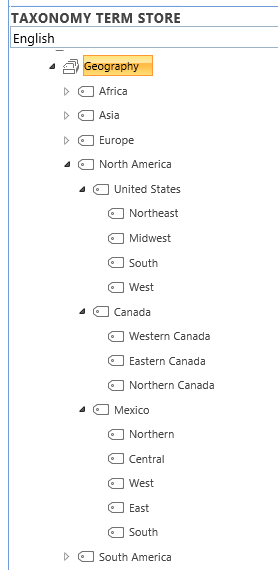
# Office AMS: CASCADED Taxonomy Picker for SharePoint Apps

|  |  |
| --- | --- |
| Summary: | Applies to: |
| This sample shows an implementation of a SharePoint Taxonomy Picker control that can be used on provider hosted SharePoint apps. | * Office 365 Multi Tenant (MT) * Office 365 Dedicated (D) * SharePoint 2013 on-premises |
| Solution: | Contoso.Components.TaxonomyPicker, version 1.0 |
| Author: | Anand Malli, Microsoft |
| //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // THIS CODE IS PROVIDED \*AS IS\* WITHOUT WARRANTY OF  // ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING ANY  // IMPLIED WARRANTIES OF FITNESS FOR A PARTICULAR  // PURPOSE, MERCHANTABILITY, OR NON-INFRINGEMENT.  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |

# How to use the CASCADED Taxonomy Picker in your Provider Hosted SP App

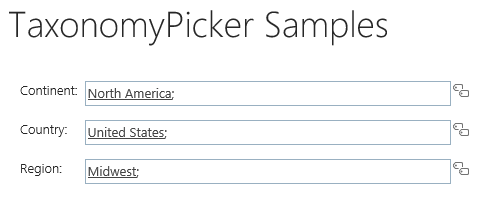
This sample is suitable when cascaded taxonomy picker control is required and you have termset structure similar to mentioned below:



Level - 1

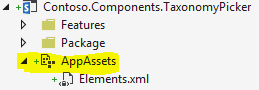
Level - 2

Level - 3

And you wanted to represent them like this with cascading filter functionalities:

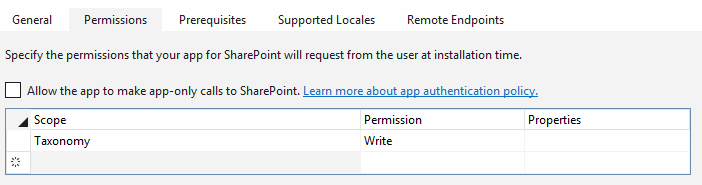
## Ensure you trigger the creation of an App Web

When you build a provider hosted app it does not necessarily have an app web associated with it whereas a SharePoint hosted app always has an app web. Since the Taxonomy Picker control uses the CSOM object model from JavaScript it’s required to have an app web. To ensure you have an app web you can just add an empty module to your SharePoint app as shown below:



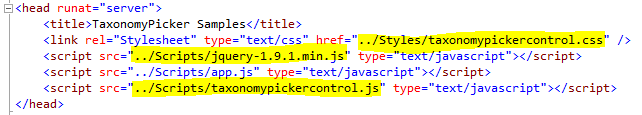
## App Permissions

The Taxonomy Picker communicates with SharePoint’s Managed Metadata Service, which requires special permissions in the app model. Working with **Closed** TermSets will require **Read** permission on the **Taxonomy** permission scope. To enable the creation of new terms in **Open** TermSets, the app will require **Write** permission on the **Taxonomy** permission scope. These permissions can be set in the AppManifest.xml as seen below:



## Required files

The Taxonomy Picker is implemented as a jQuery extension, which means it requires a reference to jQuery on and pages it will be used. In addition to jQuery, the Taxonomy Picker control requires the reference of a taxonomypicker.js and taxonomypicker.css files included in the sample solution.



