AssetTree Custom View Example

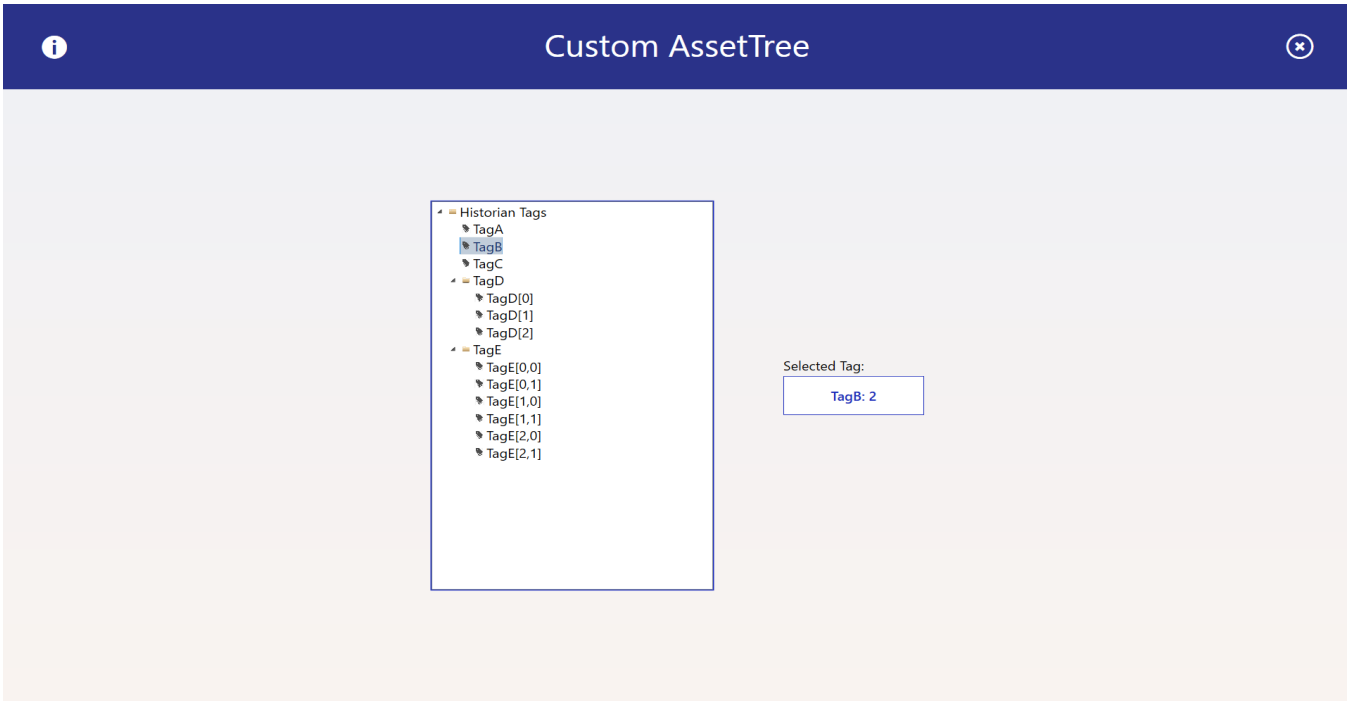
Customize your asset tree view.

Download the solution AssetTree Custom View Example.dbsln.

This solution example demonstrates how to create a personalized asset tree.

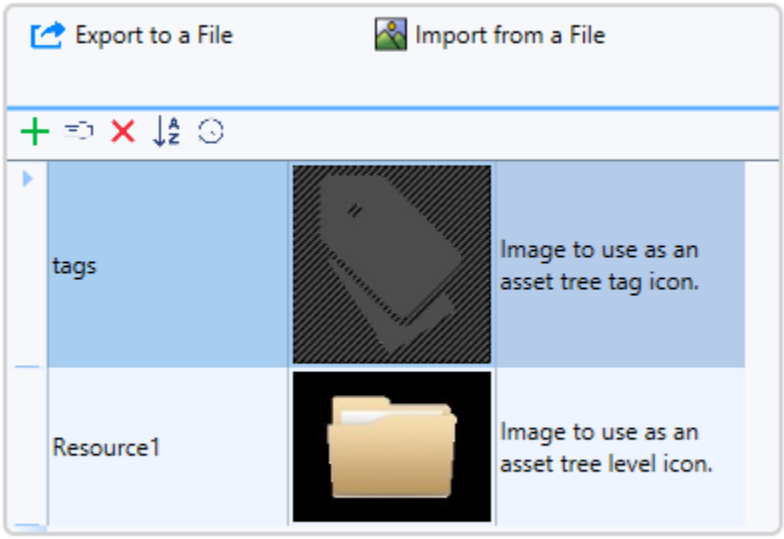
Summary

This solution example demonstrates how to create an asset tree with personalized labels, icons and tags.



Technical Information

To customize asset trees with your own icons, you first need to add images to your solution. To do so, go to Displays Images. You will be taken to a table containing all of your solution's images. Click on "Import from a File" and use the file browser to select the image you want. Once you are done, your Images table should look like this:



With the images in our solution file, we prepare the tags we want to display in the asset tree. These tags are stored in Table1 of the Historian Database. The tags' names are modified to follow a convention, and then they are added to a hash set, to track which tags are loaded into the tree.

```
DataTable table = await TK.ProjectDB.GetDataTableAsync("HistorianHistorianTags");
this.hashOfHistorian = new HashSet<string>();

foreach (DataRow row in table.Rows) {
    string tagName = TK.To<string>(row["TagName"]);

    // Remove ".Value" suffix if present
    if (tagName.EndsWith(".Value", StringComparison.CurrentCultureIgnoreCase)) {
        tagName = tagName.Remove(tagName.LastIndexOf('.'));
    }

    // Add "Tag." prefix if missing
    if (!tagName.StartsWith("Tag.", StringComparison.CurrentCultureIgnoreCase)) {
        tagName = "Tag." + tagName;
    }

    // Add tag to hash set
    this.hashOfHistorian.Add(tagName.ToLower());
}
```

To customize the asset tree, we first access the Display CodeBehind and get the asset tree's control. Then, we apply the folder icon we imported into the solution ("Resource1") to the tree root, and initialize the tree:

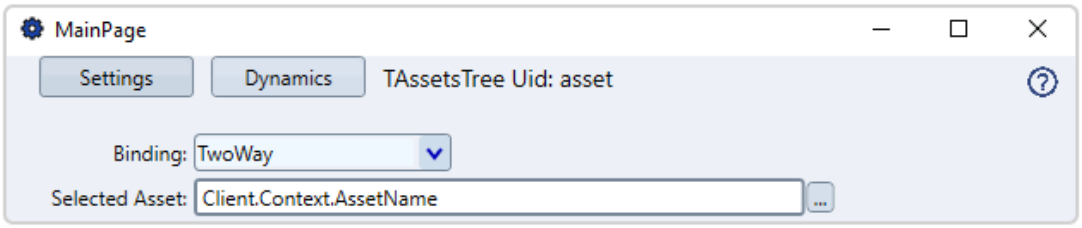
```
TAssetsTree control = this.CurrentDisplay.GetControl("asset") as TAssetsTree;

ImageSource iconRoot;
string rootName = "Historian Tags";

// Retrieves the folder image's ID from the solution's database using the image's name
int imageID = await TK.GetImageIDFromNameAsync("Resource1");
// Prepares to transfer the image as a stream of bytes
Stream stream = await TK.GetImageAsync(imageID) as Stream;
BitmapImage bitmapImage = new BitmapImage();
// Creates a bitmap image from the byte stream
// The #if #else macros ensure the procedure works fine on both WPF and HTML5
#if        HTML5_BROWSER
        bitmapImage.SetSource(stream);
#else
        bitmapImage.BeginInit();
        bitmapImage.StreamSource = stream;
        bitmapImage.EndInit();
#endif
iconRoot = bitmapImage;

// Creates the custom root of the asset tree
ElementItem tagRoot = new ElementItem(true, rootName, iconRoot, null, null);
control.InitializeCustom(new ElementItem[] {tagRoot}, this.GetObjectChildren);
```

GetObjectChildren is the method responsible for generating the tree structure. Initially, this method loads the images that will be applied to levels and tags, just like in the code snippet above. The rest of the code handles tree expansion when items are clicked. The code snippet below extracts an asset's name and deals with clicks on level assets:

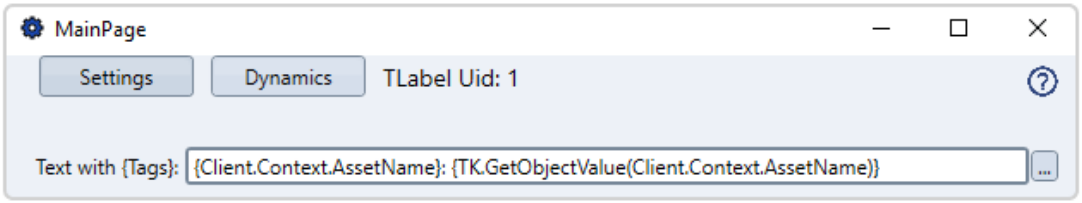
```
private async Task<ElementItem[]> GetObjectChildren(object sender, ElementItem item) {
        // List of elements to append to the tree
        List<ElementItem> elements = new List<ElementItem>();
        // Formats the string representing the selected item
        string asset = (item.FullDisplayString ?? "").Replace('\\', '.').Replace('|', '.').Replace(".[",
"[");
        if(asset.StartsWith("Historian Tags/"))
        {
                asset = asset.Substring(15);
        }
        else asset = "Tag";
        // Creates a reference to the object represented by the selected item
        ObjRef objRef = TK.ObjServer.DB.GetObjRef(asset, false);
    if (objRef != null) {
                if (objRef.RunObj is ListObj) {
        // Handle list-based tags
        foreach (RunObj runObj in objRef.RunObj as ListObj) {
            string fullName = runObj.GetName().ToLower();
            if (this.hashOfHistorian.Contains(fullName)) {
                                ElementItem elementItem;
                                if(runObj is ITagObj)
                                        elementItem = new ElementItem(false, runObj.GetSimpleName(),
iconTag, null, null);
                else
                                        elementItem = new ElementItem(true, runObj.GetSimpleName(),
iconLevel, null, null);
                elements.Add(elementItem);
            }
        }
    }
}
```

Note that the `GetObjectChildren` method is longer than this. Please check the CodeBehind of this solution's MainPage to see the full method, which includes code to interpret and display array tags.

Finally, to display the value of each tag, we open the asset tree's settings and configure its Selected Asset field as follows:



Then, we configure the Label containing the tag's name and value:



In this section...
