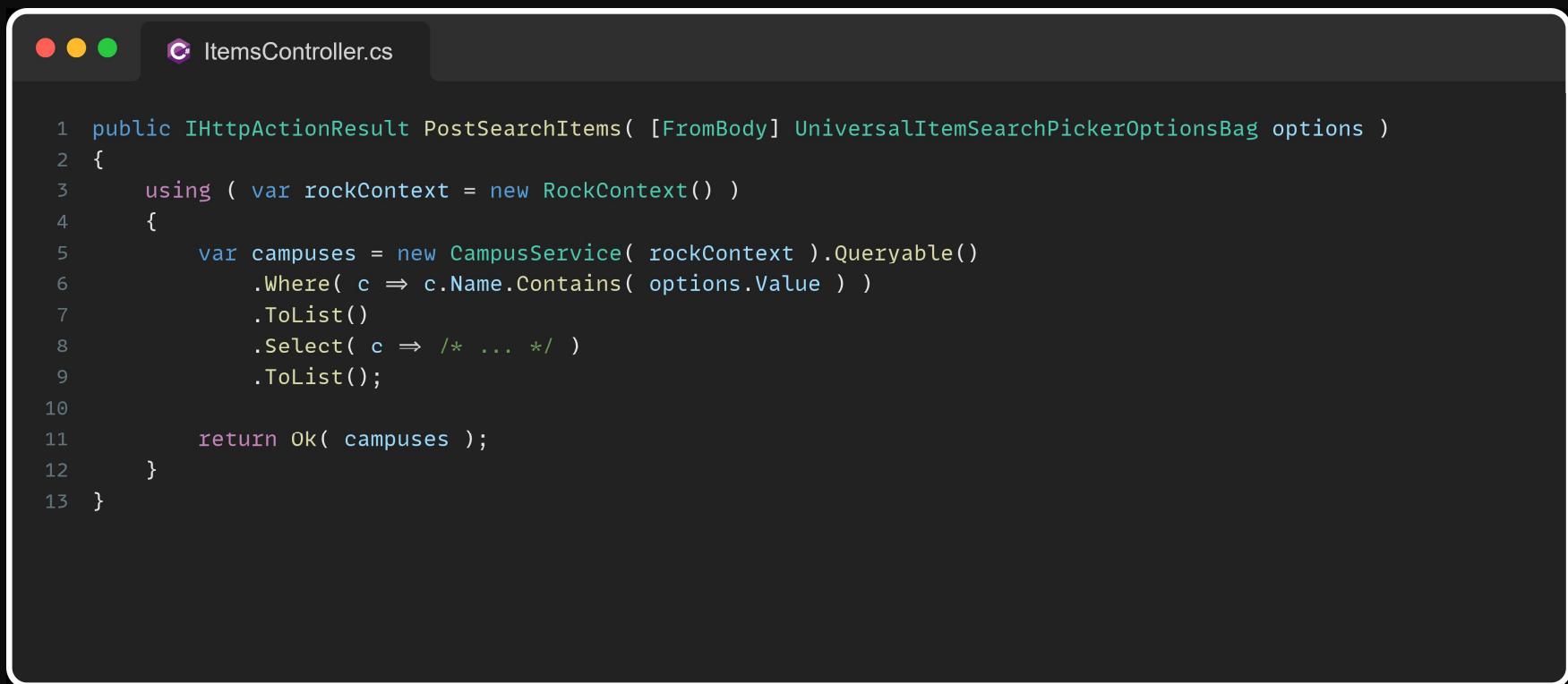
---



```
1 [RestControllerGuid( "AE3ABE89-D40C-4EB6-BAAE-C477DAAD71AD" )]
2 public class ItemsController : ApiControllerBase
3 {
4     [HttpPost]
5     [System.Web.Http.Route( "api/v2/plugins/com.myorganization/models/widgets/search" )]
6     [RestActionGuid( "F327830B-5350-456C-BE63-B3860E942BA1" )]
7     public IHttpActionResult PostSearchItems( [FromBody] UniversalItemSearchPickerOptionsBag options )
8     {
9         /* ... */
10    }
11 }
```



