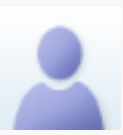
6 Replies | Latest reply on Apr 10, 2018 7:41 PM by ruser3t30054468



ruser3t30054468 Mar 30, 2018 12:58 AM

[Br] カスタムフィルタについて

This question is **Not Answered.** (Mark as assumed answered)

BridgeSDK付属のサンプルスクリプト(CustomSortExtensionHandler.jsx, SDKNode.jsx, SDKSystem.jsx)を元にして、フィルタを追加しようとしています。

本体

```
01. function CustomFilterExtensionHandler() {
02.
03.
04.     $.level = 2; // Debugging level
05.     /**
06.       The context in which this sample can run.
07.       @type String
08.     */
09.     this.requiredContext = "\tNeed to be running in context of Bridge\n";
10. }
11.
12.
13. /**
14.  Functional part of this sample, creates several nodes of the new type, and one ma
15.  which stores the node-identifying prefix and root node.
16.
17.  <p>Creates the ExtensionHandler object, whose definition includes
18.  the 'makeModel()' method, which Bridge uses to create the ExtensionModel object
19.  at node-display time.
20.
21.  Registers the ExtensionHandler, associating it with the node-identifying prefix.
22.      @return True if the sample ran as expected, false otherwise
23.      @type Boolean
24. */
25. CustomFilterExtensionHandler.prototype.run = function()
26. {
27.     if(!this.canRun()) { return false; }
28.
29.
30.     $.writeln("About to run CustomFilterExtensionHandler");
31.
32.
33.     // Make sure the XMP object is loaded and available so we can access a nodes m
34.     if( XmpScriptLib == undefined ) {
35.         if( Folder.fs == "Windows" ) {
36.             var pathToLib = Folder.startup.fsName + "/AdobeXMPScript.dll";
37.         } else {
38.             var pathToLib = Folder.startup.fsName + "/AdobeXMPScript.framework";
```

```
        }
        var libfile = new File( pathToLib );
        var XmpScriptLib = new ExternalObject("lib:" + pathToLib );
    }

    var rootFs = app.document.thumbnail.spec;

    // Create some nodes of the script-defined type
    var root = new SDKNode(rootFs.name, true);


    for (var i = 0;i < rootFs.getFiles ().length; i++){
        if (rootFs.getFiles ()[i].constructor.name == "Folder"){
            root.addNode (new SDKNode (rootFs.getFiles ()[i].name, true));
        }else{
            root.addNode (new SDKNode (rootFs.getFiles ()[i].name, false));
        };
    };


    // Create the manager object with the node-identifying prefix,
    // and add the root node
    var sys = new SDKSystem("bridge:cfNode:", root, rootFs.parent.fsName);


    // Create the node extension handler object
    var CFHandler = new ExtensionHandler("customFilterHandler");


    // Register the node handler and prefix    with the application
    app.registerExtension(CFHandler);
    app.registerPrefix( "bridge:cfNode:", CFHandler);


    /**
        Required. Returns a canonical URI for the path.  If the path does not con
        registered prefix, the this method must append it to the path and return
        bridge:beNode:/CSRoot
        @param path The path of the node as a string
    */
    CFHandler.getBridgeUriForPath = function(path) { return path; }


    /**
        Extends the handler with a helper function so all the thumbnails can
        be collected for the search
        @param kids An array of SDK nodes.
    */
    ExtensionHandler.prototype.getThumbsToSearch = function(kids)
    {
        for(var h = 0;h < kids.length;h++)
        {
            if(kids[h].isContainer()) { this.getThumbsToSearch(kids[h].getChildre
            CFHandler.thumbsToSearch.push(kids[h]);
        }
    }


    /**
        Search implementation: Performs a search and returns the search result,
        a container Thumbnail object that contains the matching nodes.
```

```
100.          The passed 'scope' is the Thumbnail object that the search starts from, a
101.          'spec' is the SearchSpecification object containing all search details as
102.          set by the user in the Find dialog.
103.
104.          @param scope  The Thumbnail to start searching from
105.          @param spec The SearchSpecification details
106.      */
107.      CFHandler.getBridgeUriForSearch = function(scope, spec)
108.      {
109.          // Create a search results node - give it a unique name for each search c
110.          var randomnumber=Math.floor(Math.random()*101);
111.          var d = new Date();
112.          var postfix = d.getSeconds() + randomnumber;
113.
114.
115.          var sNode = new SDKNode("SearchNode" + postfix, true)
116.          sNode.children = new Array();
117.          sys.addSearchNode(sNode); // This overwrites any previous search node obj
118.
119.
120.          var sThumb = new Thumbnail("bridge:cfNode:/SearchNode" + postfix);
121.          sThumb.core.children.cacheData.status = "bad";
122.
123.
124.          // Get the scope specifiers (whether to search subfolders)
125.          var searchScopes = spec.scopeSpecifiers;
126.
127.
128.          //  Create an array to hold the thumbnails to   be checked against search
129.          CFHandler.thumbsToSearch = new Array();
130.
131.
132.          // This is the node from which to start
133.          var scopeNode = sys.getNodeFromPath(scope.uri);
134.
135.
136.          var children;
137.          // Check if the search is to include subfolders
138.          if(searchScopes[0] == "DoSubFolders")
139.          {
140.              // Get the node children and find all the thumbnaiils we need to sear
141.              this.getThumbsToSearch(scopeNode.getChildren());
142.
143.
144.              children = CFHandler.thumbsToSearch;
145.          }
146.          else
147.          {
148.              children = scopeNode.getChildren();
149.          }
150.
151.          // Get the conditions of the search, as specified by the user
152.          var conditions = spec.conditionList;
153.
154.
155.          if(spec.conjunction == "or")    // This search only supports an "OR" conj
156.          {
157.              // Check each thumbnail/node and see if the criteria match
158.              for(var i = 0;i < children.length;i++)
159.              {
160.                  var currentNode = children[i];
```

```
161.
162.                    for(j = 0;j < conditions.length;j++)
163.                    {
164.                        var currentCon = conditions[j]
165.                        var searchField = currentCon.searchField;
166.                        var operand = currentCon.operand;
167.
168.
169.                        // Get the values for the field/data we are searching against
170.                        switch(searchField)
171.                        {
172.                            var thumbFieldValue;
173.                            case "CustomString":
174.                                thumbFieldValue = currentNode.getCustomString();
175.                                break;
176.                            case "CustomNumber":
177.                                thumbFieldValue = currentNode.getCustomNumber();
178.                                break;
179.                            case "CustomBool":
180.                                thumbFieldValue = currentNode.getCustomBool();
181.                                break;
182.                            case "name":
183.                                thumbFieldValue = currentNode.getName();
184.                                break;
185.                        }
186.
187.
188.                        // Check if anything matches the criteria
189.                        // This is an OR search, so when a node hits any match,
190.                        // add it to the children of the search container
191.                        switch(currentCon.operatorType)
192.                        {
193.                            case "equals":
194.                                // catches boolean values and strings
195.                                if(thumbFieldValue == operand || thumbFieldValue.toSt
196.                                break;
197.                            case "lessThanOrEqual":
198.                                if(thumbFieldValue <= operand) { addNodeToSearchResul
199.                                break;
200.                            case "greaterThanOrEqual":
201.                                if(thumbFieldValue >= operand) { addNodeToSearchResul
202.                                break;
203.                            case "endsWith":
204.                                var s = thumbFieldValue.substr((thumbFieldValue.lengt
205.                                if(s == operand) { addNodeToSearchResult(currentNode)
206.                                break;
207.                            default:
208.                        } // end switch
209.                    } // end for condtions
210.                } // end for children
211.            }
212.            else if(spec.conjunction == "and")  // This search only supports "OR" sea
213.            {
214.                alert("SDK: CustomFilterExtensionHandler:\nSample does not show this
215.            }
216.
217.
218.            // Helper function to check for the existence of a node before adding it
219.            function addNodeToSearchResult(node)
220.            {
221.                var matches = sNode.getChildren();
```

```
            var addNode = true;
            for(var i = 0;i < matches.length;i++)
            {
                if(matches[i].getPath() == node.getPath())
                {
                    addNode = false;
                    break;
                }
            }


            if(addNode) { sNode.addNode(node); }
        }


        // Return the URI to the container (a Thumbnail object) holding the searcl
        return sThumb.uri;
    }


    /**
        Creates an operator that performs the long running operation.  In this cas
        made to the this nodes XMP data.

        @param targets
        @param xmpPackets
        @param timeoutInMs Optional. The number of microseconds after wich to abo
        @param showUi Optional.  Wether to show the UI or not
        @param message Optional. A string to display via the progress UI
    */
    CFHandler.setXMP = function(targets, xmpPackets, timeoutInMs, showUi, message
    {
        var thumbs = targets;
        var xmpData = xmpPackets;

        var op = new ProgressOperator("progress", thumbs);

        op.start = function()
        {
            op.procStat = "notStarted";

            op.opStat = "incomplete";
            app.operationChanged(this);

            op.procStat = "inProgress";


            this.processedNodeCount = 0;


            app.operationChanged(this);

            op.percComp = 1;

            for(var i = 0;i < thumbs.length;i++)
            {


                // Only added so we can see the ProgressOperator UI in action
                for(var j = 0;j < 10;j++)
                {
```

```
                    $.sleep(50);
                }
                op.percComp += (100 / thumbs.length);
                app.operationChanged( this );

                var t = thumbs[i];
                t.core.metadata.metadata = new Metadata(xmpData[i]);
                t.model.privateData.node.setMetadata(new Metadata(xmpData[i]));

                this.processedNodeCount++;
            }


            op.percComp = 100;
            op.opStat = "succeeded";
            op.procStat = "completed";

            app.operationChanged( this );
            return;
        }


        op.stop = function() { return; }

        op.getConflictInfo = function() { return "unknown"; }

        op.getOperationStatus = function() { return op.opStat; }

        op.getPercentageComplete = function() { return op.percComp; }

        op.getProcessedNodeCount = function() { return this.processedNodeCount; }

        op.getProcessingStatus = function() { return op.procStat; }

        op.getProgressMessage = function() { return "Setting XMP on selected files

        op.getTotalBytesTransferred = function() { return 0; }

        op.getTotalNodeCount = function() { return thumbs.length; }

        op.getType = function() { return "progress"; }

        op.getUIPolicy = function() { return this.showUi; }

        op.resume = function() { return false; }

        op.resolveConflict = function() { }


        app.enqueueOperation(op);
        //return op;


    }


    var wrapperObject = this;
    /**
        Required. Creates and returns an ExtensionModel instance.  Called each ti
        is displayed.  The model must also implement any methods that handle any
        functionality, such as renaming a node.
```

```
344.          @return An ExtensionModel for this node
345.          @type ExtensionModel
346.      */
347.      CFHandler.makeModel = function(path)
348.      {
349.              // Add a a property to store the manual sort
350.          ExtensionModel.prototype.sortXML = "";
351.          filterModel = new ExtensionModel(path);
352.          filterModel.path = path;
353.          this.path = path;
354.          wrapperObject.initModel(filterModel, sys);
355.
356.          return filterModel;
357.
358.
359.      } // end makeModel
360.
361.
362.      // Create the InfosetMemberDescription objects
363.      var csDescriptions = new Array();
364.      csDescriptions[0] = new InfosetMemberDescription( "CustomString", "String" );
365.      csDescriptions[1] = new InfosetMemberDescription( "CustomNumber", "Integer" )
366.      csDescriptions[2] = new InfosetMemberDescription( "CustomDate", "String" );
367.      csDescriptions[3] = new InfosetMemberDescription( "CustomArray", "Array of St
368.      csDescriptions[4] = new InfosetMemberDescription( "CustomBool", "Boolean" );
369.
370.      csDescriptions[5] = new InfosetMemberDescription( "FontName", "Array of Strin
371.      csDescriptions[6] = new InfosetMemberDescription( "FontFamily", "Array of Str
372.
373.
374.      // Create the infoset
375.      var customInfoset = new Infoset("customInfoset", csDescriptions);
376.
377.
378.      // Register the infoset
379.      app.registerInfoset(CFHandler, customInfoset);
380.
381.      // Event handler displays the Custom Infoset Data palette for handled thumbna
382.      onEvent = function(event)
383.      {
384.          if(event.object instanceof Document)
385.          {
386.              if(event.type == "selectionsChanged")
387.              {
388.                  if(event.object.selections.length > 0)
389.                  {
390.                      var thumb = event.object.selections[0];
391.                      if(thumb.customFilterHandler)
392.                      {
393.                          stringText.text = thumb.customFilterHandler.customInfoset
394.                          numbersText.text = thumb.customFilterHandler.customInfose
395.                          dateText.text = thumb.customFilterHandler.customInfoset.Cu
396.                          arrStringText.text = thumb.customFilterHandler.customInfo
397.                          boolList.selection = (thumb.customFilterHandler.customInf
398.                          customSearchDataPalette.content.visible = true;
399.                      }
400.                      else
401.                      {
402.                          customSearchDataPalette.content.visible= false;
403.                      }
404.                  }
```

```
                else
                {
                    customSearchDataPalette.content.visible= false;
                }
            }
        }
    }


    // Register the event handler
    app.eventHandlers.push({handler: onEvent});


    // Create  a palette to display the custom data
    var customSearchDataPalette = new TabbedPalette(app.document, "SDK: Custom So
    var win = customSearchDataPalette.content;


    var stringLabel = win.add("statictext", [5, 9, 115, 25], "Custom String");
    var stringText = win.add("edittext", [120, 5, 320, 25], "");
    stringText.enabled = false;

    var numbersLabel = win.add("statictext", [5, 39, 115, 55], "Custom Number");
    var numbersText = win.add("edittext", [120, 35, 320, 55], "");
    numbersText.enabled = false;


    var dateLabel = win.add("statictext", [5, 69, 115, 85], "Custom Date");
    var dateText = win.add("edittext", [120, 65, 320, 85], "");
    dateText.enabled = false;


    var arrLabel = win.add("statictext", [5, 99, 115, 115], "Custom Array");
    var arrStringText = win.add("edittext", [120, 95, 320, 115], "");
    arrStringText.enabled = false;

    var boolLabel = win.add("statictext", [5, 129, 115, 145], "Custom Boolean");
    var boolList = win.add("dropdownlist", [120, 125, 320, 145]);
    var tItem = boolList.add("item", "True");
    tItem.val = true;
    var fItem = boolList.add("item", "False");
    fItem.val = false;
    boolList.enabled = false;

    // Hide the palette until a handled node is selected
    customSearchDataPalette.content.visible = false;


    // create a thumbnail and add it to the Favorites panel
    var t = new Thumbnail(sys.getPrefix() + sys.fsRoot + sys.root.path);
    app.favorites.insert(t);
    t.refresh();


    $.writeln("CustomFilterExtensionHandler initialized successfully");
    return true;
}
```

```
466.
467.  /**
468.    Determines whether sample can be run given current context.  The sample
469.    fails if these preconditions are not met:
470.    <ul>
471.    <li> Bridge must be running
472.    <li> Resource files must be available
473.    </ul>
474.    @return True if this sample can run, false otherwise
475.    @type Boolean
476.  */
477.  CustomFilterExtensionHandler.prototype.canRun = function() {
478.      if(BridgeTalk.appName == "bridge") {
479.
480.          for(index in app.extensions)
481.          {
482.              if(app.extensions[index].name == "customFilterHandler")
483.              {
484.                  $.writeln("ERROR: Cannont run twice! You must restart Bridge");
485.                  return false;
486.              }
487.          }
488.          return true;
489.      }
490.      $.writeln("ERROR:: Cannot run CustomFilterExtensionHandler");
491.      $.writeln(this.requiredContext);
492.      return false;
493.  }
494.
495.
496.  /**
497.      Initialize the model and define the methods that this model supports
498.      @param sortModel The model for the handled nodes
499.      @param sys The manager object for the handled nodes
500.  */
501.  CustomFilterExtensionHandler.prototype.initModel = function(filterModel, sys)
502.  {
503.          /**
504.              Implements the customization of the Find dialog for the custom search
505.              Creates and returns a SearchDefinition object used for the customizat
506.
507.              @return A customized SearchDefinition
508.              @type SearchDefinition
509.          */
510.          filterModel.getSearchDefinition = function()
511.          {
512.
513.
514.              // Criteria section
515.              // Add values to the search field (left box) of the Criteria section
516.              var searchField = "CustomString";
517.              var opType = "string";
518.              var searchFieldDisplay = "Custom String";
519.
520.
521.              var searchCriteria1 = new SearchCriteria(searchField, opType, searchF
522.
523.
524.              // Restrict values for the operator (middle box) of the Criteria sect
525.              searchCriteria1.operatorTypesToDisable = ["exists", "equals", "doesNo
