Managing iApp template files with iControl

Joe Pruitt, 2012-23-02

Templates

An iApp is a user-customizable framework for deploying applications. It consists of three components: Templates, Application Services, and Analytics.

- An iApp Template is where the application is described and the objects (required and optional) are defined through presentation and implementation language.
- An iApp Application Service is the deployment process of an iApp Template which bundles all of the configuration options for a particular application together. You would have an iApp Application Service for SharePoint, for example.
- iApp Analytics include performance metrics on a per-application and location basis.

In this article, I will illustrate the format of a packaged template and how you would use the modular iControl calls to support importing and exporting with that format.

The Template File (.tmpl)

Since templates consist of several pieces, the BIG-IP dev team created a packaging format for all of the data used to configure them. This is what we refer to as a .tmpl file. A .tmpl file is a text file that follows the following format (as of BIG-IP version 11.1):

```
sys application template <template_name> {
  actions {
    definition {
      html-help {
        # HTML Help for the template
      }
      implementation {
        # TMSH implementation code
      }
      presentation {
        # APL presentation language
      }
      role-acl <security role>
      run-as <user context>
    }
  }
  description <template description>
  partition <partition name>
}
```

The first line is the term “sys application template” followed by the template name. The template section contains the following sections:

- **actions** - Currently the only action is a “definition”.
- **description** - A text based description for the template.
- **partition** - The storage partition the template is to be placed in.

The action (or definition) contains the following configuration elements

- **html-help** - The HTML help that is displayed in the GUI when creating an iApp service.
- **implementation** - The TMSH implementation code used at create time for an iApp service.
- **presentation** - The APL code used to describe the GUI presentation when generating an iApp service.
- **role-acl** - The security role for this iApp. This value is typically "none".
- **run-as** - internal value to allow the implementation to be executed with a different user context. The value for this is typically "none".

iControl Methods

Presentation Help

the presentation help component of an application template is just plain HTML with a few restrictions. We've blocked out components such as iframes, scripts, tables, and a few other components that could cause security implications or page formatting problems when displayed within the BIG-IP GUI. The API methods to access the presentation help are the Management::ApplicationTemplate::get_action_presentation_help() and set_action_presentation_help() methods. The get method, returns the list of help sections for the specified templates, while the set method passes them in as input. Remember, as of version 11.1, "definition" is the only supported action.

```
String [][] Management::ApplicationTemplate::get_action_presentation_help(
  in String [] templates,
  in String [][] actions
);

Management::ApplicationTemplate::set_action_presentation_help(
  in String [] templates,
  in String [][] actions,
  in String [][] values
);
```

Implementation

The implementation is just another text block that contains the tmsh code used when creating the application service. The two methods used to get at this are the get_action_implementation() and set_action_implementation() methods that work in a very similar fashion to the presentation help methods.

```
String [][] Management::ApplicationTemplate::get_action_implementation(
  in String [] templates,
  in String [][] actions
);

Management::ApplicationTemplate::set_action_implementation(
  in String [] templates,
  in String [][] actions,
  in String [][] values
);
```

Presentation

The presentation layer is another text block that contains the APL code that describes how the user interface presentation components are displayed. The get_action_presentation() and set_action_presentation() methods are used to get and set these values.

```
String [][] Management::ApplicationTemplate::get_action_presentation(
  in String [] templates,
  in String [][] actions
);

Management::ApplicationTemplate::set_action_presentation(
  in String [] templates,
  in String [][] actions,
  in String [][] values
);
```

Description

The one last piece that we have exposed through iControl is the description meta-data field. The get_description() and set_description() methods are used to retrieve these values.

```
String [] Management::ApplicationTemplate::get_description(
  in String [] templates
);

Management::ApplicationTemplate::set_description(
  in String [] templates,
  in String [] values
);
```

Exporting To A .tmpl File With iControl

Exporting to a .tmpl file is as simple as querying the various components and packaging them up in the .tmpl file format. The first step is to create a local PowerShell object containing the relevant template configuration data. The New-Template function does just this:

```
function New-Template()
{
  $t = 1 | Select Name, Description, HtmlHelp, Implementation, Presentation;
  $t;
}
```

Now comes the fun of creating a .tmpl file and saving it locally. The Export-Template function takes as parameters the name of the local template file, the name of the template on the BIG-IP. The iControl methods are called to query the description, implementation, presentation, and presentation help and stored in the Template variable $t. The $s variable is used to store the string representation of the .tmpl file.

