Best Practices for C# API Lists and List Panels

Lists and List Panels

Lists and List Panels, or simply "Panels", are used to enumerate and loop through content. They are best used in situations where the CMS end user needs to include a flexible quantity of a specific field or set of fields or content. For example, you would create a list panel in a template to manage slide data for a slideshow, where the user would upload any number of images and enter related caption information, or you could create a list panel for managing a list of related links where each link panel has fields for link type, title text and link.

Best Practices

Create the panel Input fields in the appropriate area of your input form. A simple example with two text boxes follows:

```
while (Input.NextPanel("contact_counter", displayName: "Contact List"))
{
Input.ShowTextBox("Contact Name", "contact_name");
Input.ShowTextBox("Contact Number", "contact_number");
}
```

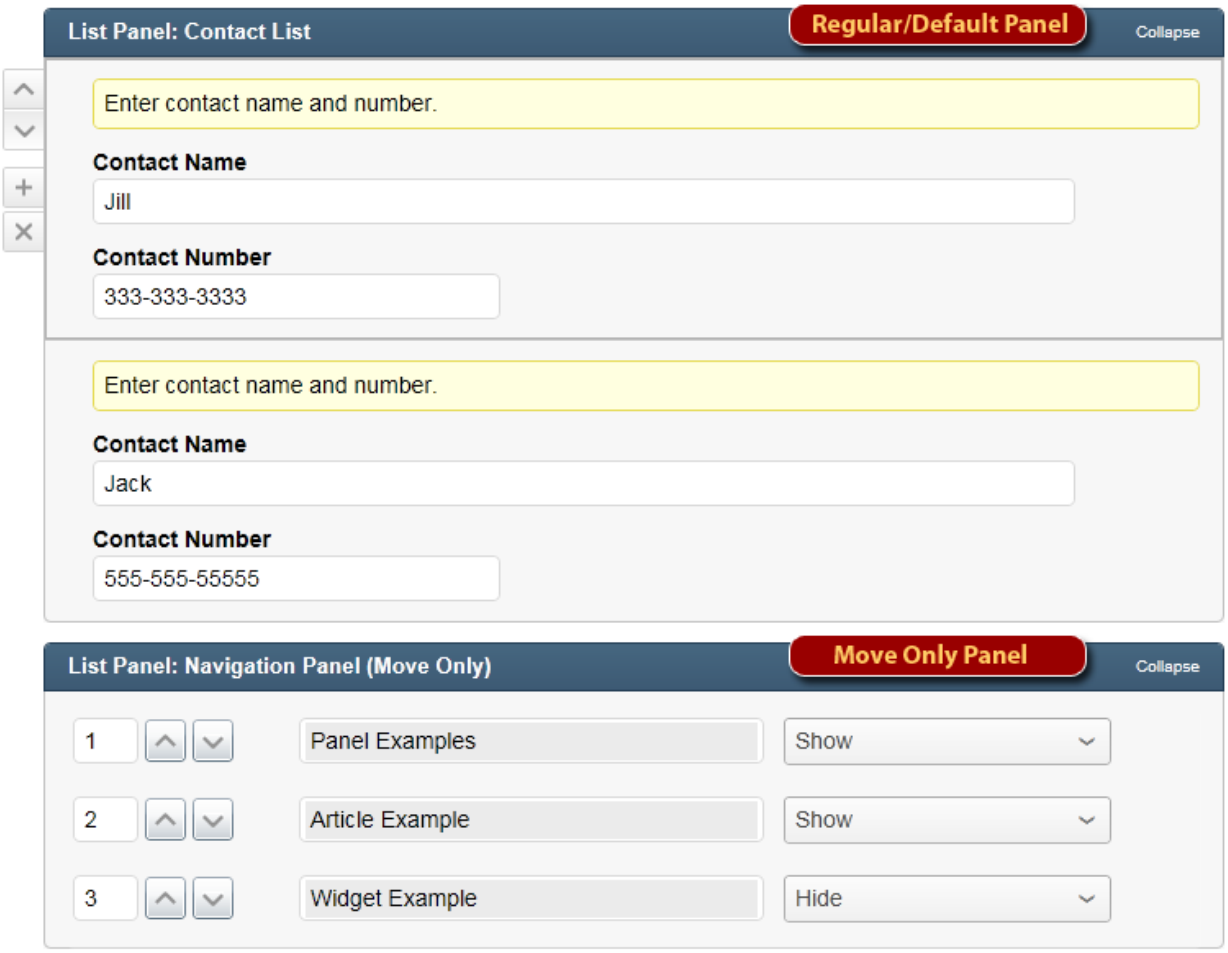
To include additional fields in your list, move the cursor to the desired point in your code, and add fields as needed within the while loop braces.