526.                  "lessThanOrEqual", "greaterThanOrEqual", "contains",
```

```
                "doesNotContain", "startsWith", "greater"];

            searchField = "CustomNumber";
            opType = "float";
            searchFieldDisplay = "Custom Number";
            var searchCriteria2 = new SearchCriteria(searchField, opType, searchF
            searchCriteria2.operatorTypesToDisable = ["exists", "doesNotExist", "
                "contains",     "doesNotContain", "endsWith",
                "startsWith", "greater"];

            searchField = "name";
            opType = "string";
            searchFieldDisplay = "Name";
            var searchCriteria3 = new SearchCriteria(searchField, opType, searchF
            searchCriteria3.operatorTypesToDisable = ["exists", "doesNotExist", "
                "lessThanOrEqual", "greaterThanOrEqual", "contains",
                "doesNotContain", "endsWith", "startsWith", "greater"];


            searchField = "CustomBool";
            opType = "boolean";
            searchFieldDisplay = "Custom Boolean";

            // Create some Operand objects
            // these populate the comparison-value choices in the far right box o
            var op1 = new Operand("true", "A true value");
            var op2 = new Operand("false", "A false value");
            var myOperators = [op1, op2];


            var searchCriteria4 = new SearchCriteria(searchField, opType, searchF
            searchCriteria4.operatorTypesToDisable = ["exists", "doesNotExist", "
                "lessThanOrEqual", "greaterThanOrEqual", "contains",
                "doesNotContain", "endsWith", "startsWith", "greater"];


            var critList = [searchCriteria1, searchCriteria2, searchCriteria3, se


            // Results section
            //var rank1 = new Rank("RankValueName", "RankDisplayName");
            //var rank2 = new Rank("RankValueName", "RankDisplayName");


            var ranksArray = [];  // These are optional, but empty arrays must be


            // create some scopeSpecifier objects
            var scope1 = new Scope( "DoSubFolders", "SDKSearch: Search subfolders
            var scope2  = new Scope("DoHidden", "SDKSearch: Search hidden files")
            // the scopeSpecifiers to an Array
            var scopesArray = [scope1, scope2]; // These are documented as optiona


            var searchDef = new SearchDefinition(critList, 10, ranksArray, scopesA
            return searchDef;
        }

            // Renames a node
        filterModel.setName = function(name)
        {
```

```
588.
589.
590.                // Get the parent of the node
591.                var parent = this.privateData.node.getParent();
592.                var update = new Thumbnail(sys.getPrefix() + parent.getPath());
593.
594.
595.                // Set the cache status to bad as we want a refresh to occur
596.                update.core.children.cacheData.status = "bad";
597.
598.
599.                // Change the name of the node
600.                this.privateData.node.setName(name);
601.
602.
603.                // If the name has been changed whilst the node is viewed in a search
604.                if(app.document.thumbnail.name == "SearchNode") { app.document.thumbna
605.
606.
607.                var newURI = sys.getPrefix() + this.privateData.node.getPath();
608.                return newURI;
609.            } // end setName
610.
611.            // Creates a new container of the script defined type
612.            // Must return an operator as this is a long running operation.
613.            // Creates a new container of the script defined type
614.            // Must return an operator as this is a long running operation.
615.            filterModel.createNewContainer = function(name)
616.            {
617.                var newNode = new SDKNode(name, true)
618.                this.privateData.node.addNode(newNode);
619.
620.
621.                // Set the cache status to bad as we want a refresh to occur
622.                this.privateData.cacheElement.children.cacheData.status = "bad";
623.                //this.refreshInfoset("all");
624.
625.
626.                return sys.getPrefix() + newNode.getPath();
627.            } // end createNewContainer
628.
629.            /**
630.                Checks whether this node requires authentication.
631.                @return True if this node requires authentication, false otherwise
632.                @type Boolean
633.            */
634.            filterModel.needAuthentication = function() { return false; }
635.
636.
637.            /**
638.                Required.  Carries out any necessary authentication.
639.            */
640.            filterModel.authenticate = function() { return; }
641.
642.            /**
643.                Required,  Checks if this node is a valid node.
644.                @return True if this node is valid, false otherwise
645.                @type Boolean
646.            */
647.            filterModel.exists = function() { return true; }
648.
```

```
649.
650.            /**
651.                Required.  Called after the model is created, sets up any necessary da
652.                that may be needed.
653.            */
654.            filterModel.initialize = function() { this.privateData.node = sys.getNodel
655.
656.            /**
657.                Required.  Stores a back pointer to the cache that contains this model
658.                @param cacheEelement The CacheElement that contains this model.
659.            */
660.            filterModel.registerInterest = function(cacheElement) { this.privateData.c
661.
662.            /**
663.                Required. Removes the association between the CacheElement and
664.                this model instance.
665.            */
666.            filterModel.unregisterInterest = function() { this.privateData.cacheElemer
667.
668.            /**
669.                Adds a custom filter criteria to Bridge.  Single FilterDescription is
670.                This allows hanlded nodes to be filtered based on a string value store
671.            */
672.            filterModel.getFilterCriteria = function()
673.            {
674.                var filter1 = new FilterDescription ("FontName", "使用フォント", "strir
675.                defFilters.push (filter1);
676.
677.                var filter2 = new FilterDescription ("FontFamily", "フォントファミリ",
678.                defFilters.push (filter2);
679.
680.                return defFilters;
681.            }
682.
683.
684.            /**
685.                Adds custom SortCriterion to Bridge for handled nodes.
686.                Three new sort options are added to Bridge's default sort criterion.
687.                The third uses a custom namespace and custom XMP data.
688.                @return An array of SortCriterion objects
689.            */
690.            filterModel.getSortCriteria = function()
691.            {
692.                var dsc = app.defaultSortCriteria;
693.                return dsc;
694.            }
695.
696.            /**
697.                Returns the parent of this node
698.            */
699.            filterModel.getParent = function()
700.            {
701.                var parent = this.privateData.node.getParent();
702.                if(parent != undefined) { return (sys.getPrefix() + parent.getPath())
703.            }
704.
705.            filterModel.supportsUserSortOrder = function() { return true; }
706.
707.
708.            /**
709.                Stores the custom sort for this container
```

```
710.          */
711.          filterModel.setUserSortOrder = function(inXML) { this.sortXML = inXML; }
712.
713.          /**
714.              Return an XML string that gives the custom sort of the children
715.              of this container.
716.          */
717.          filterModel.getUserSortOrder = function()
718.          {
719.              if(this.sortXML == "")
720.              {
721.                  var xml = "<?xml version='1.0' encoding='UTF-8' standalone='yes'
722.                  var children = this.privateData.node.getChildren();
723.                  for(var i = 0;i < children.length;i++)
724.                  {
725.                      // Use 00000000000000 if creattiondate is not supported
726.                      xml += "<item key='" + children[i].getName() + "0000000000000
727.                  }
728.                  xml += "</files></dirinfo>";
729.                  this.sortXML = xml;
730.              }
731.
732.              return this.sortXML;
733.          }
734.
735.          filterModel.addToDrag = function(pointerToOsDragObject) { return true; }
736.
737.          filterModel.wouldAcceptDrop = function(type, sources, osDragRef) { return
738.
739.          /**
740.              Required.  Updates the core node data and any custom node data
741.              for this node.  Always set the CacheData.status after updating the no
742.              @param infosetName The name of the Infoset that is being updated.
743.          */
744.          filterModel.refreshInfoset = function(infosetName)
745.          {
746.              var infoset = this.privateData.cacheElement[infosetName];
747.              var currentNode = this.privateData.node;
748.              var fsThumb = new Thumbnail ("bridge:fs:" + currentNode.getFileUrl(sy
749.              try
750.              {
751.                  if(infosetName == "immediate" || infosetName == "all")
752.                  {
753.                      if(this.privateData.node.getName().substr(0,10) == "SearchNod
754.                      {
755.                          infoset.name = "SearchNode";
756.                      }
757.                      else
758.                      {
759.                          infoset.name = fsThumb.core.immediate.name;
760.
761.                          //値を渡した後のどこかの処理で配列の順序が逆転するため、表示さ
762.                          //逆順で渡して対処で正しいのかどうか。そもそもどういう意図の配
763.                          //infoset中の配列は全部順序が逆転する？
764.                          infoset.size = fsThumb.core.immediate.size.reverse();
765.                          infoset.creationDate = fsThumb.core.immediate.creationDat
766.                          infoset.modificationDate = fsThumb.core.immediate.modific
767.
768.
769.                          infoset.isHidden = fsThumb.core.immediate.isHidden;
770.                          infoset.isLink = fsThumb.core.immediate.isLink;
```

```
                        infoset.isPackage = fsThumb.core.immediate.isPackage;
                        infoset.isDeleted = fsThumb.core.immediate.isDeleted;
                        infoset.isApplication = fsThumb.core.immediate.isApplicat
                        infoset.isContainer = fsThumb.core.immediate.isContainer;
                        infoset.isAssortment = fsThumb.core.immediate.isAssortmen
                        infoset.sortIndex = fsThumb.core.immediate.sortIndex;
                        infoset.fileUrl = fsThumb.core.immediate.fileUrl;
                        infoset.displayPath = fsThumb.core.immediate.displayPath;
                    }
                    infoset.cacheData.status = "good";
                }


                // The capabilities of a handled node
                if(infosetName == "item" || infosetName == "all")
                {
                    infoset.canCreateNewContainer = fsThumb.core.item.canCreateNe
                    infoset.canCreateNewLink = fsThumb.core.item.canCreateNewLink
                    infoset.canDelete = fsThumb.core.item.canDelete;
                    infoset.canDuplicate = fsThumb.core.item.canDuplicate;
                    infoset.canBeDragSource = fsThumb.core.item.canBeDragSource;
                    infoset.canBeDropTarget = fsThumb.core.item.canBeDropTarget;
                    infoset.canExpunge = fsThumb.core.item.canExpunge;
                    infoset.canGetFileUrl = fsThumb.core.item.canGetFileUrl;
                    infoset.canEject = fsThumb.core.item.canEject;
                    infoset.canSetName = fsThumb.core.item.canSetName;
                    infoset.canOpen = fsThumb.core.item.canOpen;
                    infoset.canLock = fsThumb.core.item.canLock;
                    infoset.isExternalEditInProgress = fsThumb.core.item.isExterna
                    infoset.isLinkToContainer = fsThumb.core.item.isLinkToContain
                    infoset.isLockedByUser = fsThumb.core.item.isLockedByUser;
                    infoset.isPhysicalFile = fsThumb.core.item.isPhysicalFile;
                    infoset.isNeverWritable = fsThumb.core.item.isNeverWritable;
                    infoset.noWritePermission = fsThumb.core.item.noWritePermissi

                    if(this.privateData.node.getName().substr(0,10) != "SearchNod
                    infoset.cacheData.status = "good";
                }

                // Add any children of this node to the children infoset
                if(infosetName == "children" || infosetName == "all")
                {
                    var arr = currentNode.getChildren();
                    for(var j = 0;j < arr.length;j++)
                    {
                        infoset.addChild(sys.getPrefix() + arr[j].getPath());
                    }


                    infoset.cacheData.status = "good";
                }


                if(infosetName == "icon" || infosetName == "all")
                {
                    infoset.bitmap = fsThumb.core.icon.bitmap;
                    infoset.cacheData.status = "good";
                }


                if(infosetName == "thumbnail" || infosetName == "all")
```

```
832.                    {
833.                        infoset.thumbnail = fsThumb.core.thumbnail.thumbnail;
834.                        infoset.hasHighQualityThumbnail = fsThumb.core.thumbnail.hasH
835.                        infoset.cacheData.status = "good";
836.                    }
837.
838.
839.                    if(infosetName == "quickMetadata" || infosetName == "all")
840.                    {
841.                        infoset.cacheData.status = "good";
842.                    }
843.
844.
845.                    // The capabilities of a handled node
846.                    if(infosetName == "itemContent" || infosetName == "all")
847.                    {
848.                        infoset.fileFormat = fsThumb.core.itemContent.fileFormat;
849.                        infoset.mimeType = fsThumb.core.itemContent.mimeType;
850.                        infoset.pageCount = fsThumb.core.itemContent.pageCount;
851.                        infoset.tooltip = fsThumb.core.itemContent.tooltip;
852.                        infoset.dynamicMediaType = fsThumb.core.itemContent.dynamicMe
853.                        infoset.hasSubContainers = fsThumb.core.itemContent.hasSubCon
854.                        infoset.canDoCameraRaw = fsThumb.core.itemContent.canDoCamera
855.                        infoset.canRotate = fsThumb.core.itemContent.canRotate;
856.                        infoset.canGetThumbnail = fsThumb.core.itemContent.canGetThum
857.                        infoset.canGetPreview = fsThumb.core.itemContent.canGetPrevie
858.                        infoset.canGetQuickPreview = fsThumb.core.itemContent.canGetQ
859.                        infoset.canGetFullSize = fsThumb.core.itemContent.canGetFullS
860.                        infoset.canGetXmp = fsThumb.core.itemContent.canGetXmp;
861.                        infoset.canSetXmp = fsThumb.core.itemContent.canSetXmp;
862.                        infoset.canLabelOrRate = fsThumb.core.itemContent.canLabelOrR
863.
864.
865.                        infoset.cacheData.status = "good";
866.                    }
867.
868.                    if(infosetName == "cameraRaw" || infosetName == "all")
869.                    {
870.                        infoset.rawSupportType = fsThumb.core.cameraRaw.rawSupportTyp
871.                        infoset.cacheData.status = "good";
872.                    }
873.
874.                    if(infosetName == "preview" || infosetName == "all")
875.                    {
876.                        infoset.preview = fsThumb.core.preview.preview;
877.                        infoset.hasHighQualityPreview = fsThumb.core.preview.hasHighQ
878.                        infoset.cacheData.status = "good";
879.                    }
880.
881.                    if(infosetName == "metadata" || infosetName == "all")
882.                    {
883.                        if (fsThumb.hasMetadata){ infoset.metadata = fsThumb.core.meta
884.                        infoset.cacheData.status = "good";
885.                    }
886.
887.                    //if(infosetName == "customInfoset" || "all") //公式の罠
888.                    if(infosetName == "customInfoset" || infosetName == "all")
889.                    {
890.                        infoset.CustomString = currentNode.getCustomString();
891.                        infoset.CustomNumber = currentNode.getCustomNumber();
892.                        infoset.CustomDate = currentNode.getCustomDate();
```

```
                    infoset.CustomArray = currentNode.getCustomArray();
                    infoset.CustomBool = currentNode.getCustomBool();

                    //fontNameとfontFamily収集
                    var fontflag = true;
                    var fontAry = [];
                    for (i = 0; fontflag == true; i ++){
                        if (fsThumb.core.metadata.metadata == undefined || fsThuml
                            fontflag = false;
                        }else{
                            fontAry.push(fsThumb.core.metadata.metadata.read ("ht
                            };
                        };
                    if (fontAry.length) infoset.FontName = fontAry;

                    fontflag = true;
                    fontAry = [];
                    for (i = 0; fontflag == true; i ++){
                        if (fsThumb.core.metadata.metadata == undefined || fsThuml
                            fontflag = false;
                        }else{
                            fontAry.push(fsThumb.core.metadata.metadata.read ("ht
                            };
                        };
                    if (fontAry.length) infoset.FontFamily = fontAry;


                    this.privateData.cacheElement.customInfoset.cacheData.status =
                }
            }
            catch(error) { $.writeln("ERROR: " + infosetName + " : " + error); }

        } // end refreshInfoset


        /**
            Required.  Returns the cache status of a node data set.
            @param infosetName The name of the Infoset in which to check the cach
        */
        filterModel.getCacheStatus = function( infosetName)
        {
            if(typeof this.privateData.cacheElement == "undefined") { return "bad
            if(typeof this.privateData.cacheElement.infosetName== "undefined")
            {
                var ce = this.privateData.cacheElement;
                var info = ce[infosetName];
                var cd = info.cacheData;
                var stat = cd.status;
                return stat;
            }

            return this.privateData.cacheElement[infosetName].cacheData.status;
        }

        /**
            Return 'undefined' from this method. Any private data structures will
            cleaned automatically
        */
        filterModel.terminate = function() { return undefined; }