# Search Field Type



A screenshot of a code editor window titled "ItemsController.cs". The window has a dark theme with orange window controls. The code is written in C# and defines a controller action named "PostSearchItems". The action takes a parameter of type `IHttpActionResult` and `[FromBody] UniversalItemSearchPickerOptionsBag options`. Inside the action, it uses a `RockContext` to query a `CampusService`, filtering by the `Name` containing the search term from the options. The result is returned as an `Ok` response.

```
1 public IHttpActionResult PostSearchItems( [FromBody] UniversalItemSearchPickerOptionsBag options )
2 {
3     using ( var rockContext = new RockContext() )
4     {
5         var campuses = new CampusService( rockContext ).Queryable()
6             .Where( c => c.Name.Contains( options.Value ) )
7             .ToList()
8             .Select( c => /* ... */ )
9             .ToList();
10
11         return Ok( campuses );
12     }
13 }
```



# Search Field Type

---

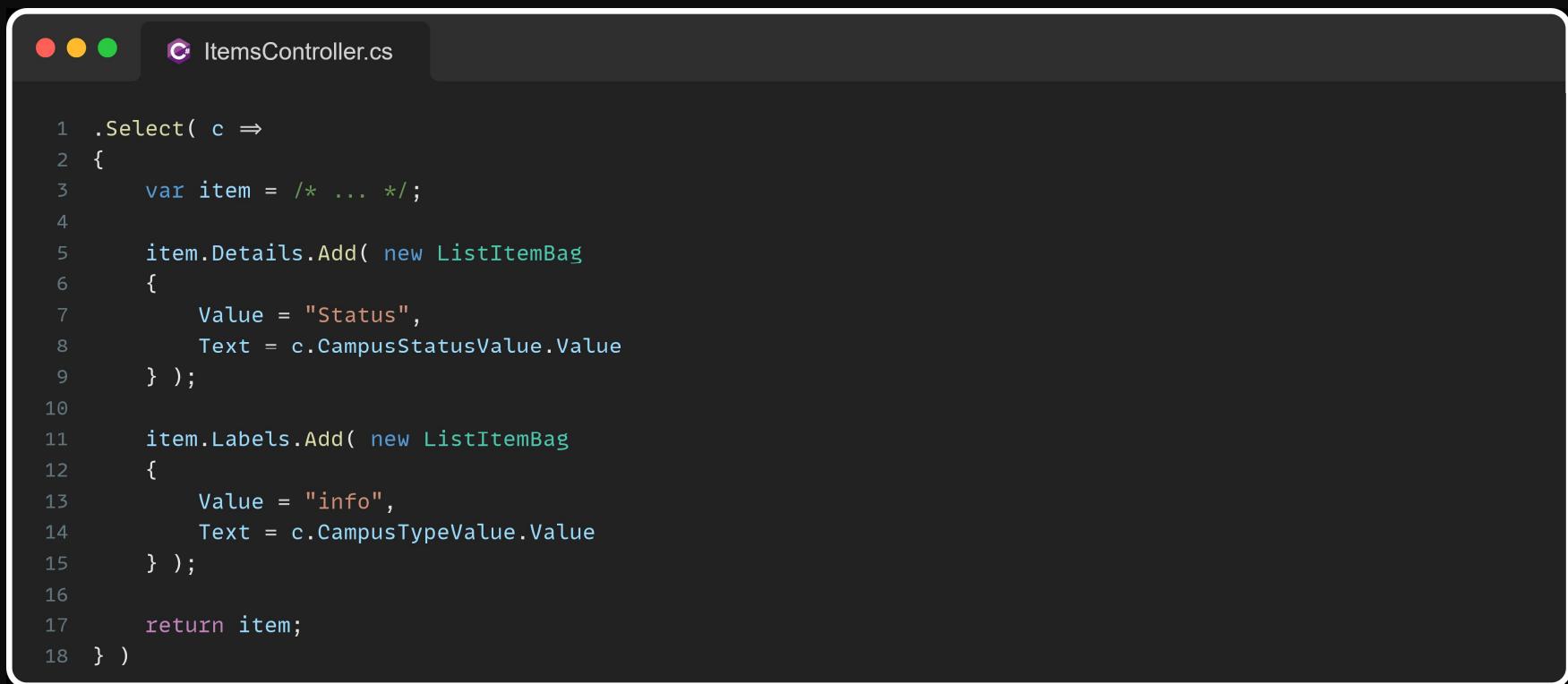


A screenshot of a macOS application window titled "ItemsController.cs". The window contains the following C# code:

```
1 .Select( c =>
2 {
3     var item = new UniversalItemSearchPickerItemBag
4     {
5         Value = c.Guid.ToString(),
6         Title = c.Name,
7         Description = c.Description,
8         Details = new List<ListItemBag>(),
9         Labels = new List<ListItemBag>()
10    );
11
12    return item;
13 } )
```



# Search Field Type



A screenshot of a code editor window titled "ItemsController.cs". The window has a dark theme with orange window controls. The code is written in C# and defines a method that returns a selected item with details and labels added.

```
1 .Select( c =>
2 {
3     var item = /* ... */;
4
5     item.Details.Add( new ListItemBag
6     {
7         Value = "Status",
8         Text = c.CampusStatusValue.Value
9     });
10
11    item.Labels.Add( new ListItemBag
12    {
13        Value = "info",
14        Text = c.CampusTypeValue.Value
15    });
16
17    return item;
18 } )
```



# Search Field Type

---

Sample

Search

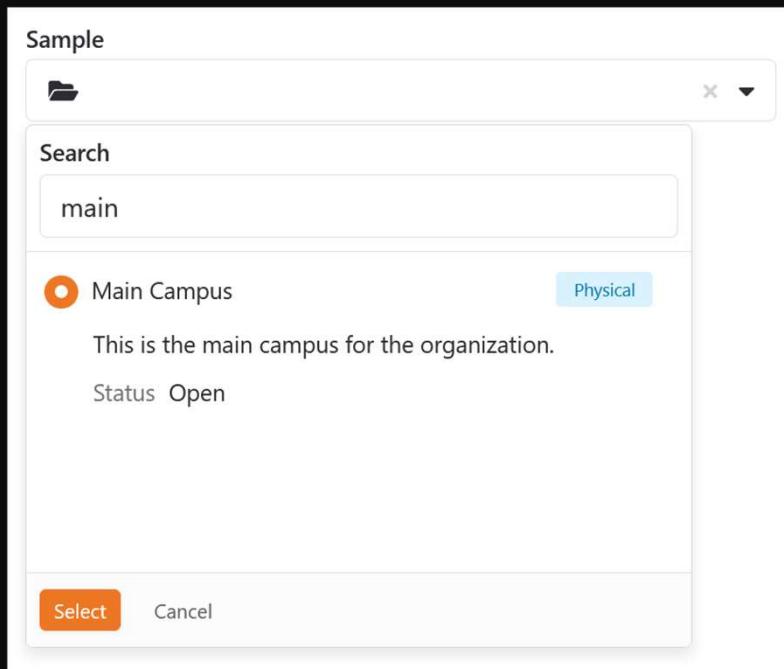
main

Main Campus      Physical

This is the main campus for the organization.

Status Open

Select      Cancel





# Tree Field Type

```
 1 [FieldTypeGuid( "30EE533F-C11E-4846-AA79-1FCD96B37ACE" )]
 2 public class TestFieldType : UniversalItemTreePickerFieldType
 3 {
 4     protected override bool IsMultipleSelection => false;
 5
 6     protected override List<ListItemBag> GetItemBags( IEnumerable<string> values,
 7             Dictionary<string, string> privateConfigurationValues );
 8
 9     protected override string GetRootRestUrl( Dictionary<string, string> privateConfigurationValues );
10 }
```



# Tree Field Type

---



```
1 protected override string GetRootRestUrl( Dictionary<string, string> privateConfigurationValues )
2 {
3     return "/api/v2/plugins/com.myorganization/models/widgets/tree";
4 }
```



