Run PowerShell Scripts through a Server Event

Scenario

You want to use a K2 blackpearl workflow to issue PowerShell commands through a script which accepts a parameter and outputs a value.

Requirements

- Visual Studio
- K2 for Visual Studio
- Familiarity with the K2 Workspace
- Full access to a folder on the hard drive of the K2 host server (both for the account you're using as the developer AND for the account K2 uses to run your workflow).
- Alternate credentials under which you want the server event to execute (optional; this account must have access to the aforementioned folder on the K2 host server)
- LDAP over SSL (LDAPS) in the development and production environments
- Access to the instance of SQL Server running the K2 Server (if refreshing the User Manager Cache is required)

References

(a) How to enable LDAP over SSL with a third-party certification authority
(b) Refresh the User Manager Cache

Special Note to Users of K2 Studio

If you've been using K2 Studio for all of your K2 blackpearl development, you'll need to install a copy of Visual Studio, then install K2 for Visual Studio afterwards. **K2 Studio does not have the capability to write the server event code you'll require.**

Be sure to install a copy of Visual Studio supported by the version of K2 you're running. For example, versions of K2 blackpearl prior to 4.6.3 (4.12060.1530.0) will not support Visual Studio 2010. Furthermore, if you intend to continue reliance on K2 Studio as your IDE, consider installing Visual Studio and K2 for Visual Studio on a separate workstation if you can.

Overview

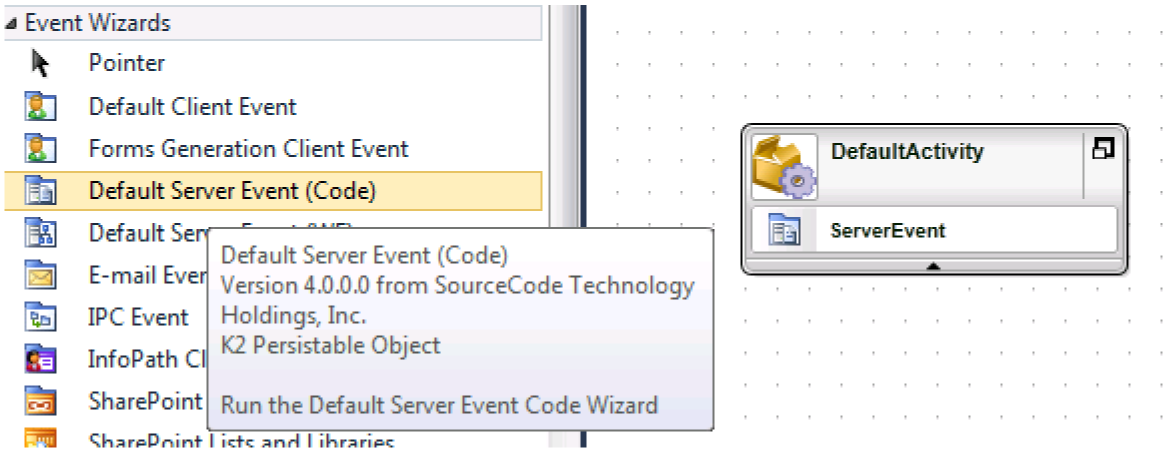
You're going to use several technologies to achieve your objective. In the broad strokes, you'll use K2 blackpearl to create a Server Event, and use C# to create a PowerShell Runspace in which a Powershell script, located on the server, will be invoked. Data will flow from a datafield in the workflow into the script, and results will be returned to a datafield in the workflow.

To get started, load your workflow solution in Visual Studio, and locate or create the activity in which you want the PowerShell script executed.

Steps

1. Create the Server Event

Drag the Default Server Event (Code) wizard into the target activity from the toolbox.