The various sections of the .tmpl are created and then the .tmpl file is saved to disk.

```
function Export-Template()
{
  param(
    $TemplateFile = $null,
    $TemplateName = $null,
    [Boolean]$Force = $false
  );
```

```
$t = New-Template;
$t.Name = $TemplateName;

$descA = (Get-F5.iControl).ManagementApplicationTemplate.get_description( @($t.Name) );
$implAofA = (Get-F5.iControl).ManagementApplicationTemplate.get_action_implementation( @($t.Name),
$presAofA = (Get-F5.iControl).ManagementApplicationTemplate.get_action_presentation( @($t.Name), @(
$helpAofA = (Get-F5.iControl).ManagementApplicationTemplate.get_action_presentation_help( @($t.Name

$t.Description = $descA[0];
if ( $t.Description.Length -eq 0 ) { $t.Description = "none"; }
$t.Implementation = $implAofA[0][0];
$t.Presentation = $presAofA[0][0];
$t.HtmlHelp = $helpAofA[0][0];

$s = "sys application template ";
$s += Split-Path -Leaf $t.Name;
$s += " {`r`n";
# Begin Action
$s += " "*4;
$s += "actions {`r`n";

# Begin definition
$s += " "*8;
$s += "definition {`r`n";

# Help
$s += " "*12;
$s += "html-help {`r`n";
$s += $t.HtmlHelp;
if ( $t.HtmlHelp.Length -gt 0 ) { $s += "`r`n"; }
$s += "}`r`n";

# Implementation
$s += " "*12;
$s += "implementation {`r`n";
$s += $t.Implementation;
if ( $t.Implementation.Length -gt 0 ) { $s += "`r`n"; }
$s += "}`r`n";

# Presentation
$s += " "*12;
$s += "presentation {`r`n";
$s += $t.Presentation;
if ( $t.Presentation.Length -gt 0 ) { $s += "`r`n"; }
$s += "}`r`n";

$s += " "*12;
$s += "role-acl none`r`n";
$s += " "*12;
$s += "run-as none`r`n";

# End Definition
$s += " "*8;
$s += "}`r`n";

# End Action
$s += " "*4;
$s += "}`r`n";
```

```
  $s += " "*4;
  $s += "description $($t.Description)`r`n";
  $s += " "*4;
  $s += "partition $( (Split-Path $t.Name).SubString(1) )`r`n";

  #End template
  $s += "}";

  if ( $null -eq $TemplateFile )
  {
    $TemplateFile = ".\Templates\" + (Split-Path -Leaf $t.Name) + ".tmpl";
  }

  $s | Out-File $TemplateFile;

  Write-Host "Template '$TemplateName' saved to file '$TemplateFile'";
}
```

Importing A .tmpl File With iControl

Once you have exported a .tmpl file (whether via iControl, the BIG-IP admin GUI, or from the iApp CodeShare on DevCentral), you may want to import them back onto the device. You can do this through the GUI, but's that no fun. I'm going to show you how to do it programmatically.

The Import-Template function takes as input a local .tmpl file along with a Force flag to indicate whether you want to overwrite the application template if it already exists.

The content is read into the $content variable and then it's passed into the Parse-Template function described in the next section. The Management::ApplicationTemplate::create() method is used to create the new empty template. If an exception is thrown (most likely due to prior existence), then the Force flag is used to determine whether we will overwrite the existing template. If not, an error message occurs and processing halts.

If either the template is new or we are overwriting it, the iControl methods are used to set the implementation, presentation,and presentation help for the template on the BIG-IP. At this point, the template has been imported.

```
function Import-Template()
{
  param(
    $TemplateFile = $null,
    [Boolean]$Force = $false
  );

  $content = [System.IO.File]::ReadAllText((Resolve-Path $TemplateFile).Path)
  $t = Parse-Template -Template $content;
  $bContinue = $true;

  if ( $null -ne $t )
  {
    Write-Host "CREATING Template: '$($t.Name)'...";
    try
    {
      (Get-F5.iControl).ManagementApplicationTemplate.create( @($t.Name) );
    }
    catch
    {
      $bContinue = $Force;
      if ( $bContinue )
      {
        Write-Host "Template exists, overwriting...";
      }
      else
      {
        Write-Host "Template exists, stopping import.  Use '-Force' to overwrite.";
      }
    }

    if ( $bContinue )
    {
      Write-Host "SETTING IMPLEMENTATION...";
      (Get-F5.iControl).ManagementApplicationTemplate.set_action_implementation( @($t.Name), @( @("de
      Write-Host "SETTING PRESENTATION...";
      (Get-F5.iControl).ManagementApplicationTemplate.set_action_presentation( @($t.Name), @( @("defi
      Write-Host "SETTING PRESENTATION HELP...";
      (Get-F5.iControl).ManagementApplicationTemplate.set_action_presentation_help( @($t.Name), @( @(
    }
  }
}
```

As I mentioned above, there are a few utility methods used to parse the template file. The Parse-Template method takes in the whole content of a template file and extracts the various sections into a new Template object and returns it to the caller.

```
function Parse-Template()
{
  param($Template = $null);
  $t = $null;

  if ( $Template.StartsWith("sys application template") )
  {
    $t = New-Template;

    $t.Name = Get-NextToken $Template "sys application template";
    $definition = Extract-Section $Template "definition";
    $t.HtmlHelp       = Extract-Section $definition "html-help";
    $t.Implementation = Extract-Section $definition "implementation";
    $t.Presentation   = Extract-Section $definition "presentation";
  }
  $t;
}

function Extract-Section()
{
  param($Template = $null,
    $Section = $null
  );

  $Value = $null;

  $idxStart = $Template.IndexOf("$Section {");
  $idxEnd = -1;
  if ( -1 -ne $idxStart )
  {
    $idxStart += $Section.Length + 2;
    $p = 1;
    for($i = $idxStart; $i -lt $Template.Length; $i++)
    {
      switch ($Template[$i])
      {
        "{" {
          $p++;
        }
        "}" {
          $p--;
          if ( $p -eq 0 ) {
            $idxEnd = $i;
            $i = $Template.Length;
            break;
          }
        }
      }
    }
    if ( -1 -ne $idxEnd )
    {
      $Value = $Template.SubString($idxStart, $idxEnd-$idxStart).Trim();
    }
  }
  $Value;
}
```

Conclusion

Armed with the functions in this script, you can now interoperate with the application template formats generated by the the BIG-IP administrative GUI and the DevCentral CodeShare.

Download The Full Source

The entire source for this sample can be downloaded from the iControl CodeShare under the topic PowerShelliAppTemplateControl.