# Tree Field Type

```
 1 [HttpPost]
 2 [System.Web.Http.Route( "api/v2/plugins/com.myorganization/models/widgets/tree" )]
 3 [RestActionGuid( "0CE18321-E833-47AD-AA16-2E949C5546E8" )]
 4 public IHttpActionResult PostTreeItems( [FromBody] UniversalItemTreePickerOptionsBag options )
 5 {
 6     using ( var rockContext = new RockContext() )
 7     {
 8         var locationService = new LocationService( rockContext );
 9         var expandGuids = GetExpandGuids( locationService,
10             options.ExpandToValues?.AsGuidList() );
11         var locations = LoadLocations( locationService,
12             options.ParentValue.AsGuidOrNull(),
13             expandGuids );
14
15         return Ok( locations );
16     }
17 }
```



# Tree Field Type

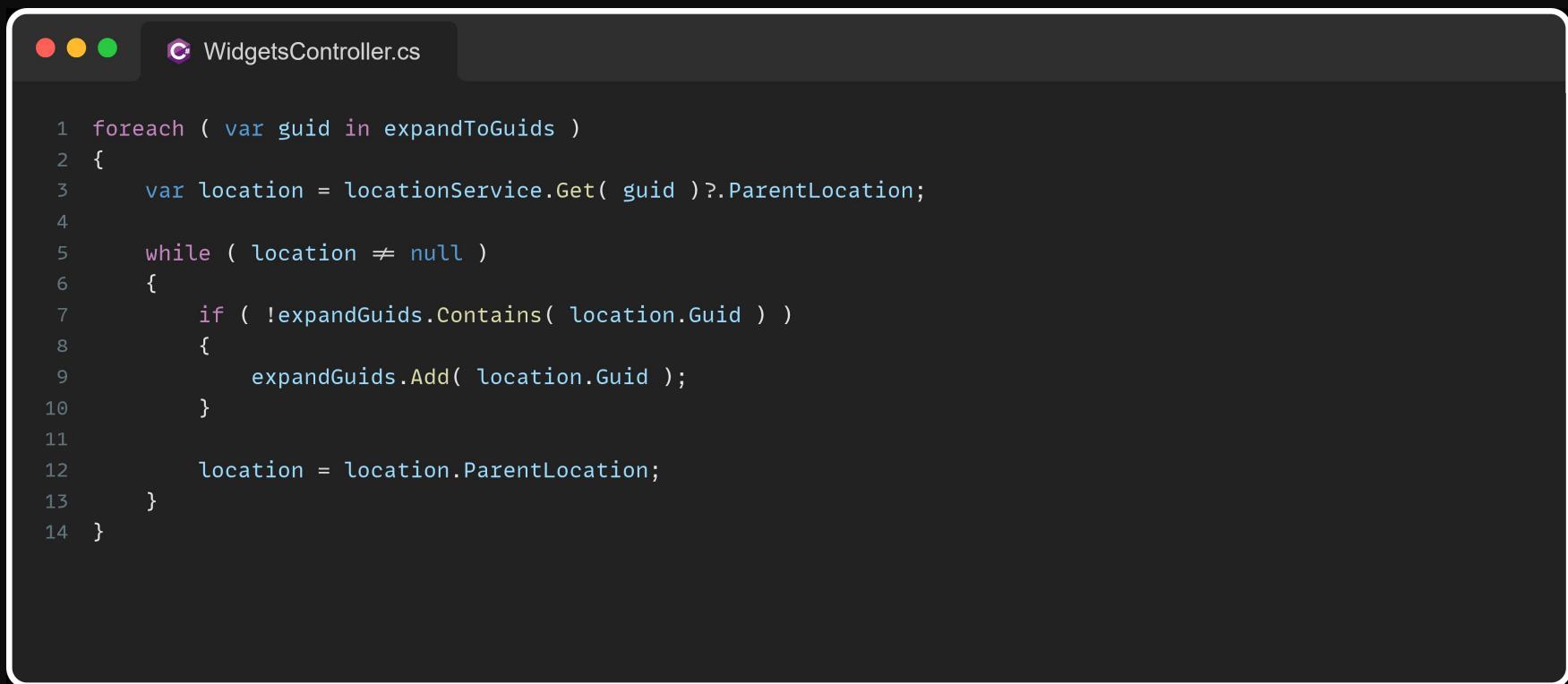


A screenshot of a code editor window titled "WidgetsController.cs". The window has a dark theme with orange window controls. The code shown is C# and defines a private method "GetExpandGuids". The code uses nullable reference types and LINQ. It initializes a list of GUIDs, checks if the input list is null, and then iterates through each GUID in the input list to add it to the output list.

```
1 private List<Guid> GetExpandGuids( LocationService locationService, List<Guid> expandToGuids )
2 {
3     var expandGuids = new List<Guid>();
4
5     if ( expandToGuids == null )
6     {
7         return expandGuids;
8     }
9
10    foreach ( var guid in expandToGuids )
11    {
12    }
13
14    return expandGuids;
15 }
```



# Tree Field Type

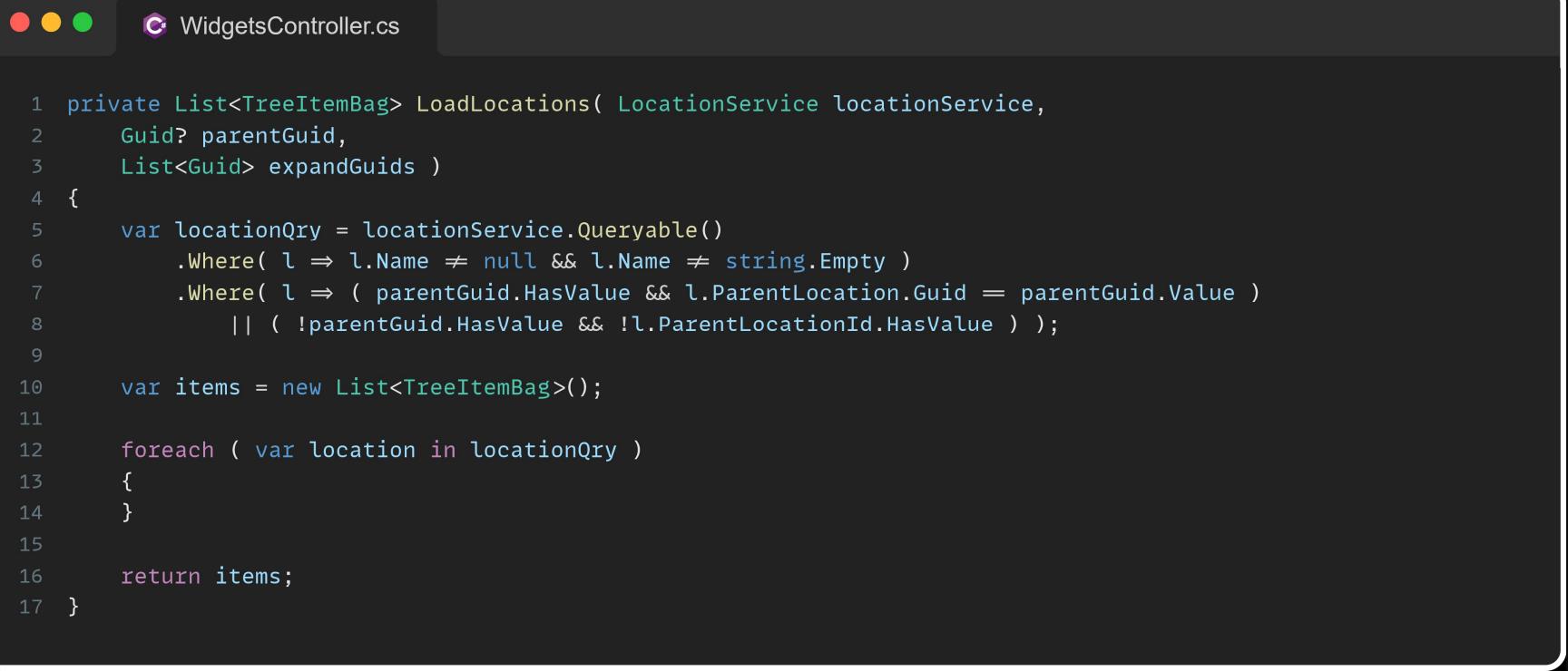


A screenshot of a code editor window titled "WidgetsController.cs". The code is written in C# and performs a recursive search to find all parent locations for a given list of GUIDs.

```
1 foreach ( var guid in expandToGuids )
2 {
3     var location = locationService.Get( guid )?.ParentLocation;
4
5     while ( location != null )
6     {
7         if ( !expandGuids.Contains( location.Guid ) )
8         {
9             expandGuids.Add( location.Guid );
10        }
11
12        location = location.ParentLocation;
13    }
14 }
```



# Tree Field Type

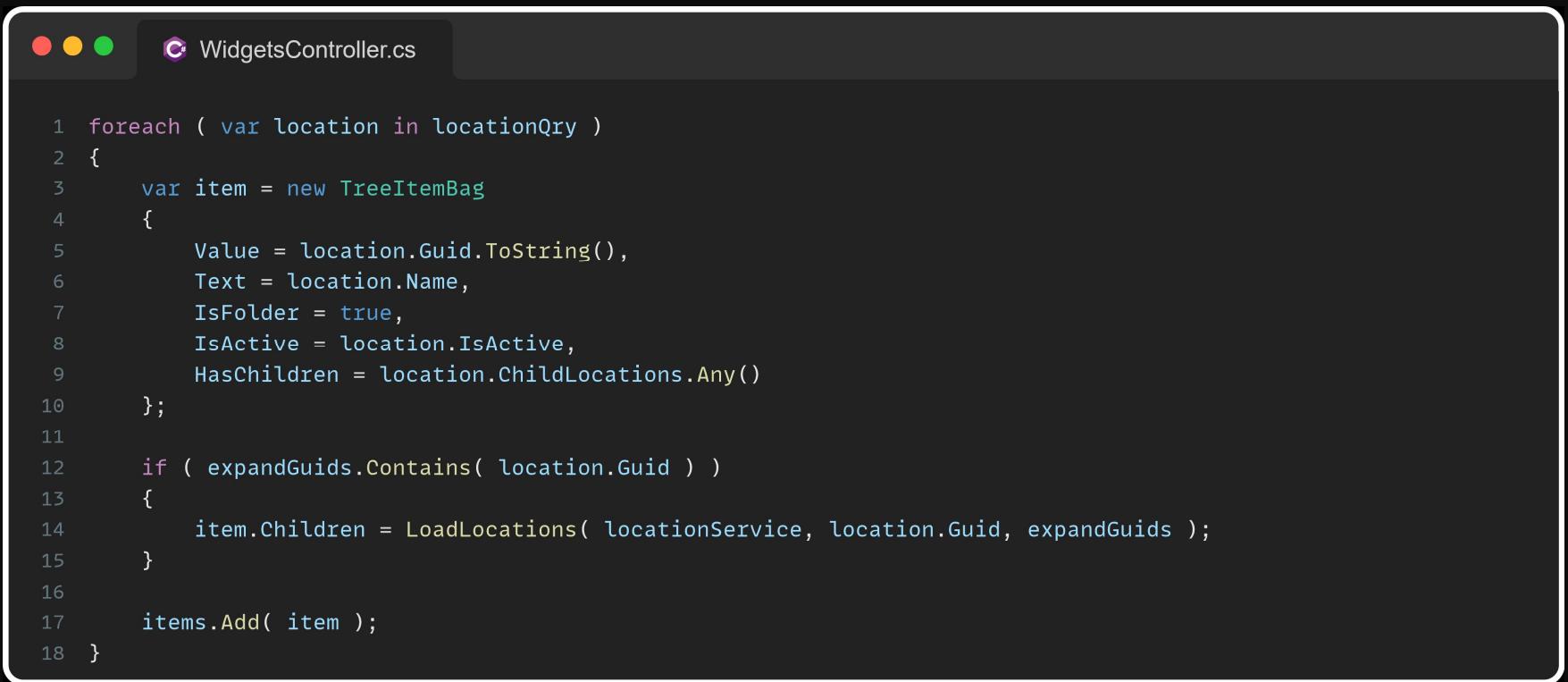


A screenshot of a code editor window titled "WidgetsController.cs". The code is written in C# and defines a method named "LoadLocations". The method takes three parameters: "LocationService", "Guid?", and "List<Guid>". It returns a "List<TreeItemBag>". The implementation uses LINQ queries to filter locations based on their name and parent location GUID. It then initializes a list of items and iterates through the query results to add them to the list. Finally, it returns the list of items.

```
1 private List<TreeItemBag> LoadLocations( LocationService locationService,
2     Guid? parentGuid,
3     List<Guid> expandGuids )
4 {
5     var locationQry = locationService.Queryable()
6         .Where( l => l.Name != null && l.Name != string.Empty )
7         .Where( l => ( parentGuid.HasValue && l.ParentLocation.Guid == parentGuid.Value )
8             || ( !parentGuid.HasValue && !l.ParentLocationId.HasValue ) );
9
10    var items = new List<TreeItemBag>();
11
12    foreach ( var location in locationQry )
13    {
14    }
15
16    return items;
17 }
```



# Tree Field Type



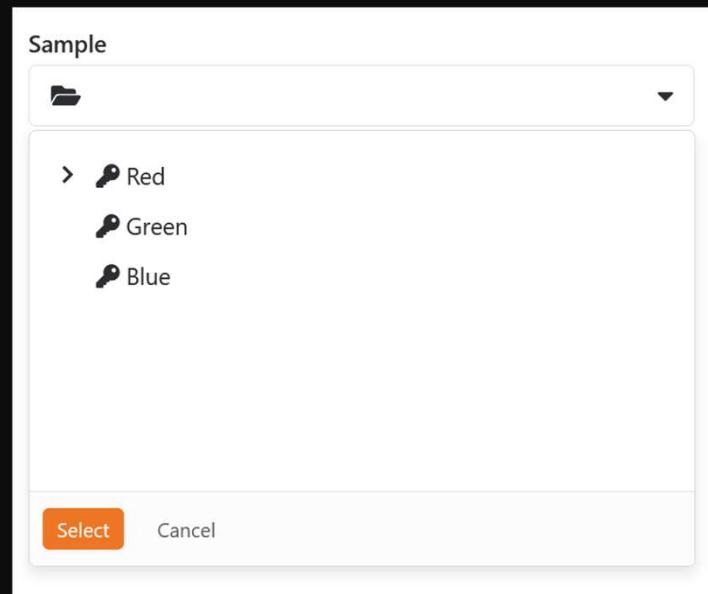
A screenshot of a code editor window titled "WidgetsController.cs". The code is written in C# and defines a method that iterates through a list of locations, creating a TreeItemBag for each one. It checks if the location's Guid is in a list of expandable GUIDs and adds children if it is. Finally, it adds the item to a list of items.

```
1 foreach ( var location in locationQry )
2 {
3     var item = new TreeItemBag
4     {
5         Value = location.Guid.ToString(),
6         Text = location.Name,
7         IsFolder = true,
8         IsActive = location.IsActive,
9         HasChildren = location.ChildLocations.Any()
10    };
11
12    if ( expandGuids.Contains( location.Guid ) )
13    {
14        item.Children = LoadLocations( locationService, location.Guid, expandGuids );
15    }
16
17    items.Add( item );
18 }
```



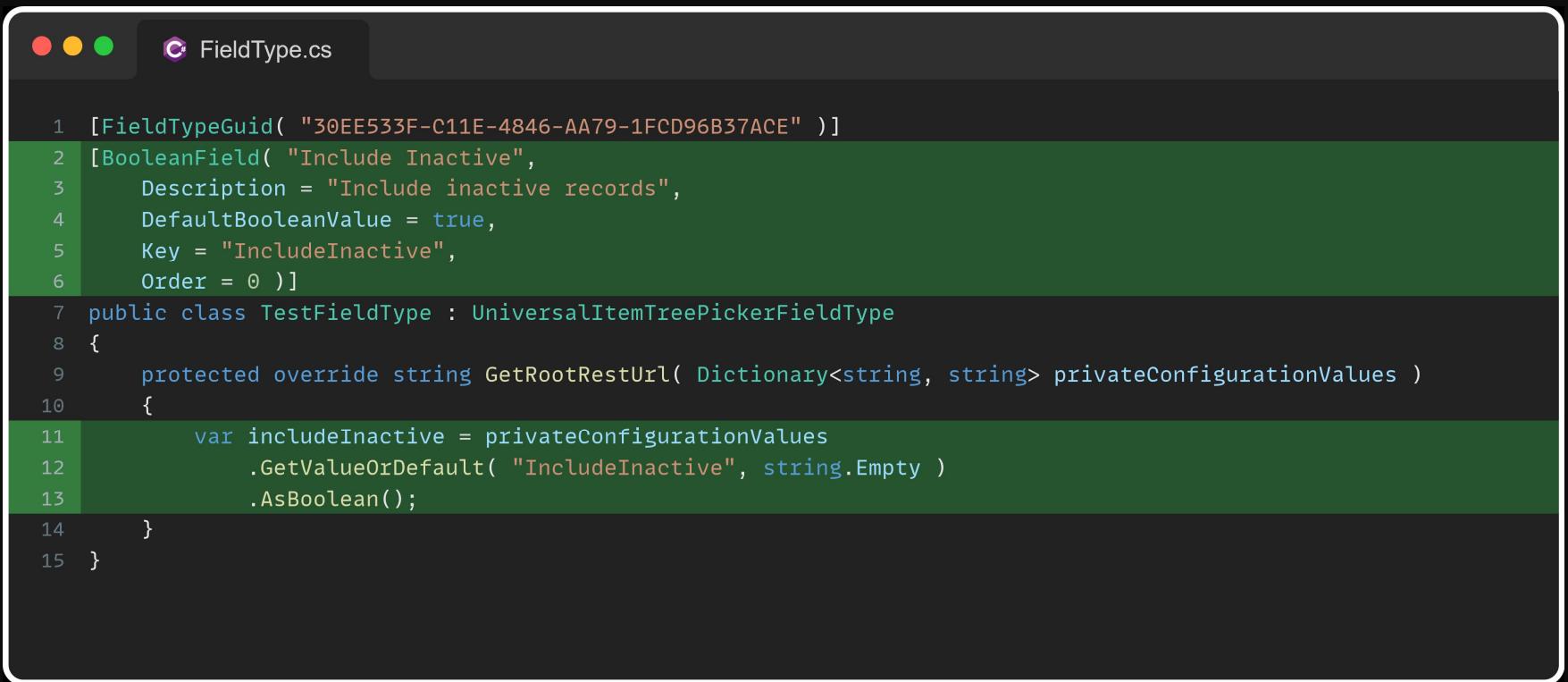
# Tree Field Type

---





# Field Type Configuration



A screenshot of a code editor window titled "FieldType.cs". The code is written in C# and defines a class named "TestFieldType" that inherits from "UniversalItemTreePickerFieldType". The class contains a constructor with attributes and a method named "GetRootRestUrl".

```
1 [FieldTypeGuid( "30EE533F-C11E-4846-AA79-1FCD96B37ACE" )]
2 [BooleanField( "Include Inactive",
3     Description = "Include inactive records",
4     DefaultBooleanValue = true,
5     Key = "IncludeInactive",
6     Order = 0 )]
7 public class TestFieldType : UniversalItemTreePickerFieldType
8 {
9     protected override string GetRootRestUrl( Dictionary<string, string> privateConfigurationValues )
10    {
11        var includeInactive = privateConfigurationValues
12            .GetValueOrDefault( "IncludeInactive", string.Empty )
13            .AsBoolean();
14    }
15 }
```



# Questions?

---