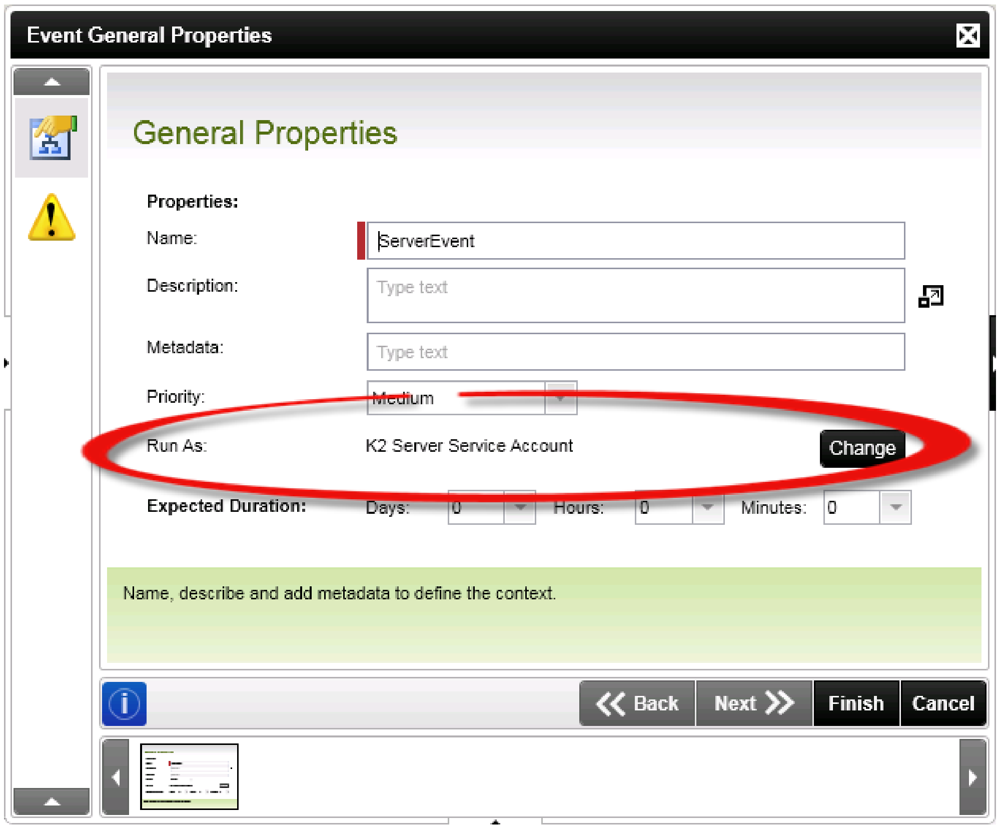

Event Wizards
Pointer
Default Client Event
Forms Generation Client Event
Default Server Event (Code)
Default Ser
E-mail Ever
IPC Event
InfoPath Cl
SharePoint
SharePoint Lists and Libraries
Default Server Event (Code)
Version 4.0.0.0 from SourceCode Technology Holdings, Inc.
K2 Persistable Object
Run the Default Server Event Code Wizard
DefaultActivity
ServerEvent

LDAP over SSL

If an exception is thrown when you drag the Server Event to the canvas, it's possible the cause may be that LDAP over SSL is not configured for your environment. In Windows Server 2008, LDAPS is not configured by default. K2 notes that "a configuration issue may occur due to the missing LDAPS support as the environment will not have a certification authority (CA) installed and also will not have the appropriate server authentication certificate configured." See the link provided in reference (a) for additional information.

Run with Alternate Credentials

Server Events may be executed using arbitrary account credentials. This is particularly useful if you're having scripts run to perform actions requiring elevated account privileges. To modify the account credentials to be used, right-click on the Server Event and select **Properties**. On the General Properties pane, you'll see **Run As:** , followed by an account name and a **Change** button.



Event General Properties

General Properties

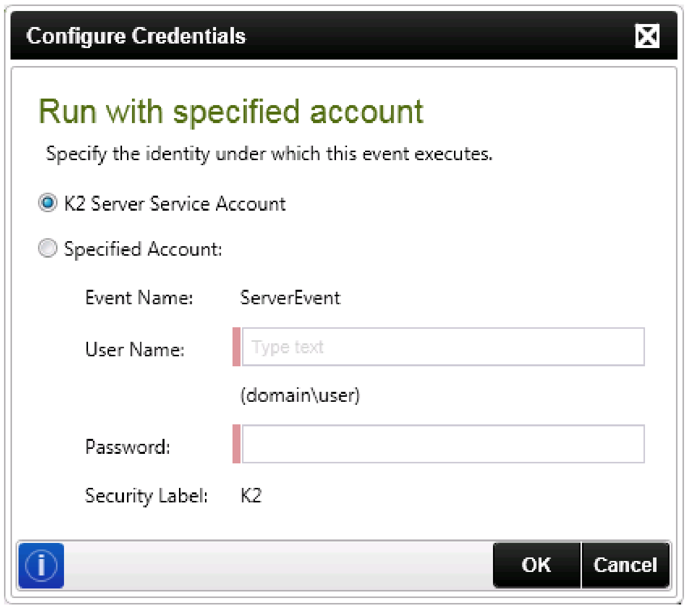
Properties:

Name:	ServerEvent
Description:	Type text
Metadata:	Type text
Priority:	Medium
Run As:	K2 Server Service Account

Expected Duration: Days: 0 Hours: 0 Minutes: 0

Name, describe and add metadata to define the context.

Click the **Change** button to bring up the Configure Credentials dialog. You may leave the account set to the K2 Server service account, or supply a specific user name and password.



Configure Credentials

Run with specified account

Specify the identity under which this event executes.

[✓ K2 Server Service Account]

Specified Account:

Event Name:	ServerEvent
User Name:	Type text
	(domain\user)
Password:	
Security Label:	K2

If you're creating a new account for use here, K2 has to "know about" the new account. I believe K2 imports from Active Directory at 12-hour intervals, but an import may be forced by refreshing the K2 User Manager Cache. See reference (b) for details.

2. Create and Store the PowerShell Script

In a location on the server which hosts K2, open Notepad and type the following into a text document:

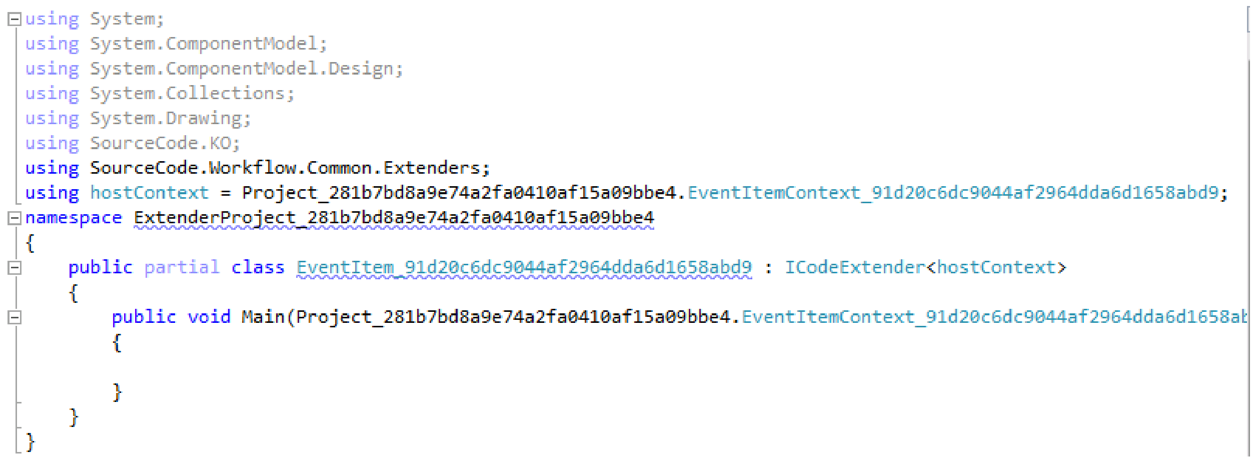
```
param([double] $fahrenheit)

##convert it to Celsius
$celsius= $fahrenheit - 32
$celsius = $celsius / 1.8

##output the answer
"$fahrenheit degrees Fahrenheit is $celsius degrees Celsius."
```

Save this file as **`Convert-Temperature.ps1`**. Carefully note the path you used for storing the file.

3. Code the Server Event

Okay. Time for some fun stuff.

Right-Click on the Server Event and select **View Code → Event Item** to get into some code. Here's an example of what you'll see by default:



```
using System;
using System.ComponentModel;
using System.ComponentModel.Design;
using System.Collections;
using System.Drawing;
using SourceCode.KO;
using SourceCode.Workflow.Common.Extenders;
using hostContext = Project_281b7bd8a9e74a2fa0410af15a09bbe4.EventItemContext_91d20c6dc9044af2964dda6d1658abd9;
namespace ExtenderProject_281b7bd8a9e74a2fa0410af15a09bbe4
{
    public partial class EventItem_91d20c6dc9044af2964dda6d1658abd9 : ICodeExtender<hostContext>
    {
        public void Main(Project_281b7bd8a9e74a2fa0410af15a09bbe4.EventItemContext_91d20c6dc9044af2964dda6d1658ab
        {

        }
    }
}
```

The system generated all of this – including all of that hex. Don't worry about copying it. Your focus is that **`Main()`** subroutine. It's going to contain the C# code necessary to create a runspace for your PowerShell script.

Add a Reference to PowerShell

The first thing you're going to need here is a reference to PowerShell.

Navigate back to your main project tab. Right-click anyplace on the canvas to bring up your workflow's general properties. Down the left side of the wizard, you should see several icons – I have eight of them, the last of them is shaded purple. **Click the second-to-last icon** to bring up the **Process References** pane.

Click **Add** to bring up the **Add References** dialog. You want to add a reference to **System.Management.Automation.dll.**

Create Input and Output Datafields

Before you get back into the code, let's visit the K2 Object Browser and **create a datafield** of type **Integer** called **`CurrentTemp`**. Set the default value to **32**. We're going to have our code read this value.

Next, **create another datafield** of type **String** called **`_ServerEvent-ConvertTemperature`**. We're going to have our code write to this variable, to allow us to monitor our code from the K2 Workspace.

Lay Some Code

Now go back into your server event and add:

```
using System.Management.Automation.Runspaces;
```

to reference the PowerShell DLL, and add the following code to **`Main()`**:

```
/* var to hold the output variable name */
const string outputVariable = "_ServerEvent-ConvertTemperature";
```

Now let's add a pair of constants to use for the path and name of that PowerShell script:

```
/* constants containing the location of the scripts and the name of the specific script*/
const string scriptpath = @"C:\PSScripts\";
const string file = "Convert-Temperature.ps1";
```

Time for our argument. In this case, we're passing in an integer for the value of the temperature in Fahrenheit.

```
/* arguments */
int fahrenheit;
```

We could set a value here from within the script (e.g., `int Fahrenheit = 32;`), but the exercise would probably be more useful to us if we used a value from one of the workflow datafields, like this:

```
int fahrenheit = (int) K2.ProcessInstance.DataFields["CurrentTemperature"].Value;
```

Eventually, you'll need a separator for your multiple arguments. Since PowerShell uses space to delimit these, let's create a constant for this.

Let's also create a string to hold our PowerShell command.

```
/* arguments separator */
const string separator = " ";

/* string containing the PowerShell command */
string cmd = "& ";
```

The ampersand tells PowerShell to execute the command that follows.

Let's assemble the `cmd` var to give it the script path and name.

```
/* add the script path */
cmd += String.Concat(scriptpath, file, separator);
```

Now let's append our argument.

```
/* add any arguments (separated by separator) */
cmd += String.Concat(fahrenheit);
```

If you had multiple arguments, you'd simply add them like this:

```
cmd += String.Concat(arg1, separator, arg2, separator, arg3, separator, arg4);
```

The PowerShell Runspace

Now you need some runspace goop. This is the code you'll need to create the actual PowerShell runspace.

```
/* runspace goop */
Runspace runspace = null;
Pipeline pipeline = null;

try
{
    runspace = RunspaceFactory.CreateRunspace();
    runspace.Open();
    pipeline = runspace.CreatePipeline();
    pipeline.Commands.AddScript(cmd);
    var results = pipeline.Invoke();

    foreach (var obj in results)
    {
       //consume the results
       K2.ProcessInstance.DataFields[outputVariable].Value = obj.ToString();
    }
}

catch (Exception ex)
{
   K2.ProcessInstance.DataFields[outputVariable].Value = ex.ToString();
   throw;
}
```

```
finally
{
   if (pipeline != null) pipeline.Dispose();
   if (runspace != null) runspace.Dispose();
}
```

The code creates Runspace and Pipeline objects, and your `cmd` var is passed to the pipeline instance. An output variable called `results` receives the output from the pipeline – or, more directly, the output of your Convert-Temperature function. We have a value for `outputVariable` which is the name of a datafield, which is where the `results` value – or, if something's gone south, any exceptions – will go. Finally, we dispose of the Runspace and Pipleline instances.

The content of the **`Main()`** function follows. (Don't forget to add the reference to the runspaces namespace!):

```
     {
        /* var to hold the output variable name */
        const string outputVariable = "_ServerEvent-ConvertTemperature";

        /* constants containing the location and name of the script*/
        const string scriptpath = @"C:\PSScripts\";
        const string file = "Convert-Temperature.ps1";

        /* arguments */
        int fahrenheit = (int) K2.ProcessInstance.DataFields["CurrentTemperature"].Value;


        /* arguments separator */
        const string separator = " ";

        /* string containing the PowerShell command */
        string cmd = "& ";

        /* add the script path */
        cmd += String.Concat(scriptpath, file, separator);

        /* add any arguments (separated by separator) */
        cmd += String.Concat(fahrenheit);

        /* runspace goop */
        Runspace runspace = null;
        Pipeline pipeline = null;

        try
        {
            runspace = RunspaceFactory.CreateRunspace();
            runspace.Open();
            pipeline = runspace.CreatePipeline();
            pipeline.Commands.AddScript(cmd);
            var results = pipeline.Invoke();

            foreach (var obj in results)
            {
                //consume the results
                K2.ProcessInstance.DataFields[outputVariable].Value = obj.ToString();
            }
        }
        catch (Exception ex)
        {
            K2.ProcessInstance.DataFields[outputVariable].Value = ex.ToString();
            throw;
        }

        finally
        {
            if (pipeline != null) pipeline.Dispose();
            if (runspace != null) runspace.Dispose();
        }
    }         //public void Main…
  }           //public partial class EventItem…
}             //using hostcontext…
```

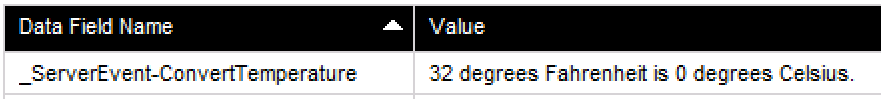
(I added the comments above to help clarify what brace was for what.)

4. Build and Deploy

If you've created a new activity to house the Server Event, make sure it's connected to the rest of your workflow. **Save**, **compile**, and **deploy** to your development environment.

Run your workflow, then consult the **Process Instance Data** in your **Workspace**.

Here's the output I received.



Data Field Name	Value
_ServerEvent-ConvertTemperature	32 degrees Fahrenheit is 0 degrees Celsius.