The parameter “displayName:” is optional but gives you control over the ControlPanel label that wraps the input panel in the form (the default is the while iterator field name). Panel loops also have parameters for setting minimum number of entries, maximum number of entries, as well as other formatting options. See http://help.crownpeak.com/cmsapi/ Main Classes > Input > NextPanel for examples. The following example sets a minimum of 2 panels and maximum of 5, and also adds an end-user instruction message.

```
while (Input.NextPanel("contact_counter", min:2, max:5, displayName: "Contact List"))
{
Input.ShowMessage("Enter contact name and number.");
Input.ShowTextBox("Contact Name", "contact_name");
Input.ShowTextBox("Contact Number", "contact_number", width: 30);
       }
```

Input Panel Types - there are four types of input panels in Classic (Regular [default], MoveOnly, InlineButtons and Sorted), two in Volte (Regular [default] and MoveOnly, other panel types have been deprecated)

Use the regular panel type for most content panels. MoveOnly is a compact design generally used for setting manual sort orders and showing/hiding assets in config panels (covered later).



You can also nest panels (create panels or subpanels inside of panels). Below is an example of a simple embedded panel in the input.aspx.

```
<% while (Input.NextPanel("panel_counter", displayName:"Nested Panel Top"))
    { Input.ShowTextBox("Panel Title", "panel_title");
       while (Input.NextPanel("subpanel_counter", displayName:"Nested Panel Sub
Panel"))
        { Input.ShowTextBox("SubPanel Title", "subpanel_title");
        }
  } %>
```

Outputting a Panel

To display the code in the output, load panel entries (class PanelEntry) into a List and iterate through them as follows (for the first panel example):

```
<% List<PanelEntry> panels = asset.GetPanels("contact_counter");
   foreach (PanelEntry panel in panels) {
      %><p><strong><%= panel["contact_name"]%></strong> - <%=
panel["contact_number"]%></p><%
   } %>
```

When outputting nested/embedded lists, it is important to remember that your first/outer list is created from the asset fields via List<PanelEntry> panels = asset.GetPanels("panel_counter"); but the embedded/inner list references the current PanelEntry in the list, which would be named in your outer foreach loop. An example if your foreach variable name is "PanelEntry panel" would be List<PanelEntry> subPanels = **panel**.GetPanels("subpanel_counter")):

```
<% List<PanelEntry> panels2 = asset.GetPanels("panel_counter");
   foreach (PanelEntry panel in panels2) {
      %><h1>panel title: <%= panel["panel_title"]%></h1>
      <ul>
      <%
       List<PanelEntry> subpanels = panel.GetPanels("subpanel_counter");
        foreach (PanelEntry subpanel in subpanels) {
            %><li>sub panel title: <%= subpanel["subpanel_title"]%></li><%
         } %>
      </ul><!-- end subpanels-->
          <% } %>
```

Working with Panels

The C# CMS API includes a few classes/methods for working with panels.

- NextPanel is the Input method used when creating panels
- PanelEntry is a class that represents a single panel within a list panel, and is the usual class referred to by content stored in an asset itself.
- AssetPanelEntry is similar in that it represents a single Panel within a list panel but it is used with content that was intialized from the children of a folder. It is used for creating certain types of index and config templates and is returned in a List by calling asset.GetPanelsFromFolder().

Panels in Volte

In Volte, the CMS currently renders List Panels in the Edit Form view with a click-to-expand header based on the panel name Iterator.

In the Edit, "In Context Editing" view, existing panels are displayed and editable in the **“**Half**”** Form view, but you must switch to the Full Edit Form view to add panels.

Referencing a list using the asset object instead of the list object (for example, asset["panel_field"] instead of panel["panel_field"]) will not return any data in the C# API. (In the VBScript API it might return the first panel's content.)

Common Errors

Referencing a list
When working with list panels, try to avoid referencing a list using the asset object instead of the list object (for example, asset["panel_field"] instead of panel["panel_field"]). This will not return any data in the C# API.

Multiple similar panels on the same template
If repeating a similar panel structure in the same template, your panels must have unique content field names to prevent data being overwritten and to prevent duplicate field names, which causes a run-time CMS error message. This can be an issue especially if re-using panel functions in multiple parts of the same template (for instance, a related links function that is re-used in several sections on the page). Work around this by titling panel content fields based on an identifying parameter prefix.

Example:

```
<%
   panelInput("First Panel", "firstpanel");
   panelInput("Second Panel", "secondpanel");
     %>

<script runat="server" data-cpcode="true">
 public void panelInput(String panelLabel, String fieldName) {
       while (Input.NextPanel(fieldName + "_counter",
              displayName:panelLabel))
      { Input.ShowTextBox("Panel Field", fieldName + "_field"); }
        }
</script>
```