## Loading Required Scripts and Establishing ClientContext

The Taxonomy Picker uses SharePoint’s JavaScript Client Object Model (JSOM) for communication back to SharePoint and the Managed Metadata Service. The JavaScript below shows how to load the appropriate JSOM scripts, initialize SharePoint ClientContext, and wiring up a RequestExecutor to make cross-domain calls. Notice the reference to sp.taxonomy.js, which is a JSOM script specific to working with taxonomies:

//Wait for the page to load

$(document).ready(function () {

//Get the URI decoded SharePoint site url from the SPHostUrl parameter.

var spHostUrl = decodeURIComponent(getQueryStringParameter('SPHostUrl'));

var appWebUrl = decodeURIComponent(getQueryStringParameter('SPAppWebUrl'));

var spLanguage = decodeURIComponent(getQueryStringParameter('SPLanguage'));

//Build absolute path to the layouts root with the spHostUrl

var layoutsRoot = spHostUrl + '/\_layouts/15/';

//load all appropriate scripts for the page to function

$.getScript(layoutsRoot + 'SP.Runtime.js',

function () {

$.getScript(layoutsRoot + 'SP.js',

function () {

//Load the SP.UI.Controls.js file to render the App Chrome

$.getScript(layoutsRoot + 'SP.UI.Controls.js', renderSPChrome);

//load scripts for cross-domain calls

$.getScript(layoutsRoot + 'SP.RequestExecutor.js', function () {

context = new SP.ClientContext(appWebUrl);

var factory = new SP.ProxyWebRequestExecutorFactory(appWebUrl);

context.set\_webRequestExecutorFactory(factory);

});

//load scripts for calling taxonomy APIs

$.getScript(layoutsRoot + 'init.js',

function () {

$.getScript(layoutsRoot + 'sp.taxonomy.js',

function () {

//READY TO INITALIZE TAXONOMY PICKERS

});

});

});

});

});

## Adding the CASCADED Taxonomy Picker to Html

Any hidden input element can be converted to a Taxonomy Picker. This includes regular hidden input elements and server-side controls that render hidden inputs elements (ex: asp:HiddenField):

Client-side example  
<input type="hidden" id="taxPickerContinent" />

Server-side example  
<asp:HiddenField runat="server" ID="taxPickerContinent" />

## Transforming the Html into a Taxonomy Picker Control

The Taxonomy Picker is implemented as a jQuery extension, which makes it extremely easy to wire-up on the hidden input element:

$('#taxPickerContinent').taxpicker({ isMulti: false, allowFillIn: false, useKeywords: false, termSetId: "0cc96f04-d32c-41e7-995f-0401c1f4fda8", levelToShowTerms: 1 }, context, callbackmethod);

Or

$('#taxPickerCountry').taxpicker({ isMulti: false, allowFillIn: false, useKeywords: false, termSetId: "0cc96f04-d32c-41e7-995f-0401c1f4fda8", filterTermId: this.\_selectedTerms[0].Id, levelToShowTerms: 2, useTermSetasRootNode: false }, context, initializeRegionTaxPicker);

PARAMETERS

|  |  |
| --- | --- |
| options | The first parameter of the Taxonomy Picker sets the options for the control. The properties that can be set include:   * isMulti – Boolean indicating if taxonomy picker support multiple value * allowFillIn – Boolean indicating if the control allows fill-ins (Open TermSets only) * termSetId – the GUID of the TermSet to bind against (available from Term Mgmt) * useHashtags – Boolean indicating if the default hashtags TermSet should be used * useKeyword – Boolean indicating if the default keywords TermSet should be used * maxSuggestions – integer for the max number of suggestions to list (defaults is 10) * lcid – the locale ID for creating terms (default is 1033) * language – the language code for the control (defaults to en-us) * filterTermId – the GUID of the term which will be used to filter terms from Termset for cascading (it is optional parameter) * useTermSetasRootNode – Boolean value indicating whether to display termset as root node in the selection control (it is optional parameter) * levelToShowTerms – Numeric value indicating how many level of cascading controls are being displayed |
| context | The second parameter is an initialized SP.ClientContext object |
| callbackmethod | The third parameter is callback method to call whenever control value changes |

Sample Implementations

$('#taxPickerContinent').taxpicker({ isMulti: false, allowFillIn: false, useKeywords: false, termSetId: "0cc96f04-d32c-41e7-995f-0401c1f4fda8", levelToShowTerms: 1 }, context, initializeCountryTaxPicker);

function initializeCountryTaxPicker() {

if (this.\_selectedTerms.length > 0) {

$('#taxPickerCountry').taxpicker({ isMulti: false, allowFillIn: false, useKeywords: false, termSetId: "0cc96f04-d32c-41e7-995f-0401c1f4fda8", filterTermId: this.\_selectedTerms[0].Id, levelToShowTerms: 2, useTermSetasRootNode: false }, context, initializeRegionTaxPicker);