```

```
}


// Include the support classes
#include SDKSystem_MOD.jsx
#include SDKNode_MOD.jsx


/**
  "main program": construct an anonymous instance and run it
*/
if(typeof(CustomFilterExtensionHandler_unitTest) == "undefined") {
    new CustomFilterExtensionHandler().run();
}
function CustomFilterExtensionHandler() {


    $.level = 2; // Debugging level
    /**
      The context in which this sample can run.
      @type String
    */
    this.requiredContext = "\tNeed to be running in context of Bridge\n";
}


/**
 Functional part of this sample, creates several nodes of the new type, and one ma
 which stores the node-identifying prefix and root node.

 <p>Creates the ExtensionHandler object, whose definition includes
 the 'makeModel()' method, which Bridge uses to create the ExtensionModel object
 at node-display time.

 Registers the ExtensionHandler, associating it with the node-identifying prefix.
     @return True if the sample ran as expected, false otherwise
     @type Boolean
*/
CustomFilterExtensionHandler.prototype.run = function()
{
    if(!this.canRun()) { return false; }


    $.writeln("About to run CustomFilterExtensionHandler");


    // Make sure the XMP object is loaded and available so we can access a nodes n
    if( XmpScriptLib == undefined ) {
        if( Folder.fs == "Windows" ) {
            var pathToLib = Folder.startup.fsName + "/AdobeXMPScript.dll";
        } else {
            var pathToLib = Folder.startup.fsName + "/AdobeXMPScript.framework";
        }
        var libfile = new File( pathToLib );
        var XmpScriptLib = new ExternalObject("lib:" + pathToLib );
    }

    var rootFs = app.document.thumbnail.spec;

    // Create some nodes of the script-defined type
    var root = new SDKNode(rootFs.name, true);
```

```
48.
49.
50.     for (var i = 0;i < rootFs.getFiles ().length; i++){
51.         if (rootFs.getFiles ()[i].constructor.name == "Folder"){
52.             root.addNode (new SDKNode (rootFs.getFiles ()[i].name, true));
53.         }else{
54.             root.addNode (new SDKNode (rootFs.getFiles ()[i].name, false));
55.         };
56.     };
57.
58.
59.     // Create the manager object with the node-identifying prefix,
60.     // and add the root node
61.     var sys = new SDKSystem("bridge:cfNode:", root, rootFs.parent.fsName);
62.
63.
64.     // Create the node extension handler object
65.     var CFHandler = new ExtensionHandler("customFilterHandler");
66.
67.
68.     // Register the node handler and prefix    with the application
69.     app.registerExtension(CFHandler);
70.     app.registerPrefix( "bridge:cfNode:", CFHandler);
71.
72.
73.     /**
74.         Required. Returns a canonical URI for the path.  If the path does not con
75.         registered prefix, the this method must append it to the path and return
76.         bridge:beNode:/CSRoot
77.         @param path The path of the node as a string
78.     */
79.     CFHandler.getBridgeUriForPath = function(path) { return path; }
80.
81.
82.     /**
83.         Extends the handler with a helper function so all the thumbnails can
84.         be collected for the search
85.         @param kids An array of SDK nodes.
86.     */
87.     ExtensionHandler.prototype.getThumbsToSearch = function(kids)
88.     {
89.         for(var h = 0;h < kids.length;h++)
90.         {
91.             if(kids[h].isContainer()) { this.getThumbsToSearch(kids[h].getChildre
92.             CFHandler.thumbsToSearch.push(kids[h]);
93.         }
94.     }
95.
96.
97.     /**
98.         Search implementation: Performs a search and returns the search result,
99.         a container Thumbnail object that contains the matching nodes.
100.        The passed 'scope' is the Thumbnail object that the search starts from, a
101.        'spec' is the SearchSpecification object containing all search details as
102.        set by the user in the Find dialog.
103.
104.        @param scope  The Thumbnail to start searching from
105.        @param spec The SearchSpecification details
106.    */
107.    CFHandler.getBridgeUriForSearch = function(scope, spec)
108.    {
```

```
109.        // Create a search results node - give it a unique name for each search c
110.        var randomnumber=Math.floor(Math.random()*101);
111.        var d = new Date();
112.        var postfix = d.getSeconds() + randomnumber;
113.
114.
115.        var sNode = new SDKNode("SearchNode" + postfix, true)
116.        sNode.children = new Array();
117.        sys.addSearchNode(sNode); // This overwrites any previous search node obj
118.
119.
120.        var sThumb = new Thumbnail("bridge:cfNode:/SearchNode" + postfix);
121.        sThumb.core.children.cacheData.status = "bad";
122.
123.
124.        // Get the scope specifiers (whether to search subfolders)
125.        var searchScopes = spec.scopeSpecifiers;
126.
127.
128.        //  Create an array to hold the thumbnails to   be checked against search
129.        CFHandler.thumbsToSearch = new Array();
130.
131.
132.        // This is the node from which to start
133.        var scopeNode = sys.getNodeFromPath(scope.uri);
134.
135.
136.        var children;
137.        // Check if the search is to include subfolders
138.        if(searchScopes[0] == "DoSubFolders")
139.        {
140.            // Get the node children and find all the thumbnaiils we need to sear
141.            this.getThumbsToSearch(scopeNode.getChildren());
142.
143.
144.            children = CFHandler.thumbsToSearch;
145.        }
146.        else
147.        {
148.            children = scopeNode.getChildren();
149.        }
150.
151.        // Get the conditions of the search, as specified by the user
152.        var conditions = spec.conditionList;
153.
154.
155.        if(spec.conjunction == "or")    // This search only supports an "OR" conju
156.        {
157.            // Check each thumbnail/node and see if the criteria match
158.            for(var i = 0;i < children.length;i++)
159.            {
160.                var currentNode = children[i];
161.
162.                for(j = 0;j < conditions.length;j++)
163.                {
164.                    var currentCon = conditions[j]
165.                    var searchField = currentCon.searchField;
166.                    var operand = currentCon.operand;
167.
168.
169.                    // Get the values for the field/data we are searching against
```

```
                    switch(searchField)
                    {
                        var thumbFieldValue;
                        case "CustomString":
                            thumbFieldValue = currentNode.getCustomString();
                            break;
                        case "CustomNumber":
                            thumbFieldValue = currentNode.getCustomNumber();
                            break;
                        case "CustomBool":
                            thumbFieldValue = currentNode.getCustomBool();
                            break;
                        case "name":
                            thumbFieldValue = currentNode.getName();
                            break;
                    }


                    // Check if anything matches the criteria
                    // This is an OR search, so when a node hits any match,
                    // add it to the children of the search container
                    switch(currentCon.operatorType)
                    {
                        case "equals":
                            // catches boolean values and strings
                            if(thumbFieldValue == operand || thumbFieldValue.toSt
                            break;
                        case "lessThanOrEqual":
                            if(thumbFieldValue <= operand) { addNodeToSearchResul
                            break;
                        case "greaterThanOrEqual":
                            if(thumbFieldValue >= operand) { addNodeToSearchResul
                            break;
                        case "endsWith":
                            var s = thumbFieldValue.substr((thumbFieldValue.lengt
                            if(s == operand) { addNodeToSearchResult(currentNode)
                            break;
                        default:
                    } // end switch
                } // end for condtions
            } // end for children
        }
        else if(spec.conjunction == "and")  // This search only supports "OR" sea
        {
            alert("SDK: CustomFilterExtensionHandler:\nSample does not show this
        }


        // Helper function to check for the existence of a node before adding it
        function addNodeToSearchResult(node)
        {
            var matches = sNode.getChildren();
            var addNode = true;
            for(var i = 0;i < matches.length;i++)
            {
                if(matches[i].getPath() == node.getPath())
                {
                    addNode = false;
                    break;
                }
            }
```

```
231.
232.
233.                 if(addNode) { sNode.addNode(node); }
234.             }
235.
236.
237.             // Return the URI to the container (a Thumbnail object) holding the searcl
238.             return sThumb.uri;
239.         }
240.
241.
242.         /**
243.             Creates an operator that performs the long running operation.  In this cas
244.             made to the this nodes XMP data.
245.
246.             @param targets
247.             @param xmpPackets
248.             @param timeoutInMs Optional. The number of microseconds after wich to abor
249.             @param showUi Optional.  Wether to show the UI or not
250.             @param message Optional. A string to display via the progress UI
251.         */
252.         CFHandler.setXMP = function(targets, xmpPackets, timeoutInMs, showUi, message
253.         {
254.             var thumbs = targets;
255.             var xmpData = xmpPackets;
256.
257.             var op = new ProgressOperator("progress", thumbs);
258.
259.             op.start = function()
260.             {
261.                 op.procStat = "notStarted";
262.
263.                 op.opStat = "incomplete";
264.                 app.operationChanged(this);
265.
266.                 op.procStat = "inProgress";
267.
268.
269.                 this.processedNodeCount = 0;
270.
271.
272.                 app.operationChanged(this);
273.
274.                 op.percComp = 1;
275.
276.                 for(var i = 0;i < thumbs.length;i++)
277.                 {
278.
279.
280.                     // Only added so we can see the ProgressOperator UI in action
281.                     for(var j = 0;j < 10;j++)
282.                     {
283.                         $.sleep(50);
284.                     }
285.                     op.percComp += (100 / thumbs.length);
286.                     app.operationChanged( this );
287.
288.                     var t = thumbs[i];
289.                     t.core.metadata.metadata = new Metadata(xmpData[i]);
290.                     t.model.privateData.node.setMetadata(new Metadata(xmpData[i]));
291.
```

```
                this.processedNodeCount++;
            }


            op.percComp = 100;
            op.opStat = "succeeded";
            op.procStat = "completed";

            app.operationChanged( this );
            return;
        }


        op.stop = function() { return; }

        op.getConflictInfo = function() { return "unknown"; }

        op.getOperationStatus = function() { return op.opStat; }

        op.getPercentageComplete = function() { return op.percComp; }

        op.getProcessedNodeCount = function() { return this.processedNodeCount; }

        op.getProcessingStatus = function() { return op.procStat; }

        op.getProgressMessage = function() { return "Setting XMP on selected files

        op.getTotalBytesTransferred = function() { return 0; }

        op.getTotalNodeCount = function() { return thumbs.length; }

        op.getType = function() { return "progress"; }

        op.getUIPolicy = function() { return this.showUi; }

        op.resume = function() { return false; }

        op.resolveConflict = function() { }


        app.enqueueOperation(op);
        //return op;


    }


    var wrapperObject = this;
    /**
        Required. Creates and returns an ExtensionModel instance.  Called each ti
        is displayed.  The model must also implement any methods that handle any
        functionality, such as renaming a node.
        @return An ExtensionModel for this node
        @type ExtensionModel
    */
    CFHandler.makeModel = function(path)
    {
            // Add a a property to store the manual sort
        ExtensionModel.prototype.sortXML = "";
        filterModel = new ExtensionModel(path);
        filterModel.path = path;
```

```
353.          this.path = path;
354.          wrapperObject.initModel(filterModel, sys);
355.
356.          return filterModel;
357.
358.
359.      } // end makeModel
360.
361.
362.      // Create the InfosetMemberDescription objects
363.      var csDescriptions = new Array();
364.      csDescriptions[0] = new InfosetMemberDescription( "CustomString", "String" );
365.      csDescriptions[1] = new InfosetMemberDescription( "CustomNumber", "Integer" )
366.      csDescriptions[2] = new InfosetMemberDescription( "CustomDate", "String" );
367.      csDescriptions[3] = new InfosetMemberDescription( "CustomArray", "Array of St
368.      csDescriptions[4] = new InfosetMemberDescription( "CustomBool", "Boolean" );
369.
370.      csDescriptions[5] = new InfosetMemberDescription( "FontName", "Array of Strin
371.      csDescriptions[6] = new InfosetMemberDescription( "FontFamily", "Array of Str
372.
373.
374.      // Create the infoset
375.      var customInfoset = new Infoset("customInfoset", csDescriptions);
376.
377.
378.      // Register the infoset
379.      app.registerInfoset(CFHandler, customInfoset);
380.
381.      // Event handler displays the Custom Infoset Data palette for handled thumbna
382.      onEvent = function(event)
383.      {
384.          if(event.object instanceof Document)
385.          {
386.              if(event.type == "selectionsChanged")
387.              {
388.                  if(event.object.selections.length > 0)
389.                  {
390.                      var thumb = event.object.selections[0];
391.                      if(thumb.customFilterHandler)
392.                      {
393.                          stringText.text = thumb.customFilterHandler.customInfoset
394.                          numbersText.text = thumb.customFilterHandler.customInfose
395.                          dateText.text = thumb.customFilterHandler.customInfoset.Cu
396.                          arrStringText.text = thumb.customFilterHandler.customInfo
397.                          boolList.selection = (thumb.customFilterHandler.customInf
398.                          customSearchDataPalette.content.visible = true;
399.                      }
400.                      else
401.                      {
402.                          customSearchDataPalette.content.visible= false;
403.                      }
404.                  }
405.                  else
406.                  {
407.                      customSearchDataPalette.content.visible= false;
408.                  }
409.              }
410.          }
411.      }
412.
413.
```

```
414.      // Register the event handler
415.      app.eventHandlers.push({handler: onEvent});
416.
417.
418.      // Create  a palette to display the custom data
419.      var customSearchDataPalette = new TabbedPalette(app.document, "SDK: Custom So
420.      var win = customSearchDataPalette.content;
421.
422.
423.      var stringLabel = win.add("statictext", [5, 9, 115, 25], "Custom String");
424.      var stringText = win.add("edittext", [120, 5, 320, 25], "");
425.      stringText.enabled = false;
426.
427.      var numbersLabel = win.add("statictext", [5, 39, 115, 55], "Custom Number");
428.      var numbersText = win.add("edittext", [120, 35, 320, 55], "");
429.      numbersText.enabled = false;
430.
431.
432.      var dateLabel = win.add("statictext", [5, 69, 115, 85], "Custom Date");
433.      var dateText = win.add("edittext", [120, 65, 320, 85], "");
434.      dateText.enabled = false;
435.
436.
437.      var arrLabel = win.add("statictext", [5, 99, 115, 115], "Custom Array");
438.      var arrStringText = win.add("edittext", [120, 95, 320, 115], "");
439.      arrStringText.enabled = false;
440.
441.      var boolLabel = win.add("statictext", [5, 129, 115, 145], "Custom Boolean");
442.      var boolList = win.add("dropdownlist", [120, 125, 320, 145]);
443.      var tItem = boolList.add("item", "True");
444.      tItem.val = true;
445.      var fItem = boolList.add("item", "False");
446.      fItem.val = false;
447.      boolList.enabled = false;
448.
449.      // Hide the palette until a handled node is selected
450.      customSearchDataPalette.content.visible = false;
451.
452.
453.
454.      // create a thumbnail and add it to the Favorites panel
455.      var t = new Thumbnail(sys.getPrefix() + sys.fsRoot + sys.root.path);
456.      app.favorites.insert(t);
457.      t.refresh();
458.
459.
460.      $.writeln("CustomFilterExtensionHandler initialized successfully");
461.      return true;
462.  }
463.
464.
465.
466.
467.  /**
468.    Determines whether sample can be run given current context.  The sample
469.    fails if these preconditions are not met:
470.    <ul>
471.    <li> Bridge must be running
472.    <li> Resource files must be available
473.    </ul>
474.    @return True if this sample can run, false otherwise
```

```
  @type Boolean
*/
CustomFilterExtensionHandler.prototype.canRun = function() {
    if(BridgeTalk.appName == "bridge") {

        for(index in app.extensions)
        {
            if(app.extensions[index].name == "customFilterHandler")
            {
                $.writeln("ERROR: Cannont run twice! You must restart Bridge");
                return false;
            }
        }
        return true;
    }
    $.writeln("ERROR:: Cannot run CustomFilterExtensionHandler");
    $.writeln(this.requiredContext);
    return false;
}


/**
    Initialize the model and define the methods that this model supports
    @param sortModel The model for the handled nodes
    @param sys The manager object for the handled nodes
*/
CustomFilterExtensionHandler.prototype.initModel = function(filterModel, sys)
{
        /**
            Implements the customization of the Find dialog for the custom search
            Creates and returns a SearchDefinition object used for the customizat

            @return A customized SearchDefinition
            @type SearchDefinition
        */
        filterModel.getSearchDefinition = function()
        {


            // Criteria section
            // Add values to the search field (left box) of the Criteria section
            var searchField = "CustomString";
            var opType = "string";
            var searchFieldDisplay = "Custom String";


            var searchCriteria1 = new SearchCriteria(searchField, opType, searchF


            // Restrict values for the operator (middle box) of the Criteria sect
            searchCriteria1.operatorTypesToDisable = ["exists", "equals", "doesNo
                "lessThanOrEqual", "greaterThanOrEqual", "contains",
                "doesNotContain", "startsWith", "greater"];

            searchField = "CustomNumber";
            opType = "float";
            searchFieldDisplay = "Custom Number";
            var searchCriteria2 = new SearchCriteria(searchField, opType, searchF
            searchCriteria2.operatorTypesToDisable = ["exists", "doesNotExist", "
                "contains",     "doesNotContain", "endsWith",
                "startsWith", "greater"];
```

```
536.
537.            searchField = "name";
538.            opType = "string";
539.            searchFieldDisplay = "Name";
540.            var searchCriteria3 = new SearchCriteria(searchField, opType, searchF
541.            searchCriteria3.operatorTypesToDisable = ["exists", "doesNotExist", "
542.                "lessThanOrEqual", "greaterThanOrEqual", "contains",
543.                "doesNotContain", "endsWith", "startsWith", "greater"];
544.
545.
546.            searchField = "CustomBool";
547.            opType = "boolean";
548.            searchFieldDisplay = "Custom Boolean";
549.
550.            // Create some Operand objects
551.            // these populate the comparison-value choices in the far right box o
552.            var op1 = new Operand("true", "A true value");
553.            var op2 = new Operand("false", "A false value");
554.            var myOperators = [op1, op2];
555.
556.
557.            var searchCriteria4 = new SearchCriteria(searchField, opType, searchF
558.            searchCriteria4.operatorTypesToDisable = ["exists", "doesNotExist", "
559.                "lessThanOrEqual", "greaterThanOrEqual", "contains",
560.                "doesNotContain", "endsWith", "startsWith", "greater"];
561.
562.
563.            var critList = [searchCriteria1, searchCriteria2, searchCriteria3, sea
564.
565.
566.            // Results section
567.            //var rank1 = new Rank("RankValueName", "RankDisplayName");
568.            //var rank2 = new Rank("RankValueName", "RankDisplayName");
569.
570.
571.            var ranksArray = [];  // These are optional, but empty arrays must be
572.
573.
574.            // create some scopeSpecifier objects
575.            var scope1 = new Scope( "DoSubFolders", "SDKSearch: Search subfolders
576.            var scope2  = new Scope("DoHidden", "SDKSearch: Search hidden files")
577.            // the scopeSpecifiers to an Array
578.            var scopesArray = [scope1, scope2]; // These are documented as optiona
579.
580.
581.            var searchDef = new SearchDefinition(critList, 10, ranksArray, scopesA
582.            return searchDef;
583.        }
584.
585.            // Renames a node
586.        filterModel.setName = function(name)
587.        {
588.
589.
590.            // Get the parent of the node
591.            var parent = this.privateData.node.getParent();
592.            var update = new Thumbnail(sys.getPrefix() + parent.getPath());
593.
594.
595.            // Set the cache status to bad as we want a refresh to occur
596.            update.core.children.cacheData.status = "bad";
```

```
597.
598.
599.                // Change the name of the node
600.                this.privateData.node.setName(name);
601.
602.
603.                // If the name has been changed whilst the node is viewed in a search
604.                if(app.document.thumbnail.name == "SearchNode") { app.document.thumbna
605.
606.
607.                var newURI = sys.getPrefix() + this.privateData.node.getPath();
608.                return newURI;
609.            } // end setName
610.
611.            // Creates a new container of the script defined type
612.            // Must return an operator as this is a long running operation.
613.            // Creates a new container of the script defined type
614.            // Must return an operator as this is a long running operation.
615.            filterModel.createNewContainer = function(name)
616.            {
617.                var newNode = new SDKNode(name, true)
618.                this.privateData.node.addNode(newNode);
619.
620.
621.                // Set the cache status to bad as we want a refresh to occur
622.                this.privateData.cacheElement.children.cacheData.status = "bad";
623.                //this.refreshInfoset("all");
624.
625.
626.                return sys.getPrefix() + newNode.getPath();
627.            } // end createNewContainer
628.
629.            /**
630.                Checks whether this node requires authentication.
631.                @return True if this node requires authentication, false otherwise
632.                @type Boolean
633.            */
634.            filterModel.needAuthentication = function() { return false; }
635.
636.
637.            /**
638.                Required.  Carries out any necessary authentication.
639.            */
640.            filterModel.authenticate = function() { return; }
641.
642.            /**
643.                Required,  Checks if this node is a valid node.
644.                @return True if this node is valid, false otherwise
645.                @type Boolean
646.            */
647.            filterModel.exists = function() { return true; }
648.
649.
650.            /**
651.                Required.  Called after the model is created, sets up any necessary da
652.                that may be needed.
653.            */
654.            filterModel.initialize = function() { this.privateData.node = sys.getNodeI
655.
656.            /**
657.                Required.  Stores a back pointer to the cache that contains this model
```

```
658.            @param cacheEelement The CacheElement that contains this model.
659.        */
660.        filterModel.registerInterest = function(cacheElement) { this.privateData.
661.
662.        /**
663.            Required. Removes the association between the CacheElement and
664.            this model instance.
665.        */
666.        filterModel.unregisterInterest = function() { this.privateData.cacheEleme
667.
668.        /**
669.            Adds a custom filter criteria to Bridge.  Single FilterDescription is
670.            This allows hanlded nodes to be filtered based on a string value stor
671.        */
672.        filterModel.getFilterCriteria = function()
673.        {
674.            var filter1 = new FilterDescription ("FontName", "使用フォント", "strin
675.            defFilters.push (filter1);
676.
677.            var filter2 = new FilterDescription ("FontFamily", "フォントファミリ",
678.            defFilters.push (filter2);
679.
680.            return defFilters;
681.        }
682.
683.
684.        /**
685.            Adds custom SortCriterion to Bridge for handled nodes.
686.            Three new sort options are added to Bridge's default sort criterion.
687.            The third uses a custom namespace and custom XMP data.
688.            @return An array of SortCriterion objects
689.        */
690.        filterModel.getSortCriteria = function()
691.        {
692.            var dsc = app.defaultSortCriteria;
693.            return dsc;
694.        }
695.
696.        /**
697.            Returns the parent of this node
698.        */
699.        filterModel.getParent = function()
700.        {
701.            var parent = this.privateData.node.getParent();
702.            if(parent != undefined) { return (sys.getPrefix() + parent.getPath())
703.        }
704.
705.        filterModel.supportsUserSortOrder = function() { return true; }
706.
707.
708.        /**
709.            Stores the custom sort for this container
710.        */
711.        filterModel.setUserSortOrder = function(inXML) { this.sortXML = inXML; }
712.
713.        /**
714.            Return an XML string that gives the custom sort of the children
715.            of this container.
716.        */
717.        filterModel.getUserSortOrder = function()
718.        {
```

```
719.            if(this.sortXML == "")
720.            {
721.                var xml = "<?xml version='1.0' encoding='UTF-8' standalone='yes'
722.                var children = this.privateData.node.getChildren();
723.                for(var i = 0;i < children.length;i++)
724.                {
725.                    // Use 00000000000000 if creattiondate is not supported
726.                    xml += "<item key='" + children[i].getName() + "0000000000000
727.                }
728.                xml += "</files></dirinfo>";
729.                this.sortXML = xml;
730.            }
731.
732.            return this.sortXML;
733.        }
734.
735.        filterModel.addToDrag = function(pointerToOsDragObject) { return true; }
736.
737.        filterModel.wouldAcceptDrop = function(type, sources, osDragRef) { return
738.
739.        /**
740.            Required.  Updates the core node data and any custom node data
741.            for this node.  Always set the CacheData.status after updating the no
742.            @param infosetName The name of the Infoset that is being updated.
743.        */
744.        filterModel.refreshInfoset = function(infosetName)
745.        {
746.            var infoset = this.privateData.cacheElement[infosetName];
747.            var currentNode = this.privateData.node;
748.            var fsThumb = new Thumbnail ("bridge:fs:" + currentNode.getFileUrl(sy
749.            try
750.            {
751.                if(infosetName == "immediate" || infosetName == "all")
752.                {
753.                    if(this.privateData.node.getName().substr(0,10) == "SearchNod
754.                    {
755.                        infoset.name = "SearchNode";
756.                    }
757.                    else
758.                    {
759.                        infoset.name = fsThumb.core.immediate.name;
760.
761.                        //値を渡した後のどこかの処理で配列の順序が逆転するため、表示さ
762.                        //逆順で渡して対処で正しいのかどうか。そもそもどういう意図の配
763.                        //infoset中の配列は全部順序が逆転する？
764.                        infoset.size = fsThumb.core.immediate.size.reverse();
765.                        infoset.creationDate = fsThumb.core.immediate.creationDat
766.                        infoset.modificationDate = fsThumb.core.immediate.modific
767.
768.
769.                        infoset.isHidden = fsThumb.core.immediate.isHidden;
770.                        infoset.isLink = fsThumb.core.immediate.isLink;
771.                        infoset.isPackage = fsThumb.core.immediate.isPackage;
772.                        infoset.isDeleted = fsThumb.core.immediate.isDeleted;
773.                        infoset.isApplication = fsThumb.core.immediate.isApplicat
774.                        infoset.isContainer = fsThumb.core.immediate.isContainer;
775.                        infoset.isAssortment = fsThumb.core.immediate.isAssortmen
776.                        infoset.sortIndex = fsThumb.core.immediate.sortIndex;
777.                        infoset.fileUrl = fsThumb.core.immediate.fileUrl;
778.                        infoset.displayPath = fsThumb.core.immediate.displayPath;
779.                    }
```

```
780.                    infoset.cacheData.status = "good";
781.                }
782.
783.
784.                // The capabilities of a handled node
785.                if(infosetName == "item" || infosetName == "all")
786.                {
787.                    infoset.canCreateNewContainer = fsThumb.core.item.canCreateNe
788.                    infoset.canCreateNewLink = fsThumb.core.item.canCreateNewLink
789.                    infoset.canDelete = fsThumb.core.item.canDelete;
790.                    infoset.canDuplicate = fsThumb.core.item.canDuplicate;
791.                    infoset.canBeDragSource = fsThumb.core.item.canBeDragSource;
792.                    infoset.canBeDropTarget = fsThumb.core.item.canBeDropTarget;
793.                    infoset.canExpunge = fsThumb.core.item.canExpunge;
794.                    infoset.canGetFileUrl = fsThumb.core.item.canGetFileUrl;
795.                    infoset.canEject = fsThumb.core.item.canEject;
796.                    infoset.canSetName = fsThumb.core.item.canSetName;
797.                    infoset.canOpen = fsThumb.core.item.canOpen;
798.                    infoset.canLock = fsThumb.core.item.canLock;
799.                    infoset.isExternalEditInProgress = fsThumb.core.item.isExterna
800.                    infoset.isLinkToContainer = fsThumb.core.item.isLinkToContain
801.                    infoset.isLockedByUser = fsThumb.core.item.isLockedByUser;
802.                    infoset.isPhysicalFile = fsThumb.core.item.isPhysicalFile;
803.                    infoset.isNeverWritable = fsThumb.core.item.isNeverWritable;
804.                    infoset.noWritePermission = fsThumb.core.item.noWritePermissi
805.
806.                    if(this.privateData.node.getName().substr(0,10) != "SearchNod
807.                    infoset.cacheData.status = "good";
808.                }
809.
810.                // Add any children of this node to the children infoset
811.                if(infosetName == "children" || infosetName == "all")
812.                {
813.                    var arr = currentNode.getChildren();
814.                    for(var j = 0;j < arr.length;j++)
815.                    {
816.                        infoset.addChild(sys.getPrefix() + arr[j].getPath());
817.                    }
818.
819.
820.                    infoset.cacheData.status = "good";
821.                }
822.
823.
824.                if(infosetName == "icon" || infosetName == "all")
825.                {
826.                    infoset.bitmap = fsThumb.core.icon.bitmap;
827.                    infoset.cacheData.status = "good";
828.                }
829.
830.
831.                if(infosetName == "thumbnail" || infosetName == "all")
832.                {
833.                    infoset.thumbnail = fsThumb.core.thumbnail.thumbnail;
834.                    infoset.hasHighQualityThumbnail = fsThumb.core.thumbnail.hasH
835.                    infoset.cacheData.status = "good";
836.                }
837.
838.
839.                if(infosetName == "quickMetadata" || infosetName == "all")
840.                {
```

```
                    infoset.cacheData.status = "good";
                }

                // The capabilities of a handled node
                if(infosetName == "itemContent" || infosetName == "all")
                {
                    infoset.fileFormat = fsThumb.core.itemContent.fileFormat;
                    infoset.mimeType = fsThumb.core.itemContent.mimeType;
                    infoset.pageCount = fsThumb.core.itemContent.pageCount;
                    infoset.tooltip = fsThumb.core.itemContent.tooltip;
                    infoset.dynamicMediaType = fsThumb.core.itemContent.dynamicMe
                    infoset.hasSubContainers = fsThumb.core.itemContent.hasSubCon
                    infoset.canDoCameraRaw = fsThumb.core.itemContent.canDoCamera
                    infoset.canRotate = fsThumb.core.itemContent.canRotate;
                    infoset.canGetThumbnail = fsThumb.core.itemContent.canGetThum
                    infoset.canGetPreview = fsThumb.core.itemContent.canGetPrevie
                    infoset.canGetQuickPreview = fsThumb.core.itemContent.canGetQ
                    infoset.canGetFullSize = fsThumb.core.itemContent.canGetFullS
                    infoset.canGetXmp = fsThumb.core.itemContent.canGetXmp;
                    infoset.canSetXmp = fsThumb.core.itemContent.canSetXmp;
                    infoset.canLabelOrRate = fsThumb.core.itemContent.canLabelOrR


                    infoset.cacheData.status = "good";
                }

                if(infosetName == "cameraRaw" || infosetName == "all")
                {
                    infoset.rawSupportType = fsThumb.core.cameraRaw.rawSupportTyp
                    infoset.cacheData.status = "good";
                }

                if(infosetName == "preview" || infosetName == "all")
                {
                    infoset.preview = fsThumb.core.preview.preview;
                    infoset.hasHighQualityPreview = fsThumb.core.preview.hasHighQ
                    infoset.cacheData.status = "good";
                }

                if(infosetName == "metadata" || infosetName == "all")
                {
                    if (fsThumb.hasMetadata){ infoset.metadata = fsThumb.core.meta
                    infoset.cacheData.status = "good";
                }

                //if(infosetName == "customInfoset" || "all") //公式の罠
                if(infosetName == "customInfoset" || infosetName == "all")
                {
                    infoset.CustomString = currentNode.getCustomString();
                    infoset.CustomNumber = currentNode.getCustomNumber();
                    infoset.CustomDate = currentNode.getCustomDate();
                    infoset.CustomArray = currentNode.getCustomArray();
                    infoset.CustomBool = currentNode.getCustomBool();

                    //fontNameとfontFamily収集
                    var fontflag = true;
                    var fontAry = [];
                    for (i = 0; fontflag == true; i ++){
                        if (fsThumb.core.metadata.metadata == undefined || fsThum
                            fontflag = false;
```

```
                        }else{
                            fontAry.push(fsThumb.core.metadata.metadata.read ("ht
                            };
                        };
                    if (fontAry.length) infoset.FontName = fontAry;

                    fontflag = true;
                    fontAry = [];
                    for (i = 0; fontflag == true; i ++){
                        if (fsThumb.core.metadata.metadata == undefined || fsThuml
                            fontflag = false;
                        }else{
                            fontAry.push(fsThumb.core.metadata.metadata.read ("ht
                            };
                        };
                    if (fontAry.length) infoset.FontFamily = fontAry;


                    this.privateData.cacheElement.customInfoset.cacheData.status =
                }
            }
            catch(error) { $.writeln("ERROR: " + infosetName + " : " + error); }

        } // end refreshInfoset


        /**
            Required.  Returns the cache status of a node data set.
            @param infosetName The name of the Infoset in which to check the cache
        */
        filterModel.getCacheStatus = function( infosetName)
        {
            if(typeof this.privateData.cacheElement == "undefined") { return "bad
            if(typeof this.privateData.cacheElement.infosetName== "undefined")
            {
                var ce = this.privateData.cacheElement;
                var info = ce[infosetName];
                var cd = info.cacheData;
                var stat = cd.status;
                return stat;
            }


            return this.privateData.cacheElement[infosetName].cacheData.status;
        }

        /**
            Return 'undefined' from this method. Any private data structures will
            cleaned automatically
        */
        filterModel.terminate = function() { return undefined; }
}


// Include the support classes
#include SDKSystem_MOD.jsx
#include SDKNode_MOD.jsx


/**
```

```
963.   "main program": construct an anonymous instance and run it
964. */
965. if(typeof(CustomFilterExtensionHandler_unitTest) == "undefined") {
966.     new CustomFilterExtensionHandler().run();
967. }
```

SDKSystem_MOD.jsx

```
01. function SDKSystem(prefix, root, rootFsPath)
02. {
03.     this.prefix = prefix;
04.     this.root = root;
05.     this.searchNode = undefined;
06.     this.fsRoot = rootFsPath.replace (/\\/gim, "/");
07. }
08. 
09. SDKSystem.prototype.addRoot = function(root) { this.root = root; }
10. SDKSystem.prototype.getPrefix = function(){ return this.prefix;}
11. SDKSystem.prototype.getNodeFromPath = function(path)
12. {
13.     var tmp = path.split("/");
14.     var name = tmp[tmp.length-1];
15.     if(name.substring(0, 6) == "Search") { return this.searchNode; }
16.     if(name == this.root.getName()) return this.root;
17.     var retval = this.root.findChildNodeFromPath(path);
18.     return retval;
19. }
20. 
21. SDKSystem.prototype.addSearchNode = function(n) { this.searchNode = n; }
```


```
01. function SDKSystem(prefix, root, rootFsPath)
02. {
03.     this.prefix = prefix;
04.     this.root = root;
05.     this.searchNode = undefined;
06.     this.fsRoot = rootFsPath.replace (/\\/gim, "/");
07. }
08. 
09. SDKSystem.prototype.addRoot = function(root) { this.root = root; }
10. SDKSystem.prototype.getPrefix = function(){ return this.prefix;}
11. SDKSystem.prototype.getNodeFromPath = function(path)
12. {
13.     var tmp = path.split("/");
14.     var name = tmp[tmp.length-1];
15.     if(name.substring(0, 6) == "Search") { return this.searchNode; }
16.     if(name == this.root.getName()) return this.root;
17.     var retval = this.root.findChildNodeFromPath(path);
18.     return retval;
19. }
20. 
21. SDKSystem.prototype.addSearchNode = function(n) { this.searchNode = n; }
```


SDKNode_MOD.jsx

```
01. function SDKNode(name, iscontainer)
02. {
03.     this.children = new Array();
04.     this.childrenNames = {};
05.     this.path = "/" + name;
06.     this.name = name;
07.     this.parent = undefined;
```

```
    this.container = iscontainer;
    this.iconPath = new File($.fileName).parent.fsName.replace (/\\/gim, "/") + "/
    this.metadata = undefined;
    this.setCustomInfosetData();
}
SDKNode.prototype.addNode = function(node)
{
    if(!this.getChildNode(node.getName()))
    {
        if(this.name.substr(0,10) != "SearchNode")
        {
            node.parent = this;
        }
        this.childrenNames[node.name] = node;
        this.children.push(node);
        return true;
    }
    return false;
}
SDKNode.prototype.getFirstChildNode = function() { return this.children[0]; }
SDKNode.prototype.getChildNode = function(name)
{
    SDKNode.counter++;
    return this.childrenNames[name];
}
SDKNode.prototype.findChildNode = function(name)
{
    var found = this.getChildNode(name);
    if(found instanceof SDKNode)
    {
        return found;
    }
    else
    {
        for(index in this.children)
        {
            found = this.children[index].findChildNode(name);
            if(found instanceof SDKNode)
            {
                return found;
            }
        }
    }
    return false;
}
SDKNode.prototype.findChildNodeFromPath = function(uri)
{
    var pathParts = uri.split("/");
    var currentNode = this;
    var searching = true;
    var pos = 1;
    while(searching)
    {
        if(typeof pathParts[pos+2] != "undefined")
        {
            pos++;
            currentNode = currentNode.getChildNode(pathParts[pos]);
        }
        else
        {
            currentNode = currentNode.getChildNode(pathParts[pos+1]);
```

```
            searching = false;
        }
    }
    if(currentNode instanceof Object && currentNode != false)
    {
        return currentNode;
    }
    else
    {
        return false;
    }
}
SDKNode.prototype.getChildren = function(){ return this.children; }
SDKNode.prototype.getChildrenNames = function(){ return this.childrenNames; }
SDKNode.prototype.getPath = function()
{
    var path = this.path;
    var parent = this.parent;
    while(parent)
    {
        var fullPath = parent.path + path;
        parent = parent.getParent();
        path = fullPath;
    }
    return path;
}
// For debugging only
SDKNode.prototype.listChildren = function()
{
    var kids = this.getChildren();
    for(var i = 0;i < kids.length;i++)
    {
        $.writeln(kids[i].getPath());
        kids[i].listChildren();
    }
}
SDKNode.prototype.hasChildren = function() { return this.children.length > 0; }
SDKNode.prototype.getParent = function() { return this.parent; }
SDKNode.prototype.removeNode = function(node)
{
    var name = node.getName();
    delete this.childrenNames[name];
    var tmp = new Array();
    for(var i = 0;i < this.children.length;i++)
    {
        if(this.children[i].name != name) { tmp.push(this.children[i]); }
    }
    this.children = tmp;
}
SDKNode.prototype.updateNode = function(node, oldName)
{
    delete this.childrenNames[oldName];
    this.childrenNames[node.getName()] = node;
}
SDKNode.prototype.cloneNode = function()
{
    var c = new Object();
    for( var i in this) { c[i] = this[i] }
    c.parent = undefined;
    return c;
}
```

```
SDKNode.prototype.setName = function(name)
{
    var oldName = this.name;
    this.name = name;
    this.path = "/" + name;
    if(this.parent != undefined)
    {
        this.parent.updateNode(this, oldName);
    }
}
SDKNode.prototype.isContainer = function() { return this.container; }
SDKNode.prototype.getName = function() { return this.name; }
SDKNode.prototype.getFileUrl = function(sys) { return "file:///" + sys.fsRoot + tl
SDKNode.prototype.getIcon = function()
{
    if(this.isContainer())
    {
        return new BitmapData(this.iconPath + "/BEHFolderIcon32.png");
    }
    else
    {
        return new BitmapData(this.iconPath + "/BEHFileIcon32.png");
    }
}
SDKNode.prototype.getThumb = function()
{
    if(this.isContainer())
    {
        return new BitmapData(this.iconPath + "/BEHFolderIcon128.png");
    }
    else
    {
        return new BitmapData(this.iconPath + "/BEHFileIcon128.png");
    }
}
SDKNode.prototype.getPreview = function()
{
    if(this.isContainer())
    {
        return new BitmapData(this.iconPath + "/BEHFolderIcon512.png");
    }
    else
    {
        return new BitmapData(this.iconPath + "/BEHFileIcon512.png");
    }
}
SDKNode.prototype.getMetadata = function()
{
    if(this.metadata == undefined)
    {
        var xmpMeta = new XMPMeta( );
        var d = new Date()
        d.setFullYear(2007,0,1);
        d.setHours(1, 0, 0, 0);
        var xmptoday = new XMPDateTime(d);
        xmptoday.convertToLocalTime();

        xmpMeta.setProperty( XMPConst.NS_EXIF, "DateTime", xmptoday);
        xmpMeta.setProperty( XMPConst.NS_EXIF, "DateTimeOriginal", xmptoday);

        xmpMeta.setProperty(XMPConst.NS_XMP, "CreatorTool", "Bridge SDK");
```

```
            xmpMeta.setProperty(XMPConst.NS_XMP, "CreateDate", xmptoday);
            xmpMeta.setProperty(XMPConst.NS_XMP, "MetadataDate", xmptoday);

            xmpMeta.setProperty(XMPConst.NS_DC, "creator", "Created by the Bridge SDK
            xmpMeta.setProperty(XMPConst.NS_DC, "title", this.name);
            xmpMeta.setProperty(XMPConst.NS_DC, "description", "A Bridge SDK node");

            XMPMeta.registerNamespace("http://ns.adobe.bridge.sdk/a/", "extHanA");

            xmpMeta.setProperty("http://ns.adobe.bridge.sdk/a/", "Description", "Some
            xmpMeta.setProperty("http://ns.adobe.bridge.sdk/a/", "ID", Math.floor(Mat

            var newXmp = xmpMeta.serialize(XMPConst.SERIALIZE_OMIT_PACKET_WRAPPER | X
            this.metadata = new Metadata( newXmp );
        }
        return this.metadata;
    }
    SDKNode.prototype.setLabelRating = function(type, val)
    {
        md = this.getMetadata();
        md.namespace = "http://ns.adobe.com/xap/1.0/";

        if(type == "Rating")
        {
            md.Rating = val;
        }
        else
        {
            md.Label = val;
        }
    }
    SDKNode.prototype.getLabelRating = function(type)
    {
        var retval = 0;
        md = this.getMetadata();
        md.namespace = "http://ns.adobe.com/xap/1.0/";

        if(type == "Rating")
        {
            retval = md.Rating;
        }
        else
        {
            retval = md.Label;
        }
        return retval;
    }
    SDKNode.prototype.setMetadata = function(md)
    {
        this.metadata = md;
    }
    SDKNode.prototype.setCustomInfosetData = function()
    {
        this.customString = "A Simple String for node: " + this.name;

        this.customNumber = (this.container) ? 1234 : 9876;
        this.customDate = new Date().toString();
        this.customArray = ["Array", "Of", "Strings"];
        this.customBool = true;

        this.FontName = [];
```

```
    this.FontFamily = [];
}
SDKNode.prototype.getCustomString = function(){ return this.customString; }
SDKNode.prototype.getCustomNumber = function(){ return this.customNumber; }
SDKNode.prototype.getCustomDate = function(){ return this.customDate; }
SDKNode.prototype.getCustomArray = function(){ return this.customArray; }
SDKNode.prototype.getCustomBool = function(){ return this.customBool; }
SDKNode.prototype.setCustomString = function(s){ this.customString = s; }
SDKNode.prototype.setCustomNumber = function(n){ this.customNumber = n; }
SDKNode.prototype.setCustomDate = function(d){ this.customDate = d; }
SDKNode.prototype.setCustomArray = function(a){ this.customArray = a; }
SDKNode.prototype.setCustomBool = function(b){ this.customBool = b; }
function SDKNode(name, iscontainer)
{
    this.children = new Array();
    this.childrenNames = {};
    this.path = "/" + name;
    this.name = name;
    this.parent = undefined;
    this.container = iscontainer;
    this.iconPath = new File($.fileName).parent.fsName.replace (/\\/gim, "/") + "
    this.metadata = undefined;
    this.setCustomInfosetData();
}
SDKNode.prototype.addNode = function(node)
{
    if(!this.getChildNode(node.getName()))
    {
        if(this.name.substr(0,10) != "SearchNode")
        {
            node.parent = this;
        }
        this.childrenNames[node.name] = node;
        this.children.push(node);
        return true;
    }
    return false;
}
SDKNode.prototype.getFirstChildNode = function() { return this.children[0]; }
SDKNode.prototype.getChildNode = function(name)
{
    SDKNode.counter++;
    return this.childrenNames[name];
}
SDKNode.prototype.findChildNode = function(name)
{
    var found = this.getChildNode(name);
    if(found instanceof SDKNode)
    {
        return found;
    }
    else
    {
        for(index in this.children)
        {
            found = this.children[index].findChildNode(name);
            if(found instanceof SDKNode)
            {
                return found;
            }
        }
```

```
    }
    return false;
}
SDKNode.prototype.findChildNodeFromPath = function(uri)
{
    var pathParts = uri.split("/");
    var currentNode = this;
    var searching = true;
    var pos = 1;
    while(searching)
    {
        if(typeof pathParts[pos+2] != "undefined")
        {
            pos++;
            currentNode = currentNode.getChildNode(pathParts[pos]);
        }
        else
        {
            currentNode = currentNode.getChildNode(pathParts[pos+1]);
            searching = false;
        }
    }
    if(currentNode instanceof Object && currentNode != false)
    {
        return currentNode;
    }
    else
    {
        return false;
    }
}
SDKNode.prototype.getChildren = function(){ return this.children; }
SDKNode.prototype.getChildrenNames = function(){ return this.childrenNames; }
SDKNode.prototype.getPath = function()
{
    var path = this.path;
    var parent = this.parent;
    while(parent)
    {
        var fullPath = parent.path + path;
        parent = parent.getParent();
        path = fullPath;
    }
    return path;
}
// For debugging only
SDKNode.prototype.listChildren = function()
{
    var kids = this.getChildren();
    for(var i = 0;i < kids.length;i++)
    {
        $.writeln(kids[i].getPath());
        kids[i].listChildren();
    }
}
SDKNode.prototype.hasChildren = function() { return this.children.length > 0; }
SDKNode.prototype.getParent = function() { return this.parent; }
SDKNode.prototype.removeNode = function(node)
{
    var name = node.getName();
    delete this.childrenNames[name];
```

```
    var tmp = new Array();
    for(var i = 0;i < this.children.length;i++)
    {
        if(this.children[i].name != name) { tmp.push(this.children[i]); }
    }
    this.children = tmp;
}
SDKNode.prototype.updateNode = function(node, oldName)
{
    delete this.childrenNames[oldName];
    this.childrenNames[node.getName()] = node;
}
SDKNode.prototype.cloneNode = function()
{
    var c = new Object();
    for( var i in this) { c[i] = this[i] }
    c.parent = undefined;
    return c;
}
SDKNode.prototype.setName = function(name)
{
    var oldName = this.name;
    this.name = name;
    this.path = "/" + name;
    if(this.parent != undefined)
    {
        this.parent.updateNode(this, oldName);
    }
}
SDKNode.prototype.isContainer = function() { return this.container; }
SDKNode.prototype.getName = function() { return this.name; }
SDKNode.prototype.getFileUrl = function(sys) { return "file:///" + sys.fsRoot + th
SDKNode.prototype.getIcon = function()
{
    if(this.isContainer())
    {
        return new BitmapData(this.iconPath + "/BEHFolderIcon32.png");
    }
    else
    {
        return new BitmapData(this.iconPath + "/BEHFileIcon32.png");
    }
}
SDKNode.prototype.getThumb = function()
{
    if(this.isContainer())
    {
        return new BitmapData(this.iconPath + "/BEHFolderIcon128.png");
    }
    else
    {
        return new BitmapData(this.iconPath + "/BEHFileIcon128.png");
    }
}
SDKNode.prototype.getPreview = function()
{
    if(this.isContainer())
    {
        return new BitmapData(this.iconPath + "/BEHFolderIcon512.png");
    }
    else
```

```
    {
        return new BitmapData(this.iconPath + "/BEHFileIcon512.png");
    }
}
SDKNode.prototype.getMetadata = function()
{
    if(this.metadata == undefined)
    {
        var xmpMeta = new XMPMeta( );
        var d = new Date()
        d.setFullYear(2007,0,1);
        d.setHours(1, 0, 0, 0);
        var xmptoday = new XMPDateTime(d);
        xmptoday.convertToLocalTime();

        xmpMeta.setProperty( XMPConst.NS_EXIF, "DateTime", xmptoday);
        xmpMeta.setProperty( XMPConst.NS_EXIF, "DateTimeOriginal", xmptoday);

        xmpMeta.setProperty(XMPConst.NS_XMP, "CreatorTool", "Bridge SDK");
        xmpMeta.setProperty(XMPConst.NS_XMP, "CreateDate", xmptoday);
        xmpMeta.setProperty(XMPConst.NS_XMP, "MetadataDate", xmptoday);

        xmpMeta.setProperty(XMPConst.NS_DC, "creator", "Created by the Bridge SDK
        xmpMeta.setProperty(XMPConst.NS_DC, "title", this.name);
        xmpMeta.setProperty(XMPConst.NS_DC, "description", "A Bridge SDK node");

        XMPMeta.registerNamespace("http://ns.adobe.bridge.sdk/a/", "extHanA");

        xmpMeta.setProperty("http://ns.adobe.bridge.sdk/a/", "Description", "Some
        xmpMeta.setProperty("http://ns.adobe.bridge.sdk/a/", "ID", Math.floor(Mat

        var newXmp = xmpMeta.serialize(XMPConst.SERIALIZE_OMIT_PACKET_WRAPPER | XI
        this.metadata = new Metadata( newXmp );
    }
    return this.metadata;
}
SDKNode.prototype.setLabelRating = function(type, val)
{
    md = this.getMetadata();
    md.namespace = "http://ns.adobe.com/xap/1.0/";

    if(type == "Rating")
    {
        md.Rating = val;
    }
    else
    {
        md.Label = val;
    }
}
SDKNode.prototype.getLabelRating = function(type)
{
    var retval = 0;
    md = this.getMetadata();
    md.namespace = "http://ns.adobe.com/xap/1.0/";

    if(type == "Rating")
    {
        retval = md.Rating;
    }
    else
```

```
    {
        retval = md.Label;
    }
    return retval;
}
SDKNode.prototype.setMetadata = function(md)
{
    this.metadata = md;
}
SDKNode.prototype.setCustomInfosetData = function()
{
    this.customString = "A Simple String for node: " + this.name;

    this.customNumber = (this.container) ? 1234 : 9876;
    this.customDate = new Date().toString();
    this.customArray = ["Array", "Of", "Strings"];
    this.customBool = true;

    this.FontName = [];
    this.FontFamily = [];
}
SDKNode.prototype.getCustomString = function(){ return this.customString; }
SDKNode.prototype.getCustomNumber = function(){ return this.customNumber; }
SDKNode.prototype.getCustomDate = function(){ return this.customDate; }
SDKNode.prototype.getCustomArray = function(){ return this.customArray; }
SDKNode.prototype.getCustomBool = function(){ return this.customBool; }
SDKNode.prototype.setCustomString = function(s){ this.customString = s; }
SDKNode.prototype.setCustomNumber = function(n){ this.customNumber = n; }
SDKNode.prototype.setCustomDate = function(d){ this.customDate = d; }
SDKNode.prototype.setCustomArray = function(a){ this.customArray = a; }
SDKNode.prototype.setCustomBool = function(b){ this.customBool = b; }
```

読みづらくてすいません。
こちらの環境はWin10、BridgeCC2018です。

目的としては、フォントの管理であったり、イラレでのリンクファイルの状態(埋まってる埋まってない)管理をブリッジでできるといいなと。以前のイラレではメタデータ検索で埋まってないものを 検索出来たんですが、今のバージョンでは埋めたリンクファイルにも同じ要素が付けられるようにしまったので。

本題です。
０．これ動きますか？
こちらの環境では一応動きます。軽快とは程遠いですが。

１．スクリプトから登録されたお気に入りに移動するとようやく追加したフィルタが有効(フィルタパネルメニューに表示される)になるのですが、これを通常の状態でも有効にするにはどうしたらよ いでしょうか。
接頭辞を通常の"bridge:fs:"にしてみましたが、その場合フィルタは有効になりませんでした。

２．テキストファイルやフォルダなどのアイコンがすべてシステムドライブのアイコンになってしまうのですが、OSと同様の表示にするにはどうしたらよいでしょうか。
通常のサムネイルオブジェクトのiconやthumbnailはundefinedでした。サンプルスクリプトのように画像を用意したうえで拡張子毎に振り分けるとか、shell32.dll などのライブラリを使うとかが必要でしょうか。

３．動作が重い、コケるのを回避するには
20や30程度ならさほど気にはならないですが、100を越えだすと重いうえにコケる率が上がる気がします。終了時にはよくクラッシュします。クラッシュするのはサンプルでも同様でしたけど。

４．現状では、スクリプトを実行したディレクトリのみ表示されて、上下どちらの階層にも移動できません。
サンプルではすべての要素を列挙しています。それ以外に手はないのでしょうか。

改良策や構文の訂正など、何卒宜しくお願い致します。

I have the same question (0)

982 Views Tags :

10 A Mar 30, 2018 2:51 AM (in response to ruser3t30054468)

ACP

Re: [Br] カスタムフィルタについて

ぱっと見ただけでまともに検証してませんが、XMPを扱う場合ExternalObjectを利用するほうが高速になります。

[参考] Extend_Script_experimentals/XMPtool.jsx at master · ten-A/Extend_Script_experimentals · GitHub

Actions Edit Delete Like (1)

ruser3t30054468 Apr 2, 2018 5:07 PM (in response to 10 A)

Re: [Br] カスタムフィルタについて

返信ありがとうございます。

900行付近の.readの部分は以下のようにしました。

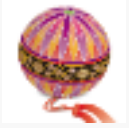
```
if (fsThumb.hasMetadata){
    var fontflag = true;
    var fontNameAry = [];
    var fontFamilyAry = [];

    var xmpFile = new XMPFile(fsThumb.spec.fsName, XMPConst.UNKNOWN, XMPConst.
    var xmp = xmpFile.getXMP ();

    for (i = 0; fontflag == true; i ++){
        if (xmp.getStructField ("http://ns.adobe.com/xap/1.0/t/pg/", "xmpTPg:F
            fontflag = false;
        }else{
            fontNameAry.push (xmp.getStructField ("http://ns.adobe.com/xap/1.0
            fontFamilyAry.push (xmp.getStructField ("http://ns.adobe.com/xap/1
            };
        };
    if (fontNameAry.length) {
        infoset.FontName = fontNameAry;
        infoset.FontFamily = fontFamilyAry;
        };
    };
```

```
if (fsThumb.hasMetadata){
    var fontflag = true;
    var fontNameAry = [];
    var fontFamilyAry = [];

    var xmpFile = new XMPFile(fsThumb.spec.fsName, XMPConst.UNKNOWN, XMPConst.
    var xmp = xmpFile.getXMP ();

    for (i = 0; fontflag == true; i ++){
        if (xmp.getStructField ("http://ns.adobe.com/xap/1.0/t/pg/", "xmpTPg:F
            fontflag = false;
        }else{
            fontNameAry.push (xmp.getStructField ("http://ns.adobe.com/xap/1.0
            fontFamilyAry.push (xmp.getStructField ("http://ns.adobe.com/xap/1
            };
        };
    if (fontNameAry.length) {
        infoset.FontName = fontNameAry;
        infoset.FontFamily = fontFamilyAry;
        };
    };
```

他にもExternalObjectで扱うべき箇所を見落としていたら教えてください。

XMPの扱いも難しいですね、配列だの構造だの、なかなか目的の値の取得ができませんでした。

リンク先でXMPMetaオブジェクトをserializeして再度XMPMetaオブジェクトを作っているようでしたが、この処理にはどんな効果があるんでしょうか？



10 A Apr 2, 2018 8:10 PM (in response to ruser3t30054468)

Re: [Br] カスタムフィルタについて

ACP

XMPの処理は、getXMPでXMP Rowデータをファイルから取り出してそれをXMP Metaとして正規化するという流れだったように思います。先に挙げたわたしのサンプルではこの順番で処理していると思います。
投稿されたコードですが、パネル自体は表示されます。しかしながら中身を表示させることが出来ませんでした。長いコードなので問題を追いきれていません。
ただ2018前提ならばCEPのEmbeddedエクステンションで作ればかなり楽になるのではないかと感じました。

Actions Edit Delete Like (0)

ruser3t30054468 Apr 3, 2018 5:29 PM (in response to 10 A)

Re: [Br] カスタムフィルタについて

同じXMPMetaオブジェクトでも中身が違うというようなことですね、ありがとうございます。
CEPはhtmlとかよくわからないので尻込みしています。javascriptもろくにわかっていないんですけどね...

表示されたパネルというのは"SDK: Custom Sort Infoset Data"というタイトルでしたら、サンプルの残骸というか、フィルタのカスタムとは直接関係がないものです。余計な部分を取っ払ってなくていろいろわかりにくくなっていますね、すいません 。

スクリプトを実行すると、スクリプト実行時点の表示ディレクトリがお気に入りパネルの最後と、フォルダパネルの最後に追加されると思います。そこに移動するとスクリプトで定義したサムネイルが 表示され、その中ではフィルターパネル(もしくはフィルターパネルメニュー)に項目が追加されているはずです。"SDK~"パネルにも値が表示されていると思います。

458行のt.refresh();の後などに、app.document.thumbnail = t; とかすると直行できます。

通常動作時の、URI接頭辞が"bridge:fs:"の場所？空間？から、スクリプト定義の場所に移動しないと、追加したフィルタやインフォセットが反映されないようです。接頭辞に紐づけさ れているextensionHandlerが動かな

いっていうことかと想像しています。

Actions Edit Delete Report Abuse Like (0)

10 A Apr 3, 2018 8:34 PM (in response to ruser3t30054468)

Re: [Br] カスタムフィルタについて

ACP

残念ながらパネル内はまっしろけでして、中のUI自体組み立てられない様子なんです。
ちなみにOSX10.11.6/BridgeCC2018にて検証しています。

CEPエクステンションはUIをHTMLで構成しますが、ScriptUIに比べれば学習コストが低く、参照できるドキュメントも比較にならないぐらい豊富です。このレベルのJavaScri ptを組み立てられるなら移行は比較的楽にできると思います。
以下にAdobe公式からリリースされているサンプルエクステンションへのリンクを掲載しておきます。

https://github.com/Adobe-CEP/Samples/tree/master/Bridge

Actions Edit Delete Like (0)

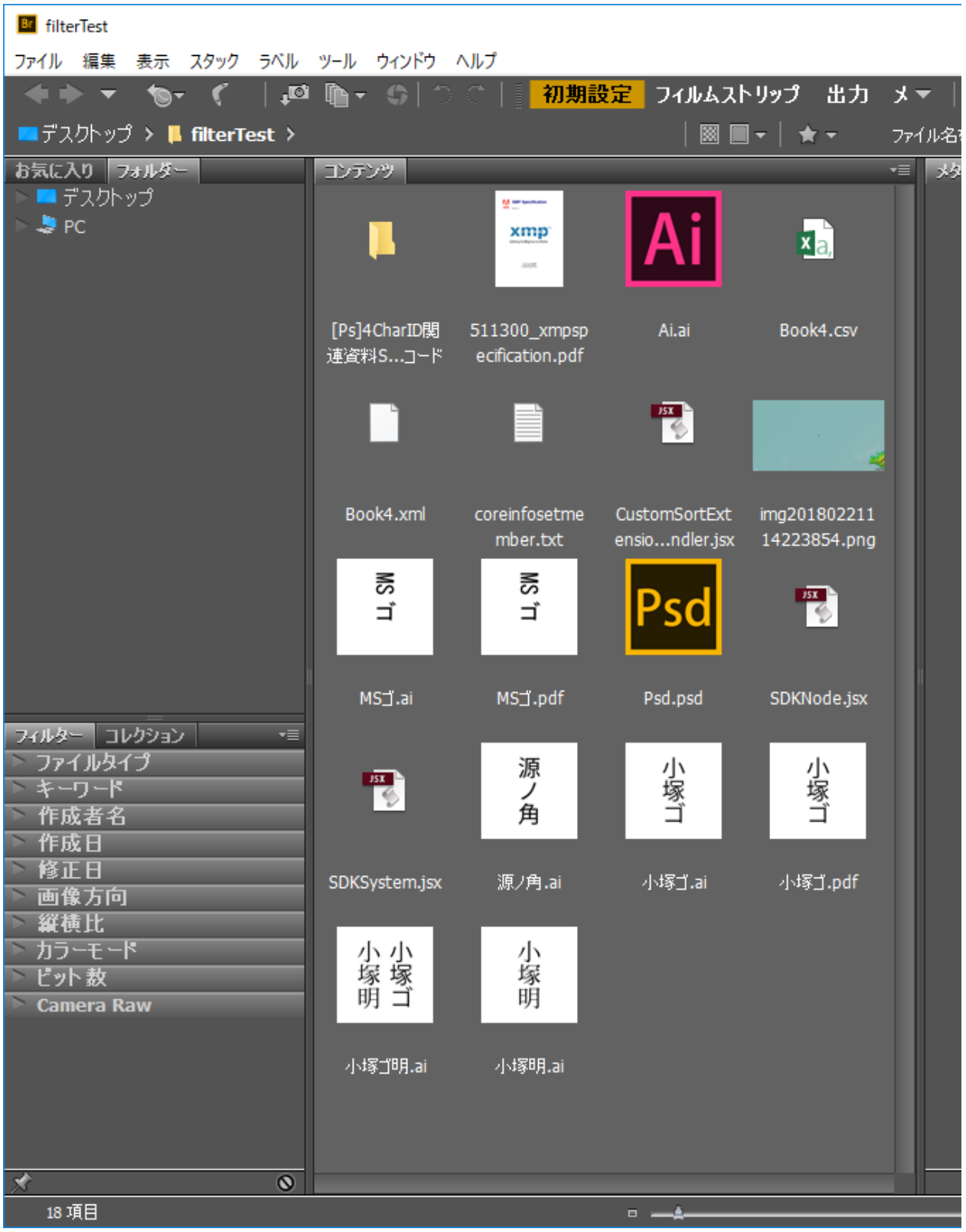
ruser3t30054468 Apr 10, 2018 7:41 PM (in response to 10 A)

Re: [Br] カスタムフィルタについて

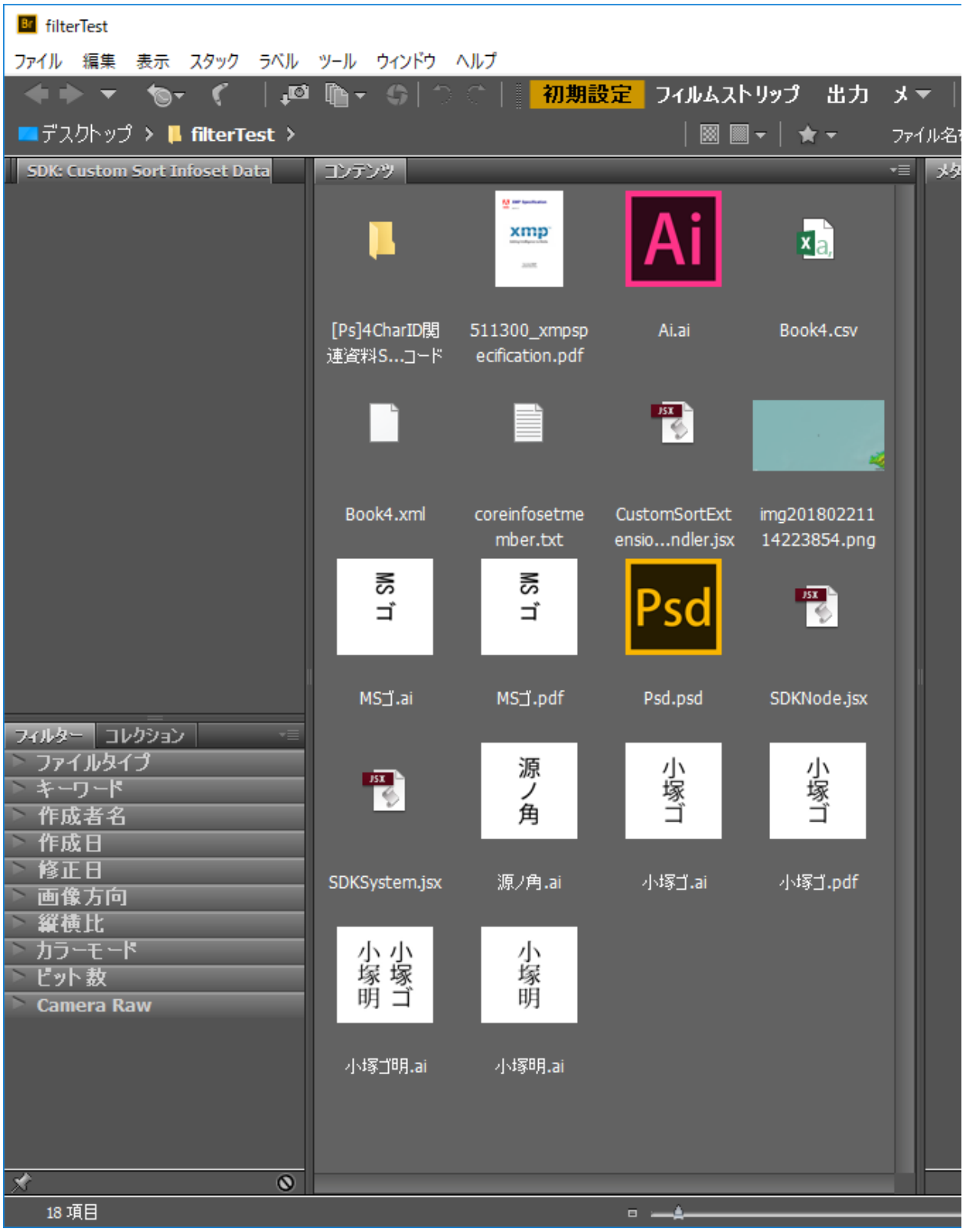
テストしていただいてありがとうございます。
サンプルから変更した部分がWindows依存の書き方になってしまっているんでしょうか。

一応、スクショ載せておきます。

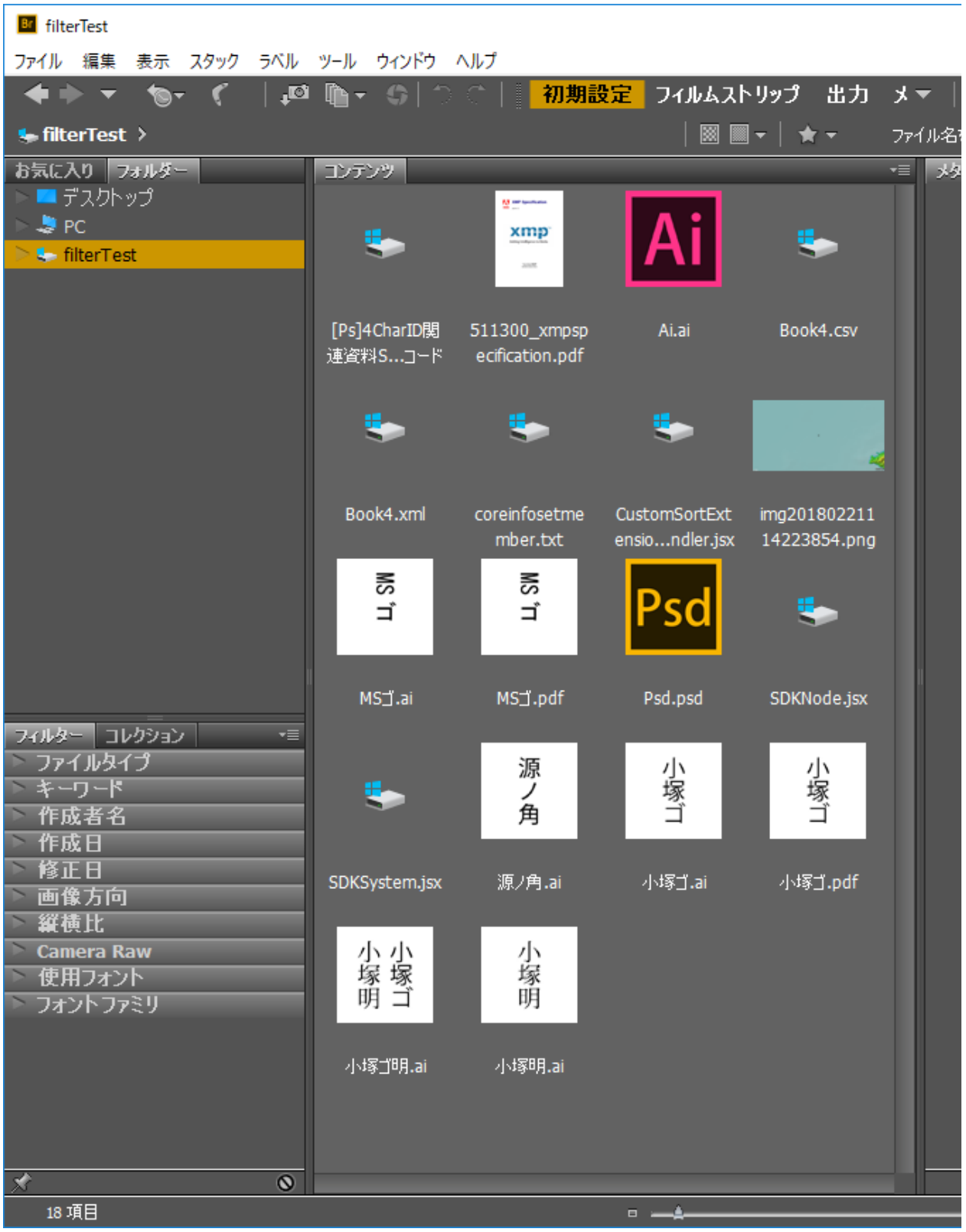
スクリプト実行前


filterTest
ファイル 編集 表示 スタック ラベル ツール ウィンドウ ヘルプ
初期設定 フィルムストリップ 出力
デスクトップ > filterTest >
お気に入り フォルダー
デスクトップ
PC
コンテンツ
[Ps]4CharID関連資料S...コード
511300_xmpspecification.pdf
Ai.ai
Book4.csv
Book4.xml
coreinfosetmember.txt
CustomSortExtensio...ndler.jsx
img20180221114223854.png
MSゴ.ai
MSゴ.pdf
Psd.psd
SDKNode.jsx
SDKSystem.jsx
源ノ角.ai
小塚ゴ.ai
小塚ゴ.pdf
小塚ゴ明.ai
小塚明.ai
フィルター コレクション
ファイルタイプ
キーワード
作成者名
作成日
修正日
画像方向
縦横比
カラーモード
ビット数
Camera Raw
18 項目

実行直後



スクリプト定義のサムネイル（今回の場合はフォルダパネルの"filterTest"）に移動後



"SDK:~"のパネルは消しています。
フィルタパネルのリスト下部、"使用フォント"と"フォントファミリ"が追加したフィルタです。
メタデータを拾い損ねているのか"カラーモード"と"ビット数"が消えてしまってますね。

フィルタの適用後

こんな感じです。

Actions Edit Delete Report Abuse Like (0)