}

}

function initializeRegionTaxPicker() {

if (this.\_selectedTerms.length > 0) {

$('#taxPickerRegion').taxpicker({ isMulti: false, allowFillIn: false, useKeywords: false, termSetId: "0cc96f04-d32c-41e7-995f-0401c1f4fda8", filterTermId: this.\_selectedTerms[0].Id, levelToShowTerms: 3, useTermSetasRootNode: false }, context);

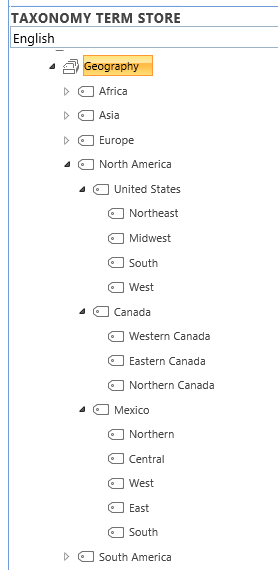
}

}

## HOW to run this sample

The sample project includes app.js file, containing initialization methods to set up the cascading taxonomy picker control.

Please ensure that you are already having a termset containing terms for at least 2 level.



Level - 1

Level - 2

Level - 3

Find out the GUID of the TermSet to bind (using Site Settings 🡪 Term Store Management) & update below line with actual TermSet GUID.

$('#taxPickerContinent').taxpicker({ isMulti: false, allowFillIn: false, useKeywords: false, termSetId: "<<TERMSET GUID>>", levelToShowTerms: 1 }, context, initializeCountryTaxPicker);

$('#taxPickerCountry').taxpicker({ isMulti: false, allowFillIn: false, useKeywords: false, termSetId: "<<TERMSET GUID>>", filterTermId: this.\_selectedTerms[0].Id, levelToShowTerms: 2, useTermSetasRootNode: false }, context, initializeRegionTaxPicker);

$('#taxPickerRegion').taxpicker({ isMulti: false, allowFillIn: false, useKeywords: false, termSetId: "<<TERMSET GUID>>", filterTermId: this.\_selectedTerms[0].Id, levelToShowTerms: 3, useTermSetasRootNode: false }, context);

## Setting Values

The sample project includes a TaxonomyPickerExtensions.cs file, containing extension methods to help set values of a Taxonomy Picker server-side. This includes extension methods for converting TaxonomyFieldValue and TaxonomyFieldValueCollection objects into JSON that the Taxonomy Picker script can read from the hidden fields. Here is an example of using these methods to set the value of two Taxonomy Picker fields using C#:

//The following code shows how to set a taxonomy field server-side

var spContext = SharePointContextProvider.Current.GetSharePointContext(Context);

using (var clientContext = spContext.CreateUserClientContextForSPHost())

{

var list = clientContext.Web.Lists.GetByTitle("MyList");

var listItem = list.GetItemById(1);

clientContext.Load(listItem);

clientContext.ExecuteQuery();

taxPickerGeographySingle.Value =

((TaxonomyFieldValue)listItem["SomeTaxFieldSingle"]).Serialize();

taxPickerGeographyMulti.Value =

((TaxonomyFieldValueCollection)listItem["SomeTaxFieldMulti"]).Serialize();

}

## Reading Values

The Taxonomy Picker will store the selected terms in the hidden field using JSON string format. These values can be accessed by other client-side scripts or server-side following a post. The JSON will include the Term Name, Id, and PathOfTerm (ex: World;North America;United States). JSON.parse can be used client-side to convert the hidden input’s value to a typed object and any number of server-side libraries can be used (ex: JSON.net)

# Language Support

The strings displayed by the control will be loaded dynamically based on the passed language. This requires you to pass the language via taking over the SPLanguage url parameter (see sample) or by hardcoding it. If no language is passed the control assumes the language is English (en-us).

$('#taxPickerContinent').taxpicker({ isMulti: true, allowFillIn: true, useKeywords: true, lcid: 1031, language: 'de-de' }, context);

If you would like to add additional languages you need to create the appropriate JavaScript language resource files:



Such a resource file is simple collection of global constants:

