## Copyright

This document is provided for informational purposes only and Software FX, Inc. makes no warranties, either express or implied, in this document. Information in this document, including URL and other Internet Web site references, is subject to change without notice. The entire risk of the use or the results from the use of this document remains with the user. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Software FX, Inc.

Software FX, Inc. may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Software FX, Inc.; the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

# Table of Contents

## Definitions and Conventions used in this Quick Start Guide

Before you can start using PowerGadgets you must get acquainted with Windows PowerShell scripts, to do this you must first invoke a PowerShell prompt. After installing PowerShell a menu option (Windows PowerShell) will appear on the Start Menu, when this option is selected a PowerShell prompt appears, as follows:
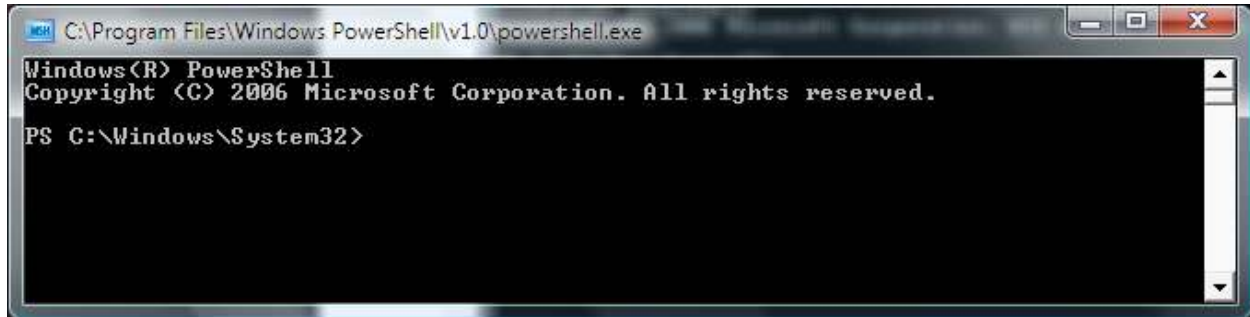


In order to facilitate the reading of this guide, all samples and scripts that must be written in a PowerShell prompt will be highlighted with a black background without any prompt symbols.

Also in this Guide, you will find the term "cmdlet" and "snapin" used frequently.  A cmdlet (pronounced "command-let") is the smallest unit of functionality in the Windows PowerShell and it is directly analogous to the built-in commands in other shells.  In a traditional shell such as Cmd.exe or KSH, commands are executable programs that range from the very simple (such as Attrib.exe) to the very complex (such as Netsh.exe).  With the Windows PowerShell, most commands are very simple and very small, hence the term cmdlet.
A cmdlet is referred to by a verb and noun pair, separated by a "-". For example:

```
get-process
```

The Windows PowerShell provides a new conceptual model for "piping" or command-to-command communication that is based on *objects*, rather than text.  The objects that cmdlets emit allow other cmdlets to act directly on the properties and methods of the object.  This piping concept is the main functional feature that PowerGadgets cmdlets take advantage of to obtain data for a gadget.  This means that before invoking a PowerGadgets cmdlet that creates a visual gadget you must first obtain data from a desired source. For example if you want to create a chart of the processes in your machine, you will type the following script in a PowerShell prompt:

```
get-process | out-chart
```

## What is PowerGadgets?

PowerGadgets is a revolutionary new data visualization product that leverages PowerShell, Microsoft's new scripting shell for Windows, to allow the creation of gadgets in Windows XP and Windows Vista. PowerGadgets requires no development environment or compilation plus its deliverables can be viewed directly on the Windows desktop without requiring a browser or any other container making PowerGadgets the perfect reporting and monitoring solution for IT and DB Professionals who don't write code.

In addition, PowerGadgets is visually and programmatically rich – scaling from vector-based graphics and managed .NET code - making it the perfect enterprise tool to report and monitor data  from many sources, including but not limited to SQL Server, ODBC, File System, Windows Management Instrumentation (WMI), Processes and even the Windows Registry. You can even use PowerGadgets to retrieve information from other Microsoft applications that are built on top of PowerShell such as Exchange Server (12), MOM and Virtual Machine Manager.

This Quick Start Guide will introduce you to the most interesting product features with basic PowerShell scripts and samples. However, be aware Microsoft PowerShell is a rich, robust, and extensible command-line environment from where you can write complex scripts enabling PowerGadgets to retrieve information from data sources not mentioned in this document. For additional information on PowerShell please visit PowerShell's web site at [www.microsoft.com/powershell](www.microsoft.com/powershell).

Although the goal of PowerShell is to enable the administration of the Windows platform, tasks that are normally carry out by  IT administrators,  PowerGadgets also includes utilitarian cmdlets that extend the reach of PowerShell beyond the IT management tasks on which you would normally use PowerShell. Therefore, if you are a developer or a DB professional, we encourage you to also take a look at this extended functionality as a way to use PowerGadgets to create the most powerful and flexible dashboards for your existent applications and databases.  For example, PowerGadgets provides utilitarian cmdlets that let you easily connect to a SQL Server or an ODBC data source for the purpose of extracting data from a database to populate gadgets.  This can be done with a single line command in PowerShell or through a powerful UI provided by PowerGadgets. For additional information please refer to "utilitarian cmdlets" later in this document.


## PowerGadgets Requirements

PowerGadgets requires minimal software to run. The PowerGadgets installer also has a small footprint and it can be easily installed and deployed across the enterprise.  Before you install PowerGadgets, please make sure you have installed:

- .NET Framework 2.0
- Windows PowerShell RC2/RTW(1.0.9567.1 or above). Available at [www.microsoft.com/powershell](www.microsoft.com/powershell).
- Windows XP, Windows Vista Windows Server 2003 or Windows Server "Longhorn".

## Adding  PowerGadgets to Windows PowerShell

PowerGadgets provides a fully automated installer so you can start creating gadgets straight from a PowerShell prompt. However, before you start using any of the cmdlets provided by PowerGadgets  it is important you understand PowerGadgets is a PowerShell Snapin that  must invoked every time you open a new PowerShell prompt, as follows:

```
add-pssnapin PowerGadgets
```

For your convenience, the PowerGadgets installer  creates a menu option can be found in the Windows Start button that starts PowerShell with the PowerGadgets Snapin already added, as depicted in the following picture:



## Changing the PowerShell Default Execution Policy

PowerShell is, by default, a secure environment.  Therefore, by default, scripts are not enabled for execution.  In order to properly use PowerGadgets and its accompanying cmdlets you must enable scripts for execution. Since PowerGadgets is digitally signed, you can allow script execution by enter any of the following commands at the shell prompt:

```
set-executionpolicy allsigned
```
This command sets the execution policy to require that all scripts must have a trusted signature to execute.  If you would like a less restrictive environment, you can enter the following command:

```
set-executionpolicy RemoteSigned
```
This command indicates that the shell will execute scripts downloaded from the web only if they are signed by a trusted source.  The least secure execution policy may be set as follows:

```
set-executionpolicy unrestricted
```
This command sets the execution policy to run scripts regardless of whether they have a digital signature.

# PowerGadgets core cmdlets

Once you have provisioned your computer with Windows PowerShell and PowerGadgets you can start creating visually rich gadgets in Windows Vista and Windows XP. Since these operating systems handle gadgets in a very different way, we will describe the process for each, with screen shots and operating system specifics whenever possible. However, you should know PowerGadgets provides 3 main cmdlets:

**Out-chart:** PowerGadgets provides a powerful charting engine providing over 60 chart types and a wealth of aesthetical and functional features. A chart plots data in the form of series and points with each series containing one or more points.

**Out-gauge:**  In addition to charts, PowerGadgets provides a out-gauge cmdlet to display single variable graphical objects. Single variable analysis is very powerful when combined with PowerGadgets' flexible real-time capabilities, explained later in this document. PowerGadgets supports the following gauge representations:



| | | |
|---|---|---|
| Radial Gauges are the most common and conventional type of gauge. These gauges are normally used to represent a quantifiable process progressing through the possible range displayed on a scale with a circular or semi circular aspect. | Linear Gauges are normally used to represent the progress of a process or a specific value within a continuous range and fall into two categories, horizontal and vertical. Common examples would be a ruler (horizontal) or a thermometer (vertical). | Digital Panels are used for both numeric and alphanumeric displays. These panels are commonly prevalent in the world such as watches, registrars, information kiosks, alarm clocks, etc. The most common are 7-segment, 14-segment and LED. |

**Out-map:** PowerGadgets also provides a powerful mapping product that allows you to depict enterprise data on geographical maps or any other vector image, as depicted in the following picture:
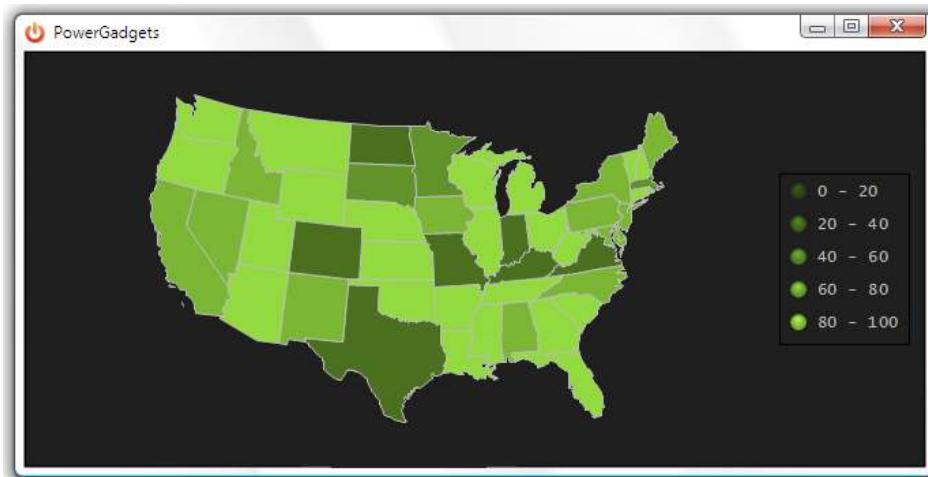
## Creating and Displaying your first Gadget in Windows XP

There is no "official" gadget concept in Windows XP. Therefore, we will define a Windows XP gadget as visual element that floats on your desktop.  A gadget can be viewed as a cascading window or as floating object (gadget) in your desktop and can be authored via a PowerShell script or through the PowerGadgets Creator.

Creating a gadget in Windows XP via PowerShell scripting is as easy as:

1) Determine the type of gadget you want to display on your desktop (chart, gauge or map)
2) Determine the windowing characteristics of the gadget (size, position)
3) Write a PowerShell script that extracts the data (scripting)
4) Pipe the PowerGadgets cmdlet with the appropriate characteristics
5) Customize the gadget through templates (configuration)

We will describe this process with a specific sample: Machine Processes. In Windows PowerShell it is extremely easy to access the processes in the system. When invoked the get-process cmdlet generates a table of the processes being executed in the machine, as follows:
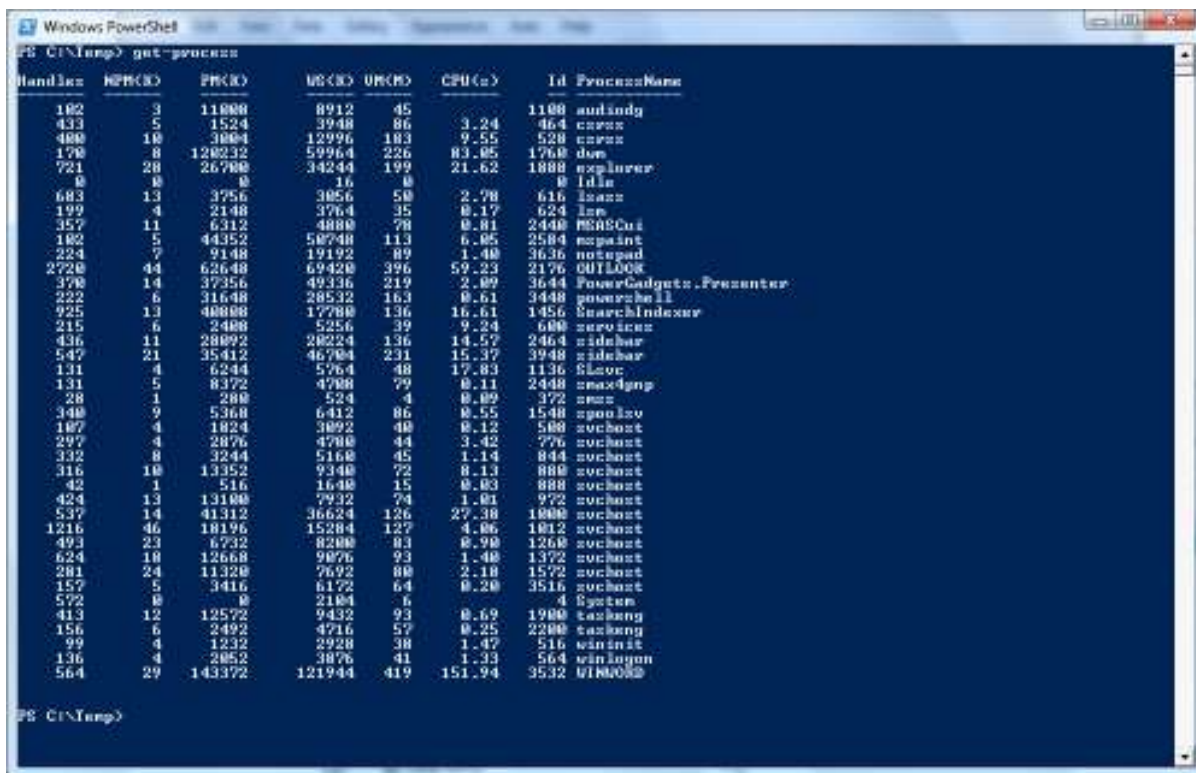


This information is much more readable through a chart where numbers are plotted as different series in a bar, line or area chart. To do this you can simply "pipe" the out-chart cmdlet to the previous script, as follows:

```
get-process | out-chart
```

Also, when you use any of the PowerGadgets cmdlets (out-chart, out-gauge, out-map), gadgets will be created as modeless windows on the desktop. This is to allow PowerShell to continue execution of the pipe and maintain the gadget alive on the desktop.

When creating gadgets, you can use the –floating switch to create gadgets in your Windows XP desktop.
The following PowerShell script displays floatable gauge measuring the system available memory:

```
get-wmiobject Win32_PerfRawData_PerfOS_Memory | out-gauge -value AvailableMBytes –floating
```

this gauge will be displayed in a Windows XP desktop as follows:
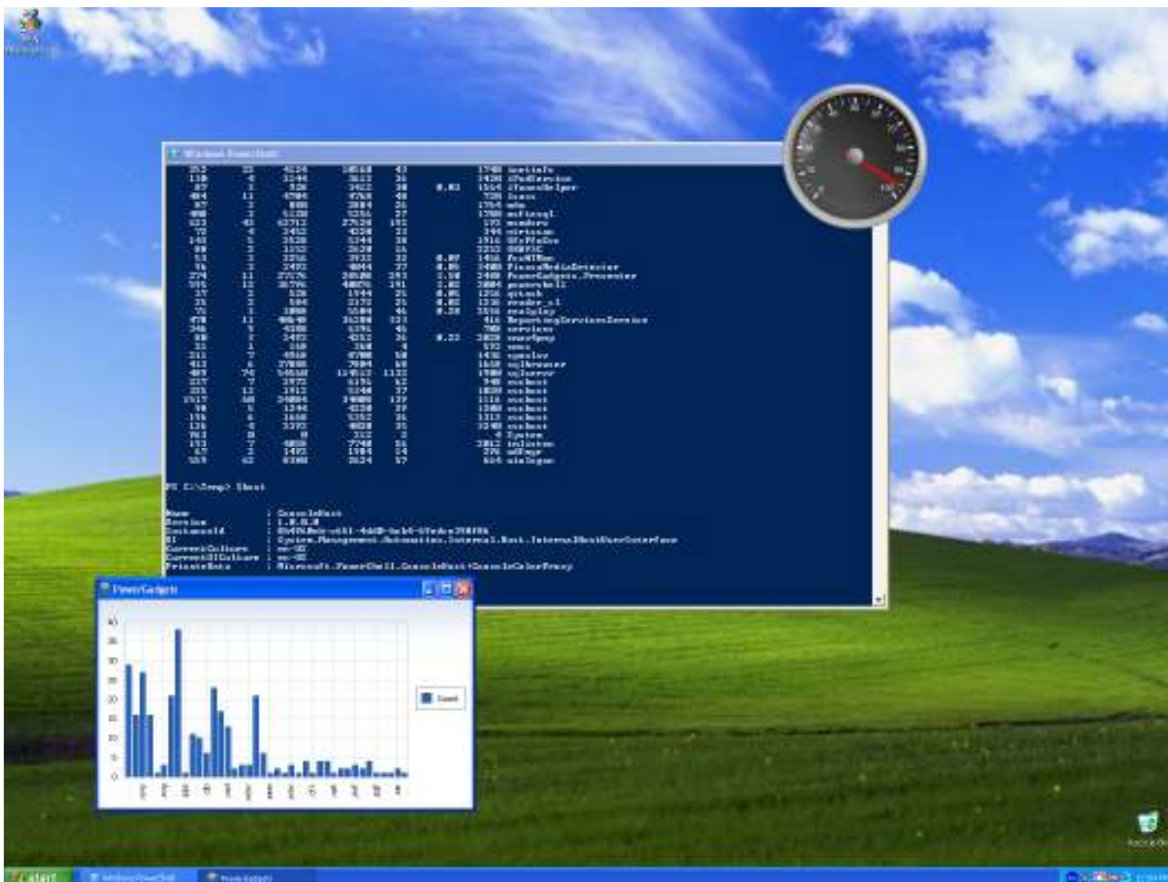


Furthermore, these visual elements can be easily sized and positioned on the desktop using the **–size, -location** and **–position** switches in your PowerShell script. For example, If you want the previously displayed gauge sized as a 150,150 pixels and anchored to the upper-right corner of the desktop, separated 10 pixels from the margin, you would write the following PowerShell script:

```
get-wmiobject Win32_PerfRawData_PerfOS_Memory | out-gauge -value AvailableMBytes –modeless –floating –size
150,150 -anchor topright –location 10,10
```

Finally, please note you can quickly wrap this functionality into PowerShell script files (.ps1) and distribute them to PowerGadgets enabled systems for easy deployment of  gadgets and dashboard elements. No browser or compilation required! For additional Deployment and Licensing considerations please contact us at sales@softwarefx.com

## Creating and Displaying your first Gadget in Windows Vista

As opposed to Windows XP, Windows Vista not only defines the notion of a gadget, but also provides a specific facility to manage these gadgets: the Windows Sidebar. When installed on Windows Vista, PowerGadgets also installs the necessary files to create Windows Vista gadgets in windows Sidebar.

> Note: To activate the Windows Vista Sidebar click the Start button, then click on All Programs. The Windows Sidebar can be found under the "Accessories" folder.

This process should be familiar to any Windows Vista user.

1) Open the Windows Vista Sidebar
2) Click on the plus (+) sign located on top of the side bar
3) From the gadgets window, click on the PowerGadgets icon as many items as different PowerGadgets you want to create on the Windows Sidebar (or simply drag the PowerGadgets icon to its corresponding place in the Windows Vista Sidebar). Each Gadget will have a unique identifier for you to reference in your PowerShell scripts (gadget1,gadget2 and so on). These names can be changed on the Settings dialog of every gadget displayed in Sidebar.



4) Finally, you can choose to populate a gadget in the Windows Sidebar via PowerShell scripting or by simply loading a gadget file (.pgf) created with the PowerGadgets Creator.

## Populating a Windows Vista Sidebar Gadget via PowerShell scripting

To add PowerGadgets to the Windows Sidebar simply double-click on the PowerGadgets icon and a placeholder gadget will appear in Sidebar. The final step in making a gadget functional is to write a PowerShell script adding the –sidebar switch with the identifier of the gadget you want to use. For example, to create a chart with the machine processes, you can write the following script:

```
get-process | out-chart –sidebar gadget1
```

This script will bind the data and populate gadget1 in Sidebar, producing the following chart:



Note: PowerGadgets supports a powerful storyboarding capabilities so you can group gadgets gadgets in the Windows Sidebar, for additional information on Storyboarding please refer to the following pages in this guide.

## Authoring a Gadget via the PowerGadgets Creator

PowerShell scripts provide a strong foundation and flexible way to create gadgets in your Windows Vista and Windows XP systems. If you have not yet installed Windows PowerShell or if you are populating gadgets from databases or web services the PowerGadgets Creator may be an easier way to create and distribute gadgets.

The PowerGadgets Creator is a fully functional GUI that lets you specify not just the gadget visual attributes but its data sources as well, a sample screen shot is provided below:
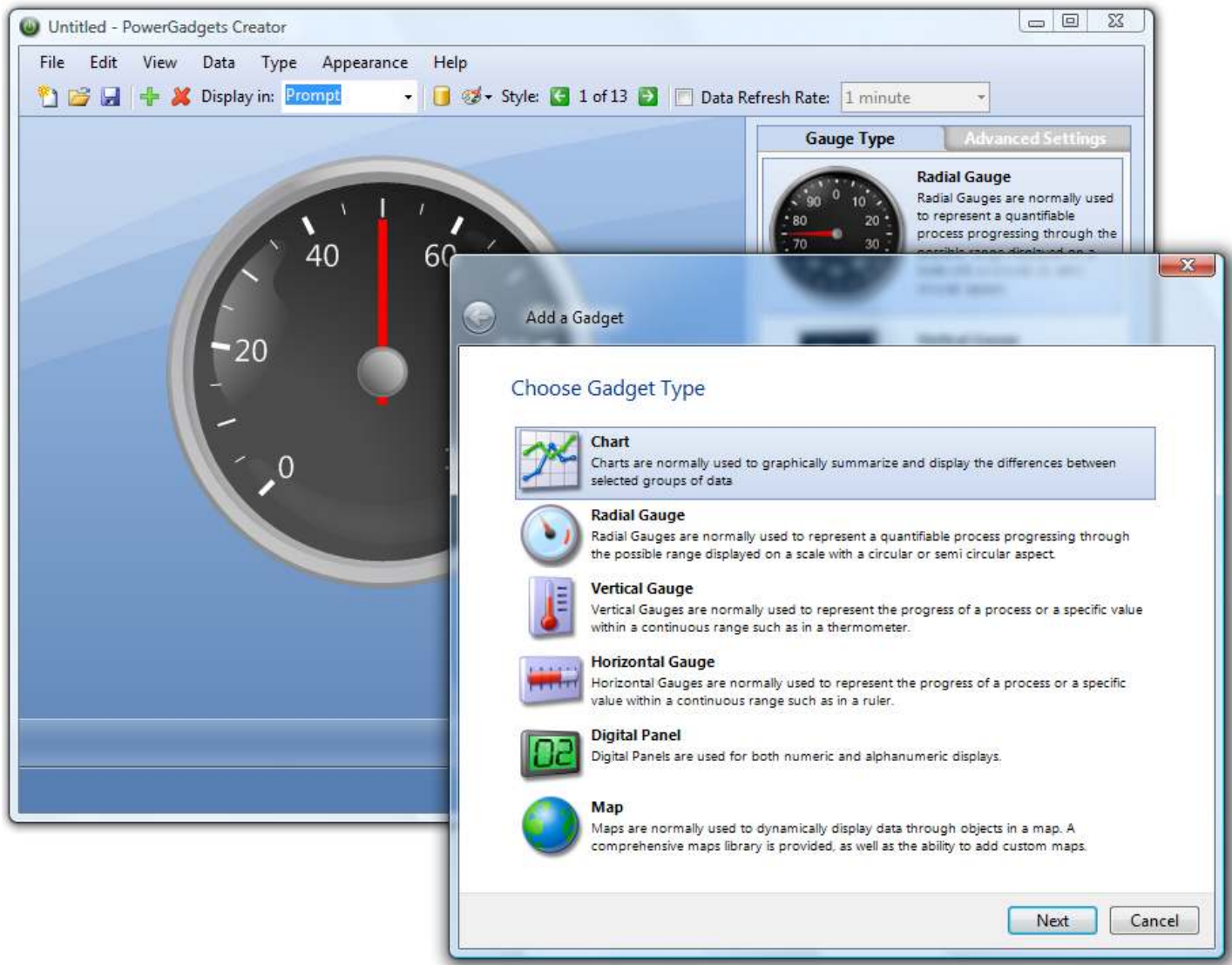


Note: Please refer to the PowerGadgets Creator chapter later in this manual for additional information, benefits  and features of this tool.

**Expanding a Sidebar Gadget**

Once a gadget has been created and bound to data, you can expand the gadget by dragging it outside the boundaries of the Sidebar, when you release the mouse the gadget will increase its size showing more chart elements like legends, axis labels as well as a fully functional toolbar that enables users to change the chart aesthetics.

Because this chart is a live object (not a picture) you can click and position the mouse over different chart elements to perform complex data analysis. For example if you have a multiple series chart you can highlight series by hovering the mouse over a legend item or the series in the chart area. You can also position and hold the mouse over an axis label until a fixed line appear showing you the values that plot outside that range, you can even interactively move this line, as depicted in the following picture:
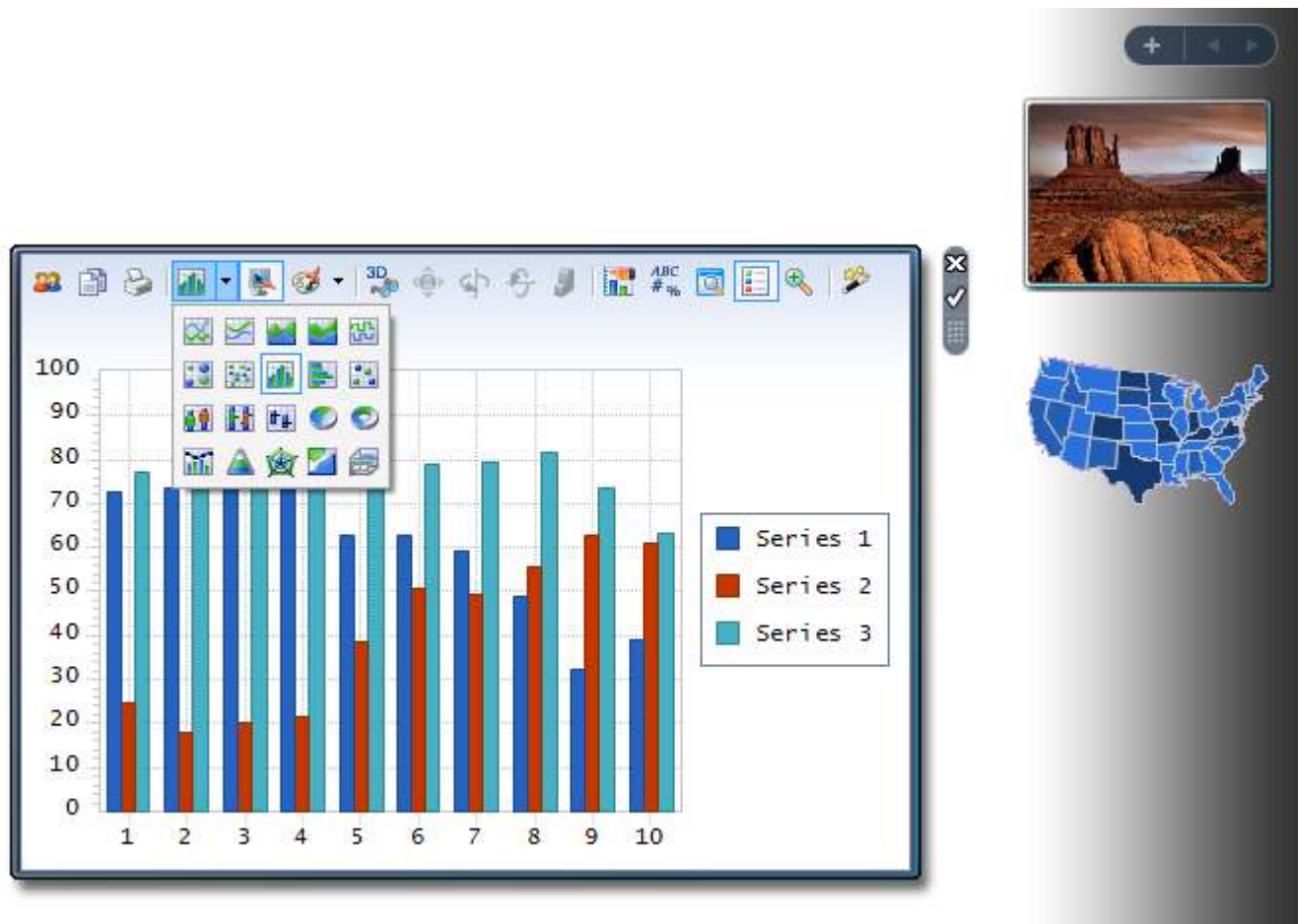


Just like this, you can change colors, print and even share these gadgets with other productivity applications like Microsoft's Word, Excel and PowerPoint.

## PowerGadgets Flyout and end-user customization capabilities.

Once a gadget has been created and bound to data, you don't necessarily have to expand the gadget outside the boundaries of the Sidebar to explore its data. Each gadget displayed on windows Sidebar is an interactive object. To enable this interactivity simply click on the Sidebar gadget and then move the mouse over any chart marker (bar or data point) and after a few seconds a powerful highlighting capability will automatically activate the PowerGadgets Flyout capabilities, as depicted in the following figure:
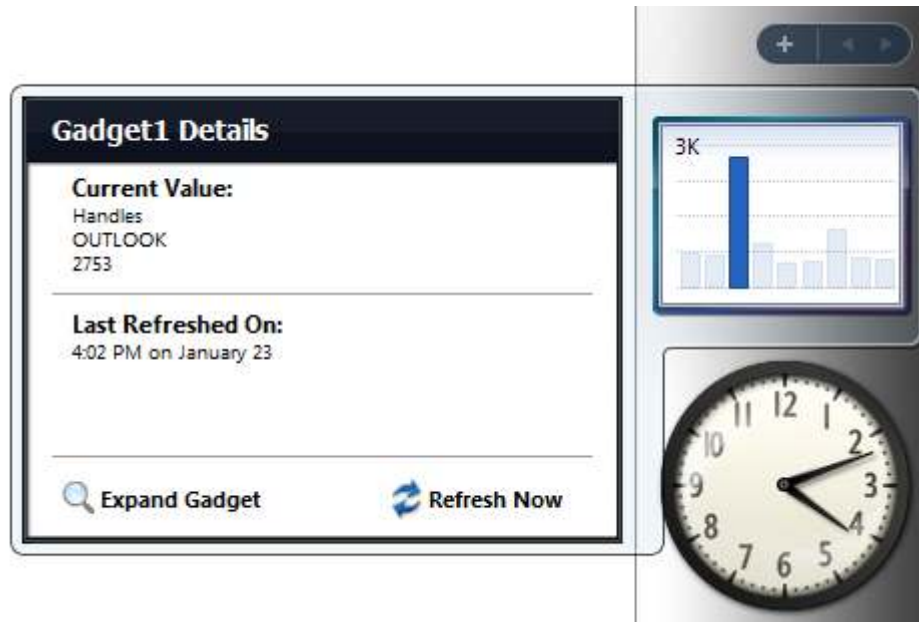


To deactivate and close the flyout window, simply click outside the gadget and its flyout boundaries.

Note: The text to be shown in the Flyout window can be customized through the TooltipMask (chart) and the mainindicator_tootltip properties (gauges). Please note you can also set these properties via the PowerGadgets Creator properties list. For example to customize the flyout for a gauge gadget, you can type the following script:



```
out-gauge –mainindicator_tooltip "My value is %v"
```

This script will produce the flyout display shown.

In addition to flyout capabilities, gadgets also allow end users to customize its visual attributes as well as refreshing rates. To activate simply right-click a gadget and select the "Options" menu:

Once the option menu has been chosen, end users can select the following options:

Color Palettes: change gadget colors to different color schemes.
Real-time data refresh: will allow end users to modify the rate set by the gadget author.
Customize and add new items to gadget: will invoke the PowerGadgets Creator and allow user to change individual gadget attributes. Please refer to PowerGadgets Creator chapter later in this manual.
Gadget File: load another gadget file (.pgf) in that sidebar position. If you have received a gadget file (.pgf) you can load it using this option.

Please note these changes will persist, even if Windows Sidebar is closed or if a Windows Vista reboots occur.
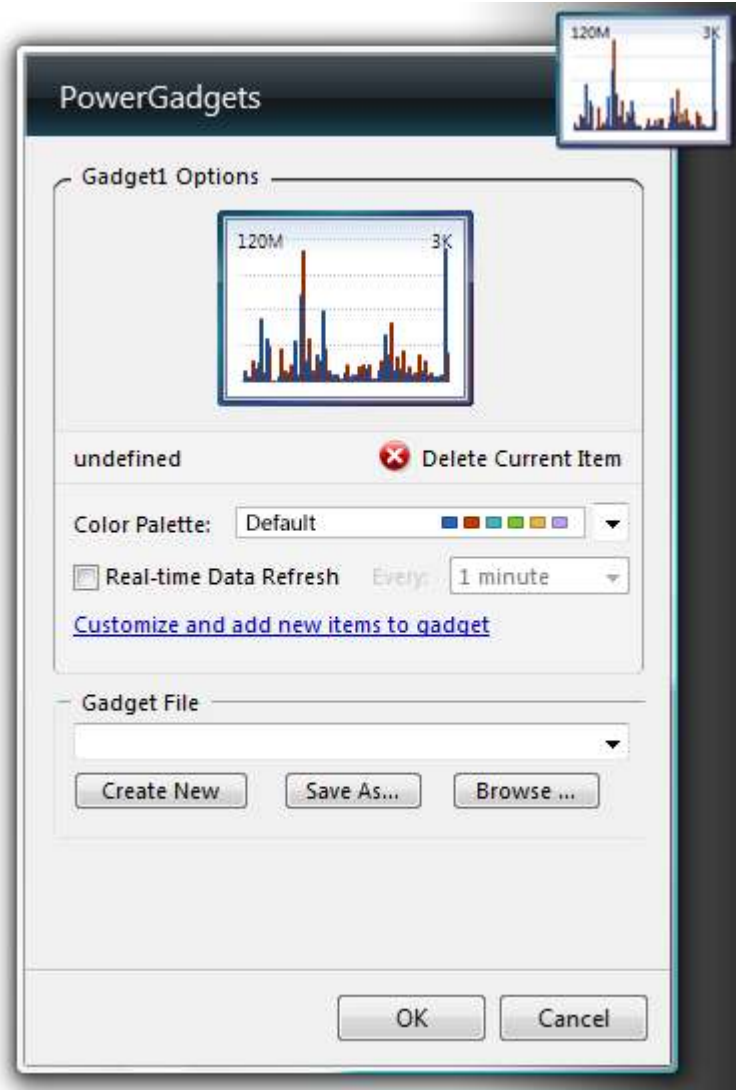
# Invoking PowerGadgets Wizards from Windows PowerShell to create Gadget templates

PowerGadgets exposes the same design-time experience found in much more graphical design surfaces like Visual Studio 2005. Just add the -configure switch to any of your PowerGadgets cmdlets and a powerful graphical Wizard that allows simple customization of chart, map and gauge settings. When used, a configuration file (template) is saved (My Documents folder) and you can then use the –template switch to load and apply these settings quickly and easily. To invoke a wizard. It is advisable that you invoke the –configure switch with the –template switch and provide a template name.

```
get-process| out-chart –configure –template MyChartTemplate
```

this script will give you access to the PowerGadgets wizard which provides a GUI for quickly customizing the chart elements. Please note the two tabs provided by this application, the first provides access to a much more graphical wizard and the second provides easy access to the properties of the chart control:
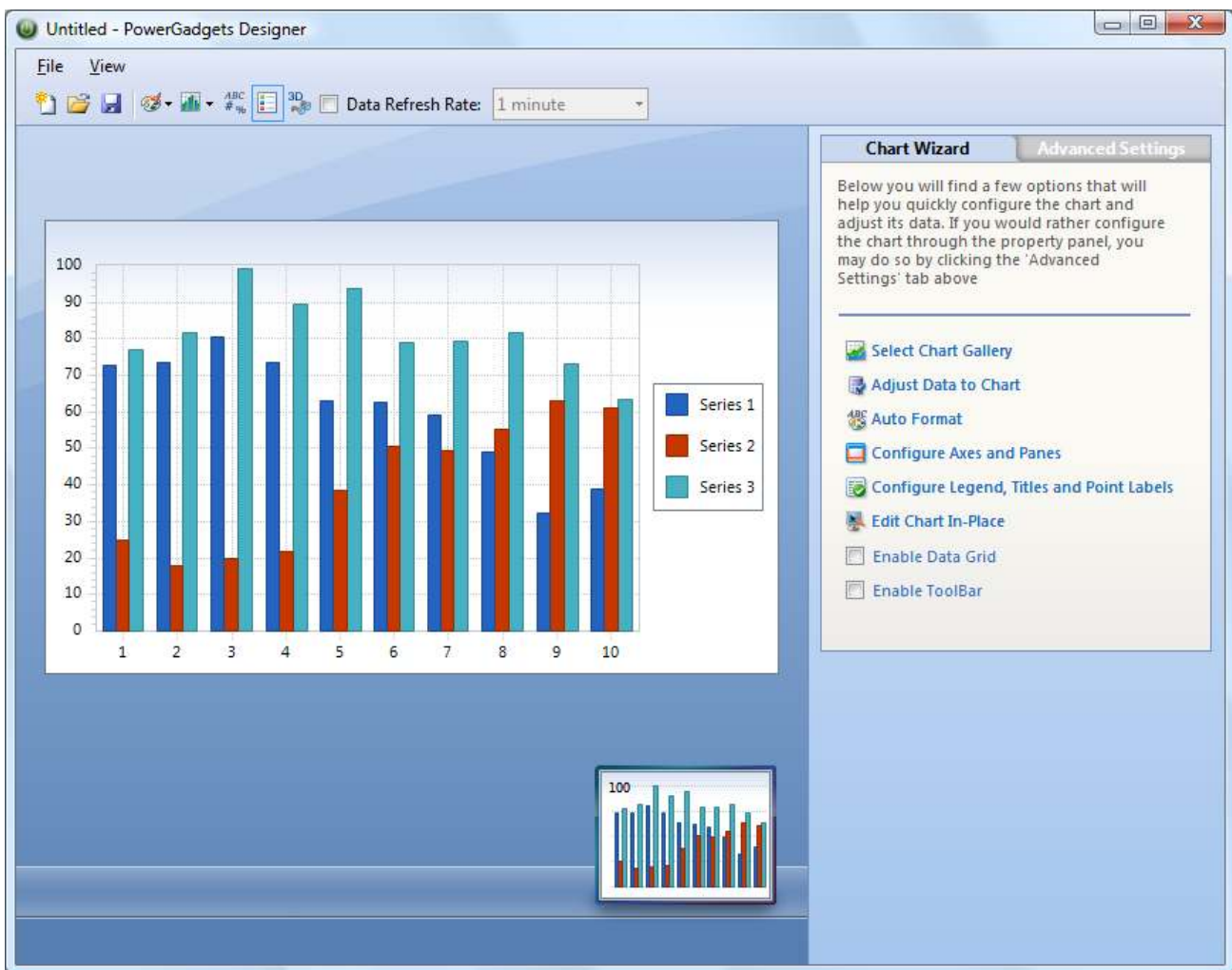


As you manipulate properties, the chart will display the changes and you can save these settings and reuse them to overwrite default values applied by PowerGadgets.

The out-gauge cmdlet also provides a –configure setting, which provides a fully graphical properties list. For example if you type:

```
get-wmiobject Win32_PerfRawData_PerfOS_Memory | out-gauge -value AvailableMBytes –configure –template MyGaugeTemplate
```

this script will give you access to the PowerGadgets Gauge wizard:
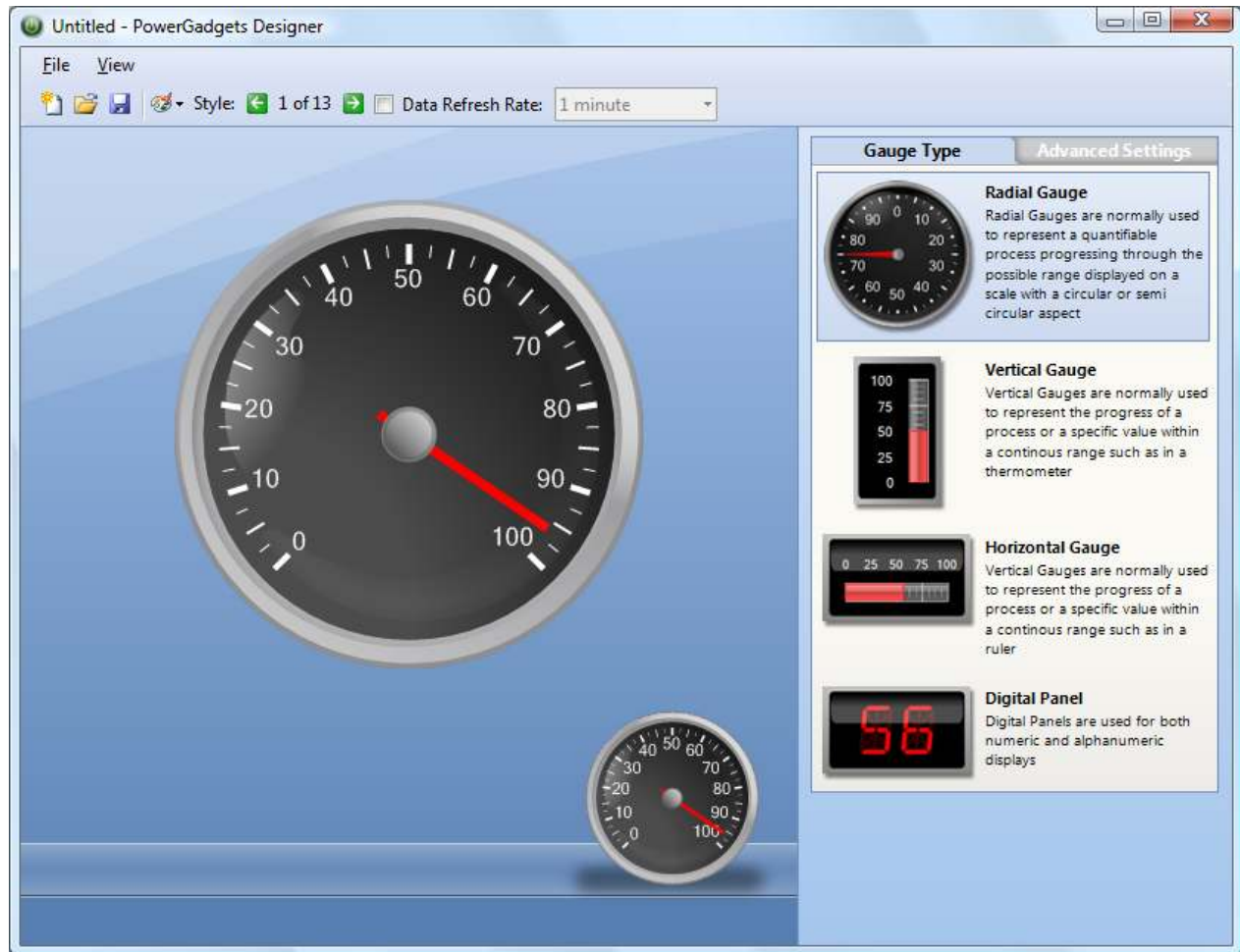


After properly configuring the chart or gauge to your visual preference, you can simply remove the –configure switch to load and apply the template.

```
get-process | out-chart –template MyChartTemplate
```

or for the gauge sample introduce before, you simply type:

```
get-wmiobject Win32_PerfRawData_PerfOS_Memory | out-gauge -value AvailableMBytes –template MyGaugeTemplate
```

Out-map also provides a graphical configuration option :
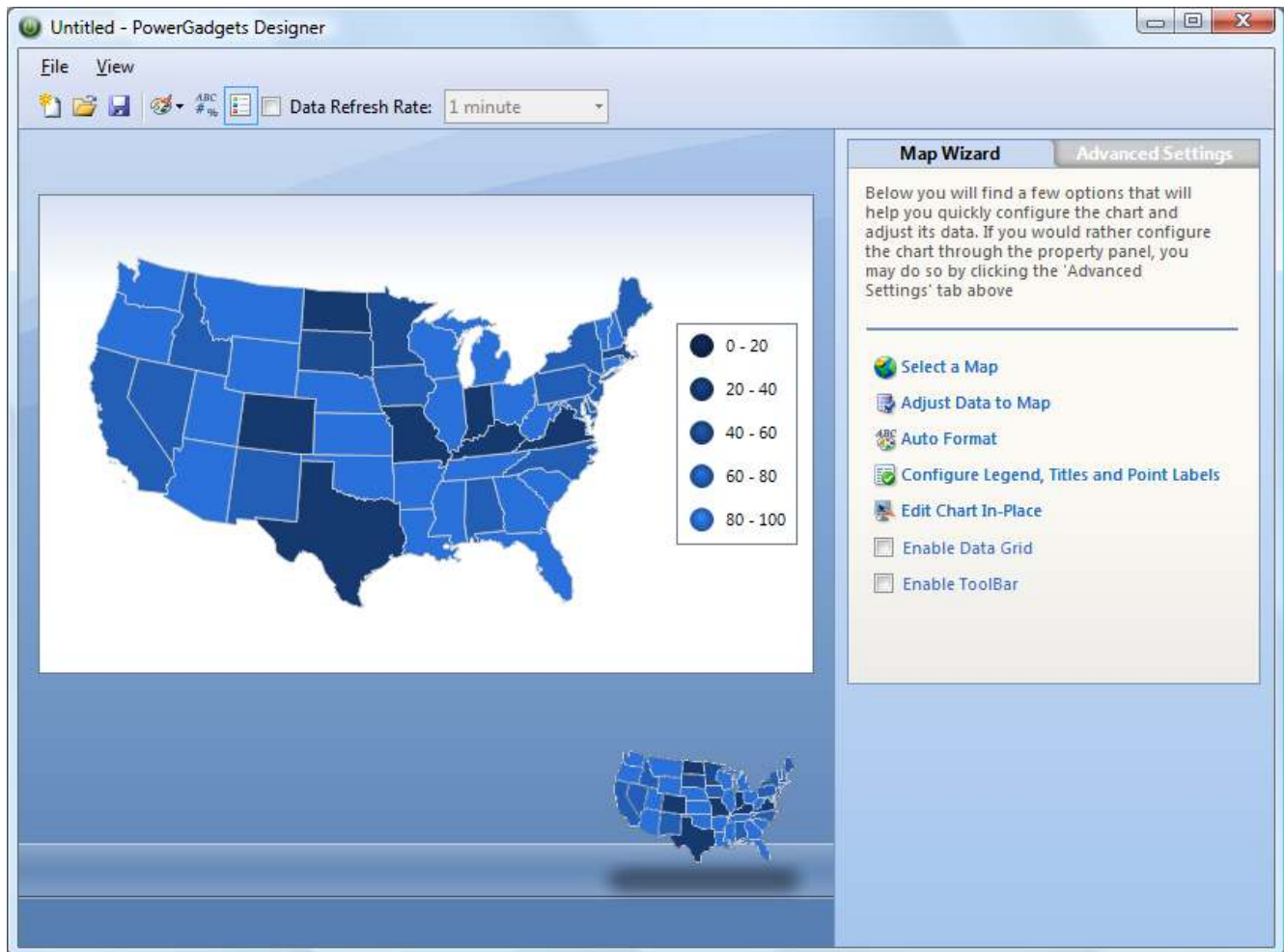


After properly configuring the map to your visual preference, you can simply remove the –configure switch to load and apply the template.

```
out-map –template MyMapTemplate
```

Please refer to the "PowerGadgets out-map cmdlet Scripting FAQ" later in this guide for additional information and parameters for the out-map cmdlet.

## Mutliple Gadgets & Storyboarding in Windows XP

By using the –group and –name switches in out-chart, out-gauge or out-map cmdlet you can group gadgets and prevent unnecessary desktop or sidebar cluttering when multiple gadgets are being generated. PowerGadgets will add a navigation toolbar so end users can play or select specific gadgets in a given group.

The following scripts creates 2 different charts in the same group:

```
ps | out-chart –group Gadget1 –name Process –title "Processes"
dir *.* | out-chart –group Gadget1 –name Files –title "Directory Files"
```
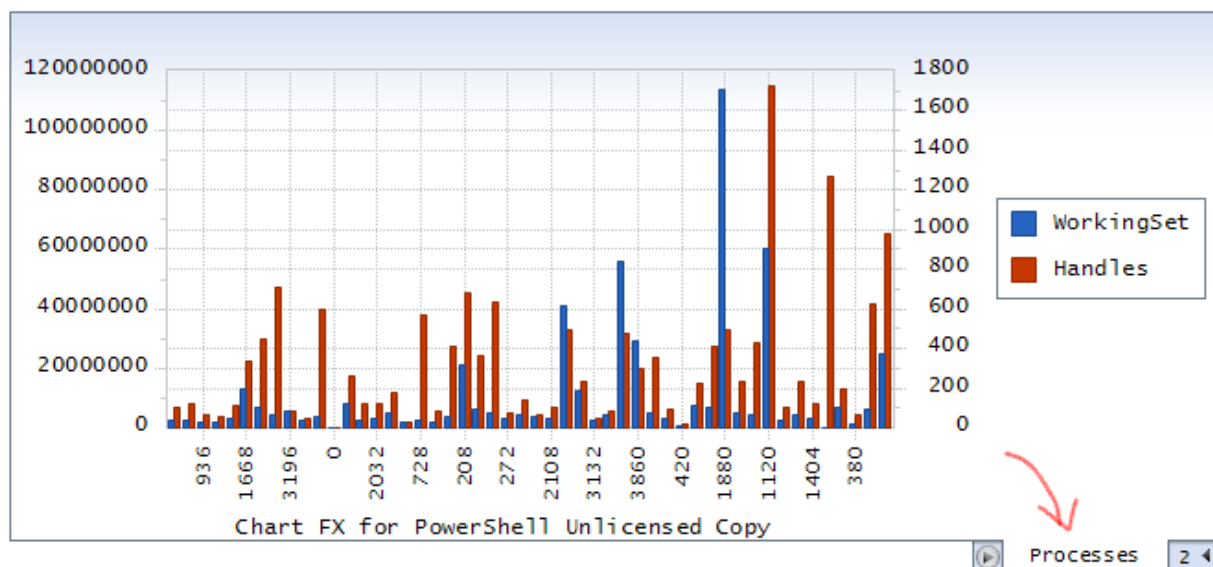


Please note you can author a PowerShell script (.ps1 file) that creates a group of gadgets to be displayed on your desktop.

Note: This feature is also supported in Windows Vista. However, if you want to create groups in Windows Vista Sidebar gadgets you will need to use different switches (see below).

## Mutliple Gadgets & Storyboarding in Windows Vista Sidebar

PowerGadgets has the ability to be started multiple times by the user, so that they could have many different gadgets in the Windows Sidebar.  However, even though you can add as many PowerGadgets in your Windows Vista Sidebar, you will want to save space for other gadgets or simply group gadgets that display data in a given category. This can easily be done by creating groups and storyboards in a single Sidebar gadget. When a group of PowerGadgets is created a VCR interface is displayed so end users  can navigate through different gadgets contained in the group.

Every time you add a PowerGadgets gadget to the Windows Vista Sidebar, you must provide a unique ID to the gadget. In order to create a group in a Sidebar gadget you must use the gadget's ID in conjunction with the -name switch to uniquely identify each gadget in a group.  For example, the following PowerShell script creates a group in a Windows Vista Sidebar gadget, assuming the default <gadget1> ID was not changed:

```
ps | out-chart –sidebar gadget1 –name Process –title "Processes"
dir *.* | out-chart –sidebar gadget1 –name Files –title "Directory Files"
```

**Adding additional gadgets to a group**
There are two ways of adding a gadget to an existent group. The first option is to use a different –name on a given ID. For example, the following script adds a gadget to the <gadget1> used before:

```
Out-map –sidebar gadget1 –name MyMap –title "Sales Map"
```

The second option is to simply use the –add switch, as follows:

```
Out-map –sidebar gadget1 –add
```

**Replacing gadgets in a group**
Replacing an existent gadget is a simple as using the same –name identifier for the gadget you want to replace in a group. For example, the following script replaces the "Directory Files" with a map.

```
Out-map –sidebar gadget1 –name Files –title "Sales Map"
```

**Resetting a group**
You can easily reset a gadget group by adding the –reset switch to your PowerShell script. For example, the following script creates a chart with the machine processes and resets the gadget groups, so only that gadget will be displayed:

```
ps | out-chart –sidebar gadget1 –reset
```

## PowerGadgets Real-time refreshing capabilities

PowerGadgets provides Real-time serialization and execution of the entire PowerShell command pipe without requiring additional PowerShell Prompts; making PowerGadgets the perfect solution to integrate gadgets and other independent data visualization elements for advanced monitoring capabilities.

You can quickly inspect PowerGadgets' real-time capabilities by using the get-date cmdlet available and PowerShell to create a digital watch using PowerGadgets's digital panel display that updates every second, as follows:

```
get-date | out-gauge –floating –refresh 0:0:1 –type digital
```

This realtime capability can be used in more complex scenarios. For example, you could monitor cpu usage graphically by creating a PowerShell script that uses a wmi-object and PowerGadgets as follows:

```
Get-wmiobject –class "Win32_PerfFormattedData_PerfOS_Processor" | where {$_.Name –eq "_Total"} | out-gauge –value PercentProcessorTime –floating –refresh 0:0:1
```

Gauges are single variable displays that can be used to create powerful real-time graphical representations from pipelined data from sources like registry, services, processes, Windows Management Instrumentation, event logs, and much more. However, charts also provide additional graphical pizzazz by providing a history of real-time information. For example, the Windows Performance monitor uses a chart (not a gauge) to display the history of CPU utilization. When creating realtime charts you must take into consideration the amount of data to be kept in the chart at one time, this is called the buffer size. By default, PowerGadgets can hold up to 20 points at any given time, however, you can overwrite the buffersize by using the realtime_buffersize switch.

```
Get-wmiobject -class Win32_PerfFormattedData_PerfOS_Processor | where {$_.Name -eq "_Total"} | out-chart  -values PercentProcessorTime -refresh 0:0:1 -realtime_buffersize 100
```

As a fully real-time enabled technology PowerGadgets allows you to control the type of real-time charts you want to create with PowerShell. You can create two types of realtime charts:

1) when the buffer is full, the chart will display a radar-like line and update the current point
2) the chart will scroll every time a new point is added to the chart.

By default, the chart will be set in scroll mode. However you can use the –realtime_mode to change this setting. For more information, please refer to the PowerGadgets API documentation located in the CHM Help file provided with the software.

## Populating PowerGadgets from Databases

PowerGadgets and its real-time and gadget features make it the perfect monitoring solution. Unfortunately, there's no support for LINQ or any other database connectivity in mechanism in PowerShell hindering the ability of other users, such as DBAs and Developers from using PowerShell as a powerful data extracting tool.  For this reason, PowerGadgets includes a versatile cmdlet named **invoke-sql**, which allows PowerShell to connect to a SQL Server or any other ODBC compliant database and pull information for data visualization purposes. To run a SQL Query to connect to SQL Server and create a gadget with meaningful database information you can use the following PowerShell script and "pipe" the appropriate PowerGadgets cmdlet:

```
invoke-sql –server MYSQLSERVER –database DB –sql "SELECT * FROM MY TABLE" | out-chart
```

Note: the invoke-sql cmdlet uses Windows Authentication to connect to SQL Server so there's no need to include authentication credentials  (user name and password) to use this command.

If you want to use an ODBC compliant connection string you can use the invoke-sql cmdlet as follows:

```
invoke-sql -connection "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=dbdemos.mdb" -sql "SELECT ItemsTotal FROM orders WHERE OrderNo<1100" | out-chart
```

Note: Populating gadgets with data coming from databases can also be done through the PowerGadgets Creator Data Wizard which provides a GUI for you to create a database connection inside a gadget file (.pgf). As depicted in the following figure.

For additional information, please refer to the PowerGadgets Creator later in this chapter.
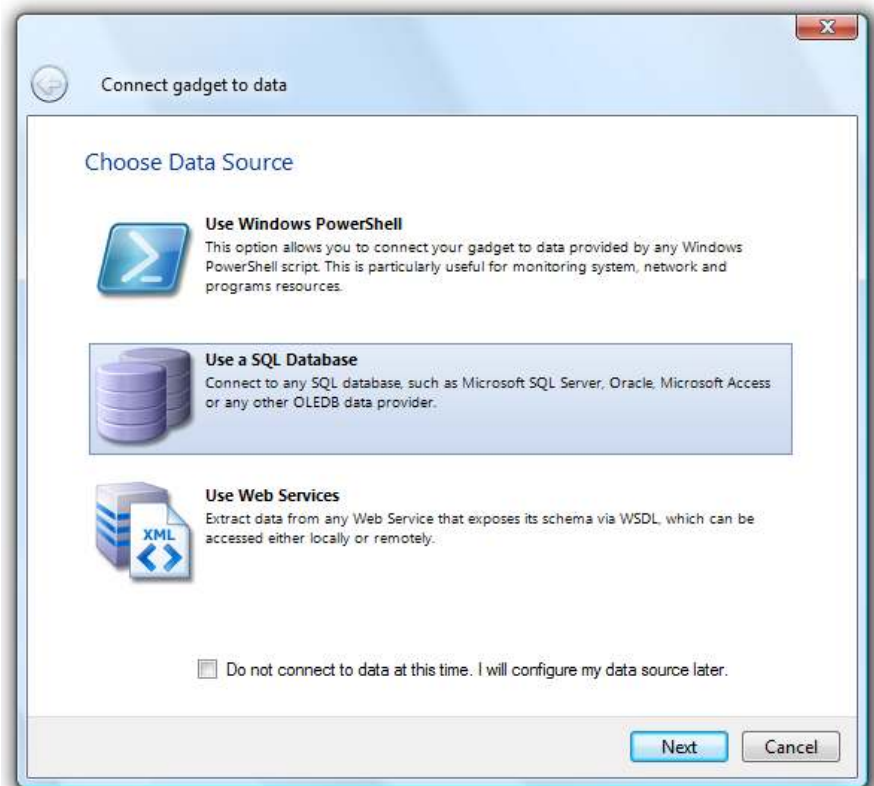
## Populating PowerGadgets from Web Services

The web service paradigm is rapidly being adopted throughout the industry from basic prototypes to full fledged online API exposure for first class web sites such as Ebay or Amazon.  PowerGadgets leverages this paradigm by including a *invoke-webservice* cmdlet for pulling data into PowerGadgets.

The following are parameters available for invoking the web service:

- wsdl – Specifies the web service description language either in the form of a web based URL or file based URI (file://d:/Temp/Test.wsdl)

- service – The name of the service.  This value is optional and is only necessary when a wsdl file has multiple services.

- Method – The name of the method to execute.

Prior to making use of a web service, you have to determine what type of information the web service returns and then appropriately associate this information to one of the gadgets.  A web service can supply a primitive value or an array of privative values such as an integer or double, however a web service can also supply complex objects multiple properties.  The sample below presumes we have a web service having a web method returning a collection of employees each having a Name and Age property similar to our CSV sample above.
Assuming the web service was published at http://www.tempuri.org/employee.wsdl containing method GetEmployee, you could use PowerGadgets invoke-webservice utilitarian cmdlet for invoking the web method as follows:

```
invoke-webservice -wsdl http://www.tempuri.org/employee.wsdl -method GetEmployee
```

As you can see, the web service returned the same set of employees with each available property listed. Plotting these values is as simple as piping the values to the out-chart cmdlet as shown:

```
invoke-webservice -wsdl http://www.tempuri.org/employee.wsdl -method GetEmployee | out-
chart -Values Age -Label Name -Title "Employees"
```

Note: Populating gadgets with data coming from web services can also be done through the PowerGadgets Creator Data Wizard which provides a GUI for you to connect to a wsdl file and create a gadget file (.pgf) that invokes any WebMethod  that returns meaningful information.

For additional information, please refer to the PowerGadgets Creator later in this chapter.

## Populating PowerGadgets from Text files (CSV)

Working with comma separated values (CSV ) is quite simple when using the PowerShell provided *import-csv* cmdlet. The *import-csv* cmdlet automatically produces an array of objects where the name for each column is exposed as an accessible property.

First create a file called employee.csv similar to the following text file in your documents folder:

```
Name,Age
Mike,32
Dennis,17
Manuel,24
Tomas,27
```

Now, use the *import-csv* command to make sure PowerShell can read the file as a CSV file by executing the following cmdlet.

```
Import-csv "c:\users\johndoe\documents\employee.csv"
```

This should produce an output similar to the following if the CSV file is properly formatted and interpreted by PowerShell.

Each column in the CSV file is treated as a Property, so plotting the CSV values with a chart is as simple as the following command:

```
import-csv "c:\users\johndoe\documents\employee.csv" | out-chart -Values Age -Label Name
-Title "Employees"
```

## Mail Delivery

Sometimes you simply want to deliver an email with the gadget's image representation. For this, PowerGadgets provides the send-mail cmdlet. The cmdlet can simply be added to the end of any chart, gauge or map cmdlet execution as shown below:

```
ps | out-chart | send-mail -From sender@softwarefx.com -To johndoe@softwarefx.com -
subject "Sample Subject" - Text "This is a sample process chart" -server mailserver
```

The following parameters are available for the send-mail cmdlet:

- from – the sender of the email

- to – the recipient or recipients of the email

- subject – the subject of the email

- text – the text to appear in the body of the email

- server – the mailserver to use for delivering the email

Email address can be either a single email as shown above, or a name/email combination such as "John Doe <johndoe@softwarefx.com>". By using the name/email combination, the name appears when the recipient reads the email – making the email more legible.

### Configuring Defaults

Many times the sender and the server will never change, hence you may want to configure default values for these parameters. To do this, simply edit the GlobalSettings.xml file located in yourAppData folder of your profile as shown where userx is the username of the currently logged in user.

In Windows Vista, the path would be as follows:

```
C:\Users\userx\AppData\Roaming\PowerGadgets
```

In Windows XP, the path would be as follows:

```
C:\Documents and Settings\userx\Application Data\PowerGadgets
```

The file's format should be as shown below:

```
<Settings>
  <send-mail>
    <Server>mailserver</Server>
    <From>John Doe [johndoe@softwarefx.com]</From>
  </Mail>
</Settings>
```

# PowerGadgets Drilldown in PowerShell scripts

PowerGadgets was originally intended and designed to allow the quick authoring and monitoring of information with minimal coding and without the complexities of a development tool like Visual Studio. However, in some cases, you will want to create the ability for end users to select a given gadget and drill down based on a current selection. This can be easily achieved with PowerGadgets and PowerShell scripts.

PowerGadgets supports two drilldown scenarios:

## Drilldown using the commands in the pipe
This option is useful mainly with hierarchical /recursive data, which uses the same chart type. It reuses the same chart but changes the underlying data itself. It also remembers the previous charts in the queue and adds a Back button to the context menu (right click).
Example:
The FolderSize.ps1 script described below is used to display the size of folders/subfolders and files in a directory. When you double click on an element, the folder name can be used as a parameter to drilldown to that specific folder as follows:

```
$items = get-childitem $args
foreach($item in $items) {
 if ($item -is [System.IO.DirectoryInfo]) {
    add-member -inputobject $item -membertype ScriptProperty -Name FolderSize -Value { $a = get-childitem
$this.FullName -include *.* -recurse | measure-object -sum Length; if ($a -eq $null) {return 0;} else {return $a.Sum} }
    write-output $item
 }
}
```

You can then invoke the Foldersize.ps1 as follows to achieve the desired drilldown capabilities:

```
foldersize | out-chart –values FolderSize –Label Name –Drilldown_Parameter {$_.Name}
```

The DrillDown_Parameter is used to specify the property that will be used to pass as parameter. Multiple properties can be passed, comma separated (e.g. –Drilldown_Parameter {$_.Name},{$_.Handles})

Note: when used in this way, out-chart will invoke only the pipe within the same command line. This means that if you have a foo.ps1 with more lines than just the line in the Usage above, those lines will not be invoked when drilldown is executed.

**Drilldown using multiple PowerShell scripts**

This option is very useful when you want to show master/detail data. This technique can also be used with hierarchical data (if you invoke the same script). Essentially, you have the freedom to invoke a new script with a parameter. Unlike option 1, this option does not use the same chart nor adds a Back button, so you need to establish your own strategy depending on what you want to do. To illustrate we will use Powergadget's invoke-sql cmdlet to extract related data from a database containing movie names and quantities available when you double-click a particular movie in a chart.

The one liner looks as follows:

```
invoke-sql -Server DBTEST -Database MyDatabase -SQL "SELECT Quantity,OriginalTitle,MovieID FROM Movies"  | Out-Chart -Values quantity -Label originaltitle -Drilldown_Parameter {$_.MovieID} -Drilldown_Script RentMovie.ps1
```

There could be many variations to the RentMovie.ps1 script according to the behavior you want to create when drilling down the chart.

Our first version of RentMovie.ps1 involves showing details on a separate chart and modeless window

**RentMovie.ps1**

```
$param=$args[0]
$query="SELECT TimesRented FROM RentMovies WHERE MovieID="+$param
invoke-sql -Server DBTEST -Database MyDatabase -SQL $query | out-gauge -type digital -Titles_0_Text "Times Rented" -Titles_0_Layout_Alignment TopCenter
```

Note: the query needs to be assigned in a variable because PowerShell complains about concatenating strings when used as parameters.

You can also create another version of Rentmovie.ps1 that shows the details on a separate modal window. The difference between this option and the previous one is that the digital panel appears on a modal window, which needs to be closed before you can click on another bar.

**RentMovie.ps1**

```
$param=$args[0]
$query="SELECT TimesRented FROM RentMovies WHERE MovieID="+$param
invoke-sql -Server DBTEST -Database MyDatabase -SQL $query | out-gauge -type digital -Titles_0_Text "Times Rented" -Titles_0_Layout_Alignment TopCenter -modal
```

Finally, you can show the details on the same chart window, as a group. This option is very useful since it shows the results encapsulated on a single window. In the following example, double clicking on a movie will show the details of all the movies released on the same year as the movie you have clicked.

```
invoke-sql -Server DBTEST -Database MyDatabase -SQL "SELECT Quantity,OriginalTitle,ReleaseDate FROM Movies"  |
Out-Chart -Values quantity -Label originaltitle -Drilldown_Parameter {$_.ReleaseDate} -Drilldown_script RentMovie2.ps1
–group Movies –Name General
```

**RentMovie2.ps1**
```
$param=$args[0]
$query="SELECT ReleaseDate, Quantity FROM Movies WHERE ReleaseDate="+$param
invoke-sql -Server DBTEST -Database MyDatabase -SQL $query | out-chart –group Movies –Name Details
```

A variation of the previous usage can be performed if you use the parameter as the gadget's name, which will create multiple entries in the group. ReleaseDate is a numeric field and will be used as the chart name, thus appearing in the group tab.

**RentMovie2.ps1**
```
$param=$args[0]
$query="SELECT ReleaseDate, Quantity FROM Movies WHERE ReleaseDate="+$param
invoke-sql -Server DBTEST -Database MyDatabase -SQL $query | out-chart –group Movies –Name $param
```

## File Output

Depending on your requirements, you may sometimes find the need to create gadgets on the file system as images. For this, PowerGadgets provides the -output parameter.  This is a very effective way to create dynamic gadget snapshots in the form of images for inclusion in web pages and other reporting tools.

This sample creates a chart representing the running processes as a png on a web server absolute path.

```
ps | out-chart –output c:\inetpub\wwwroot\myapp\processchart.png
```

The supported file extensions are: bmp, wmf, and png.  The file format is determined automatically based on the extension.

## The PowerGadgets Creator

When using PowerGadgets with Windows PowerShell you will author scripts (or .ps1 files) that invoke any of the PowerGadgets cmdlets and you can either script the gadget visual attributes or use PowerGadgets' graphical wizards to create templates (or .pgt files) or XML files that contain a gadget visual attributes. So far, you have learned how PowerGadgets seamlessly integrates to Windows PowerShell and how scripting can help you visualize and monitor data from virtually any source, including system information and corporate databases.

Although scripting from a PowerShell prompt is a really effective and flexible way of using PowerGadgets, there is a way to create gadgets that read information from a database, a web service, and even a PowerShell script without invoking a PowerShell prompt and in a full graphical environment called the PowerGadgets creator.
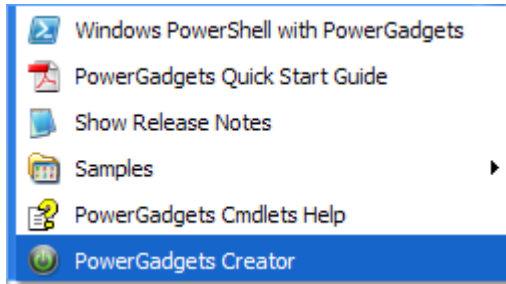


The PowerGadgets Creator is a self executable application you can use to author gadgets. When you install PowerGadgets, a shortcut to the PowerGadgets Creator is available. As depicted in the following picture:

The advantages of using the PowerGadgets Creator are:

1) Using a full graphical environment to author gadgets and reduce the learning curve as no scripting is necessary to create gadgets. In fact, this environment is so easy to use even non-technical users can create or customize the way gadgets look.

2) Creating self-contained files (or .pgf files) that are easy to redistribute. Because both the gadget visual attributes and data connection are serialized in the .pgf file, you can easily redistribute gadget files among your user base (you must have provisioned those who want to run or view gadgets with the PowerGadgets software) via a network drive or even email to your users. The PowerGadgets software even creates a file association on the target machine so gadget users can simply double-click .pgf files.

3) Simplifying the creation of gadget groups. The PowerGagets Creator lets you author multiple gadgets (charts, maps and/or gauges) into a single file that can be displayed in the Desktop or in the Windows Vista Sidebar and provides a VCR style control to navigate through the different gadgets in a group.  Grouping is a very effective way to manage desktop or Sidebar space.

4) Reducing the PowerGadgets software requirements for clients. In certain scenarios (database and web service connectivity), the only requirement to run and view gadgets is the .NET framework 2.0 (which, most times, is already part of the OS) and the PowerGadgets software.  This effectively reduces deployment complexities as PowerShell installation and manageability is not required on client machines.

5) Enabling a security framework to digitally sign gadgets. Just like PowerShell scripts, PowerGadgets files (.pgf) provide security mechanisms, through the use of certificates, that prevent malicious code to be executed in client machines. For additional information, please refer to "Digitally Signing PowerGadgets files for Distribution" later in this document.

Once you launch the PowerGadgets Creator, a wizard will appear prompting you to create your first gadget. Once you have selected your gadget type, you can then proceed to populate the gadget via a PowerShell script, a database connection or a web service as prompted as depicted in the following figure:
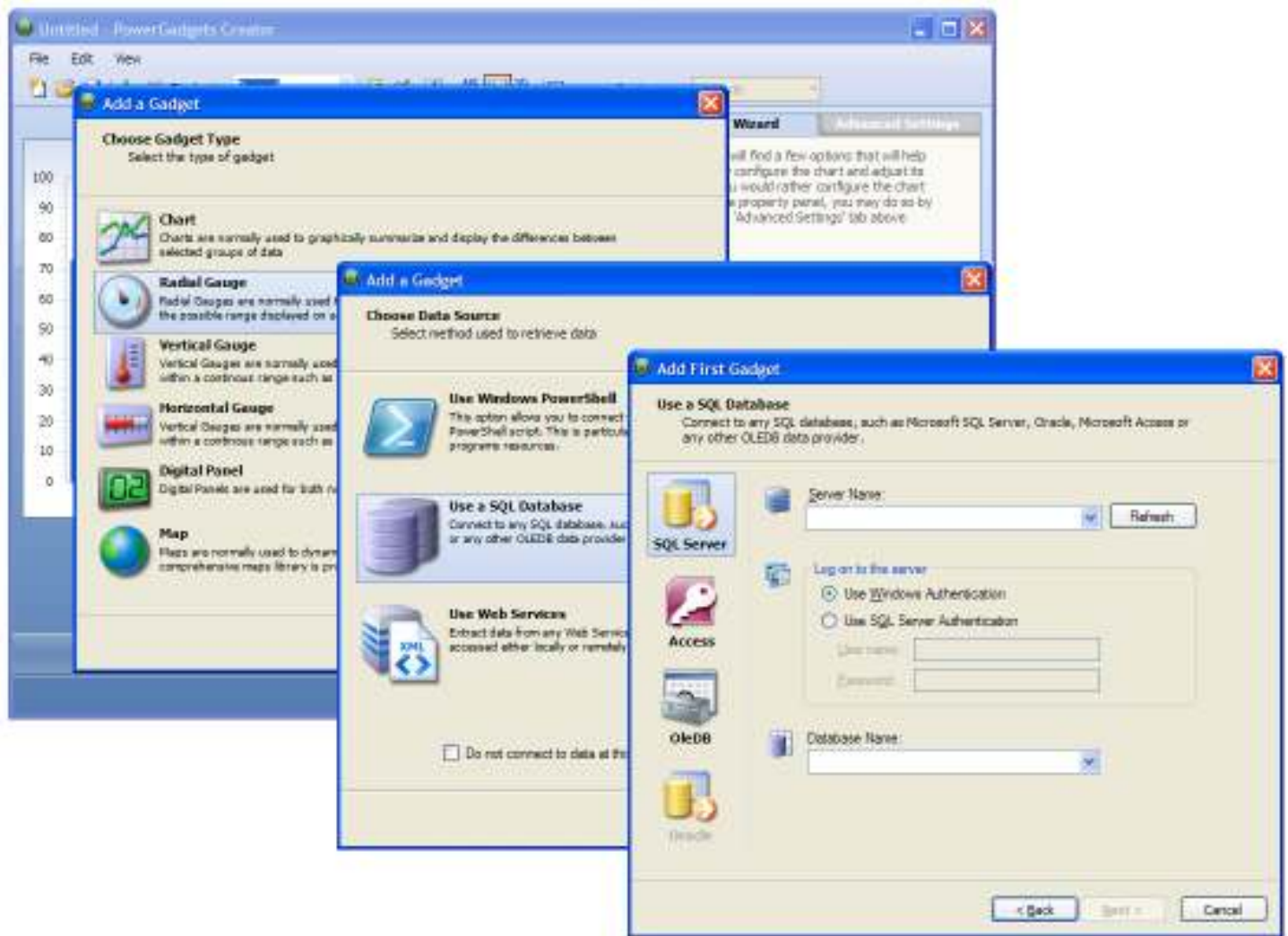


Please note you can always edit or modify the data source connection by simply overwriting the data command or by re-entering in the data wizard.

Finally, you can use the PowerGadgets Creator UI, to fully customize the way your gadget looks. These wizards and their properties are self explanatory. However, please note the following important UI elements in the PowerGadgets Creator:
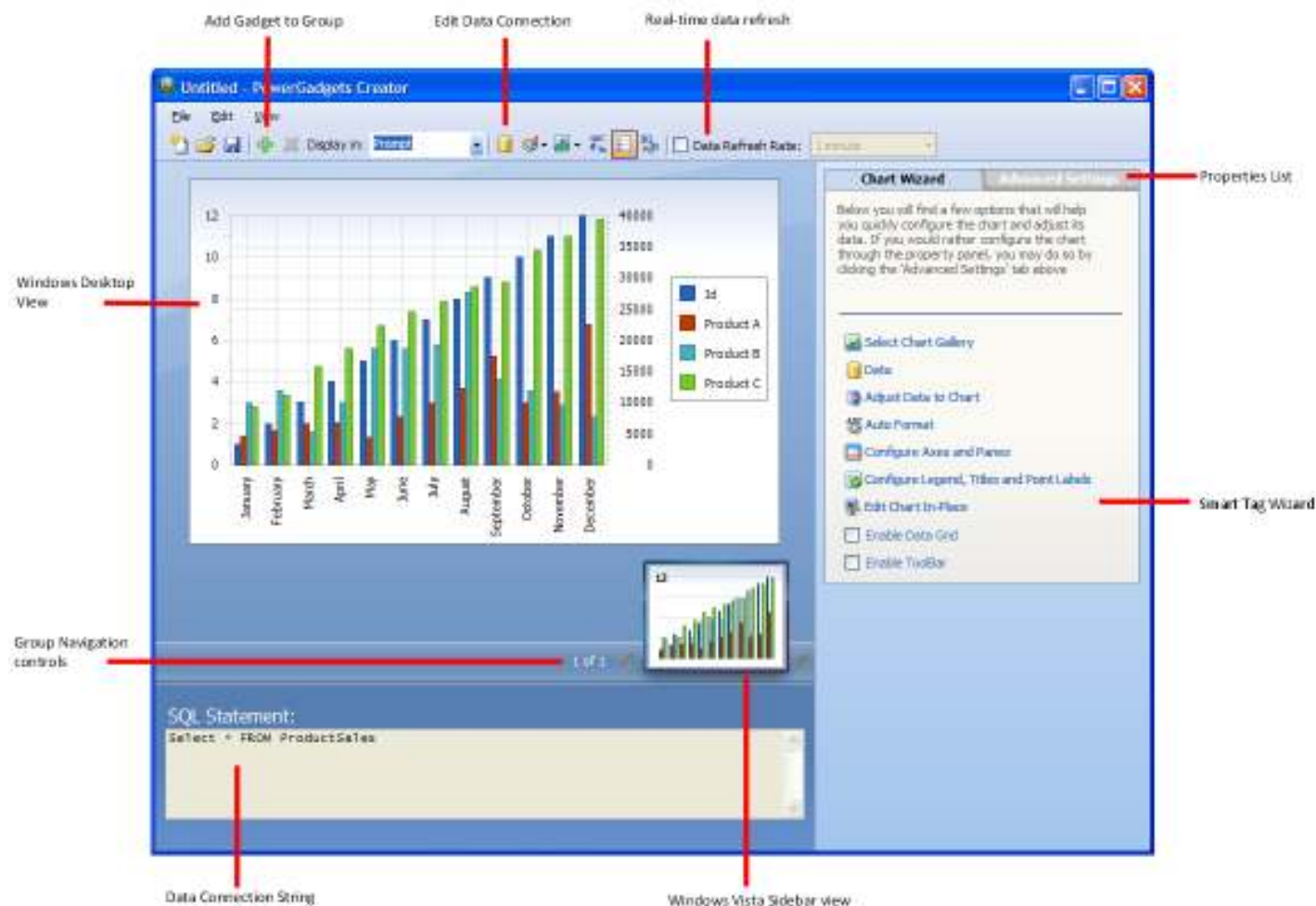


Once you have populated the gadget and its visual attributes, save it and redistribute to other users. Those who have been provisioned with the PowerGadgets software will be able to run and view gadgets. However, they will not be able to modify or alter its content or visual attributes.

# Digitally Signing PowerGadgets Files for Distribution

PowerGadgets files have been designed to be self contained and easily distributable to other machines via any common means such as email.  This section will describe some useful terminology as well as the native built in security features PowerGadgets provides out of the box for securely distributing PowerGadgets files.

## Why Digitally Signing PGF is Important and Recommended

PGF files are structured XML files containing all the necessary metadata for configuring the gadget, data extraction and manipulation for the gadget, and environment runtime settings.

While configuring a gadget's cosmetic settings does not pose any security risks, the data section typically contains code which may require elevated security.  For example, connection to a database or accessing security sensitive WMI classes.  Since this section is exposed as <Data>...</Data> XML nodes anyone can easily tamper with this section and inject additional potentially malicious code.  For example, someone can easily inject code to erase files every time the gadget is launched.

As a consequence, the only way to ensure the file has not been tampered with is by digitally signing the PGF file.  By digitally signing the certificate, client machines will be prompted if they want to accept the authority who produced the certificate prior to executing the PGF file.

## PowerGadgets Security Model

Since PowerGadgets can be easily setup to view previously authored gadgets, the PowerGadgets team has focused on making PowerGadgets as flexible and secured as possible by providing various security modes that are configurable based on your scripting needs.  These modes are *Restricted*, *AllSigned*, *RemoteSigned*, and *Unrestricted*.

Each of these modes provides varying levels of security as described below:

**Restricted** – Does not run PowerGadgets Files.  Interactive commands only.

**AllSigned** – Runs PowerGadgets Files.  All PGFS and configuration files must be signed by a trusted publisher.

**RemoteSigned** – Runs PowerGadgets Files.  All PGFS and configuration files downloaded from communication applications such as Microsoft Outlook, Internet Explorer, Outlook Express, and Windows Messenger must be signed by a trusted publisher.

**Unrestricted** – Runs PowerGadgets Files.  All scripts and configuration files regardless if digital signature is present or not.

By default, PowerGadgets is configured to run with the RemotelySigned security model.

You can change the default execution policy for all shells launched on a given computer, by updating the corresponding PowerShell registry entry and setting it's value to one of the for security levels specified above.

HKLM\SOFTWARE\Software FX, Inc\PowerGadgets\1.0\ExecutionPolicy
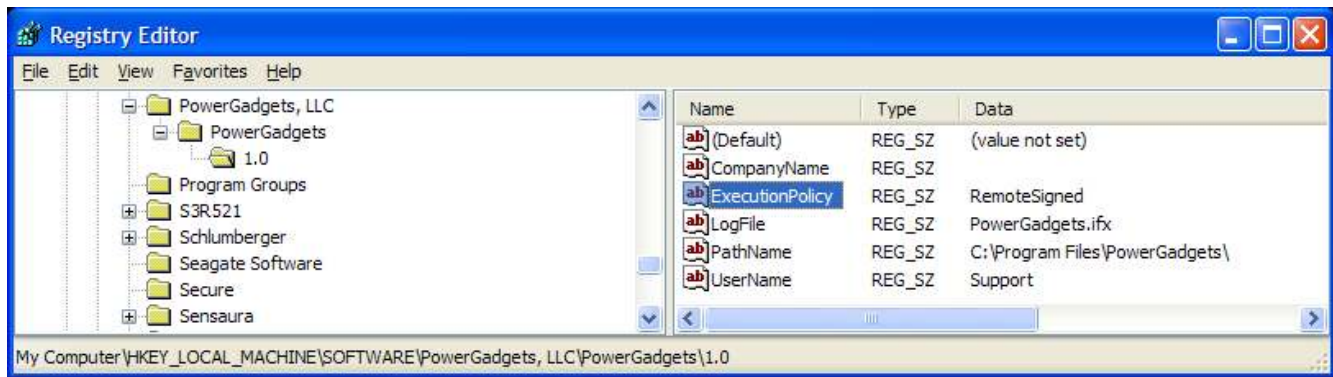


For convenience, PowerGadgets includes a cmdlet specifically for this purpose called Set-PGFExecutionPolicy where any of the four values may be used.

## Distributing and Running PowerGadgets Files

Since PowerGadgets files (.PGF) are fully contained, they can be distributed and executed on client machines similar to an executable provided the client machine has the PowerGadgets software installed.

The easiest way to have PowerGadgets files running on client machines without much intervention is simply to set the execution policy of the client machines to Unrestricted. While this may work for running scripts in all cases, there are risks involved with accidentally running a script unintentionally from an unknown source which may potentially include malicious code – causing concern for some users.

## Creating a Certificate for Digitally Signing Power Gadget Files

If you already have a certificate installed for code signing, you can skip to the next section where we explain how to digitally sign your PowerGadgets file.

## Creating a Certificate Authority

To create a certificate, you can either A) purchase and install one from a trusted authority such as Verisign or, if you are working within a controlled environment such as a local network, you can become an authority for the network and then create your own certificates. By creating your own certificate however, the certificate must be installed on every client computer as the certificate is originating from an non-verifiable computer.

Note:  This step is only necessary if you do not already have a certificate from a trusted authority such as Verisign.

Lets start off by first preparing the Microsoft management console for viewing certificates currently installed on the computer.

1. Launch Microsoft Management Console by clicking Start->Run and typing MMC followed by the Enter key.
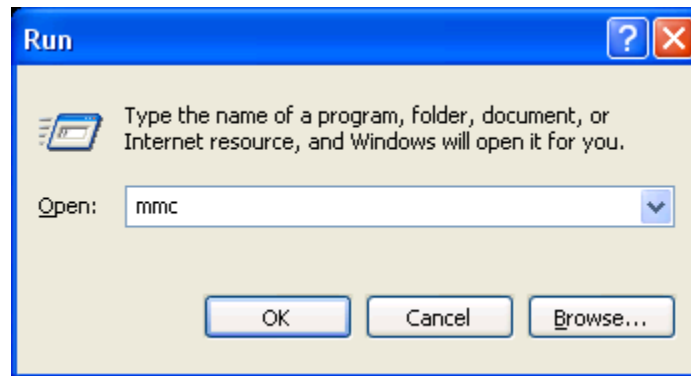


2. Now, choose the Add/Remove Snap-in... option from the file menu and click the Add button when the Add/Remove Snap-in dialog appears. You should see a series of snapins available for integration with the management console as shown below.
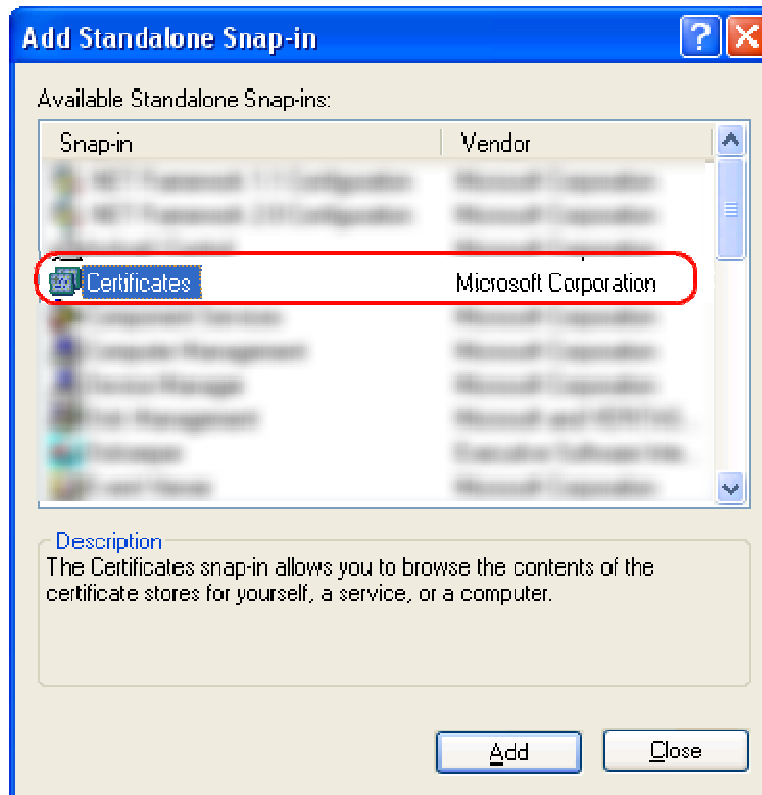
3. Select the Certificates from the available Snap-ins and click the Add button.  When the confirmation dialog appears, accept the default "My User Account" and click the Finish button.

4. You should now be able to see all the available certificates installed for the computer.

Now an authority needs to be created for managing the certificates originating from the computer.  To do this, as well as creating the certificate itself, you must use the makecert.exe program.  This is available as part of the Microsoft .NET Framework SDK or Microsoft Windows Platform SDK.  The latest is the .NET Framework 2.0 SDK  – after installing, makecert.exe is found in the "C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin\" directory.

To create a local authority for certificates, run the following command:

**makecert -n "CN=PowerGadgets Local Certificate Root" -a sha1 -eku 1.3.6.1.5.5.7.3.3 -r -sv root.pvk root.cer -ss Root -sr localMachine**

Note:  For additional information on the advanced parameters specified, you may run the makecert.exe program with a "-!" (dash followed by exclamation) as the first parameter.

Both the sha1 and md5 algorithms require a password.  As consequence, you will immediately be prompted for the private key text.  Enter a password to use as part of the encryption algorithm.  For best security, it is recommended you use a combination of letters and numbers with sufficient length.  You will then be re-prompted for the same key for the self signing process ( the -r option specified above ) where the key is used to sign itself.

You should now see the Certificate Authority listed within the Certificates MMC window under the Trusted Root Certification Authorities as shown below.
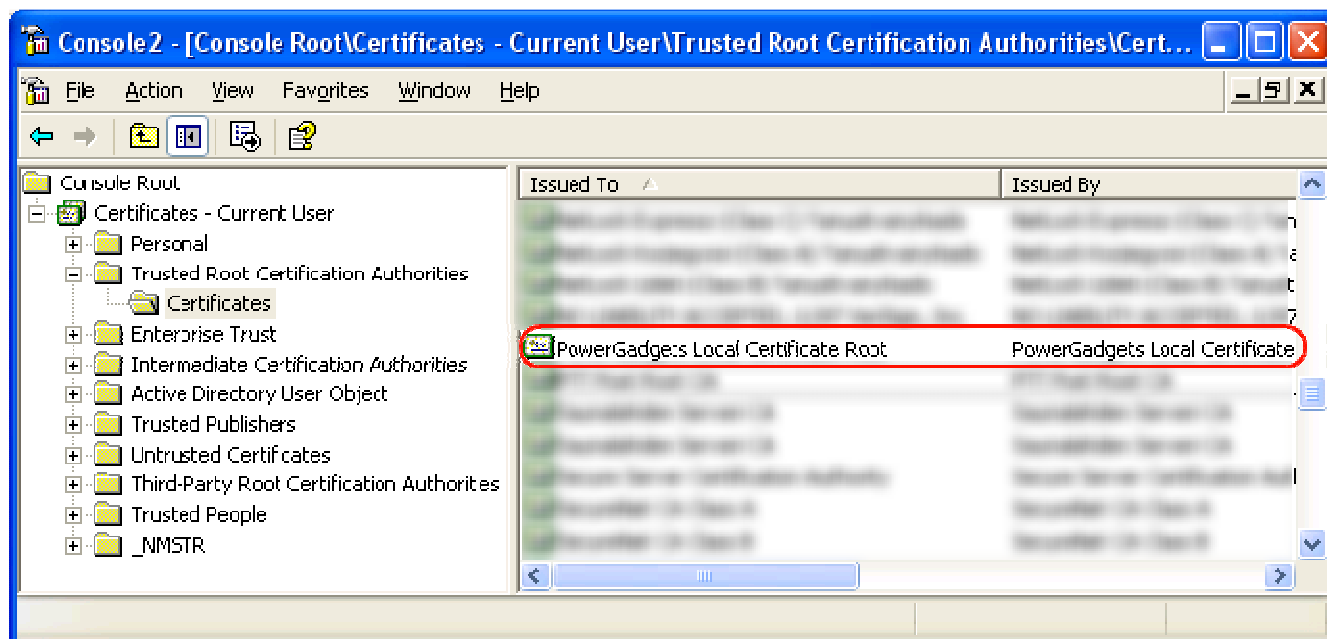


## Creating a Certificate

With a certificate authority in place, you can create as many certificates as necessary. For our purposes, we will create a single certificate and use this certificate to sign PowerGadgets files for execution under the *AllSigned* or *RemoteSigned* security models.

To create a certificate using the authority we just created, simply use the following command:

**makecert -pe -n "CN=PowerGadgets User" -ss MY -a sha1 -eku 1.3.6.1.5.5.7.3.3 -iv root.pvk -ic root.cer**

You will be prompted for the private password used earlier, and once supplied the certificate should become available in the Personal section with thing the Certificates MMC Snap-in as shown below.
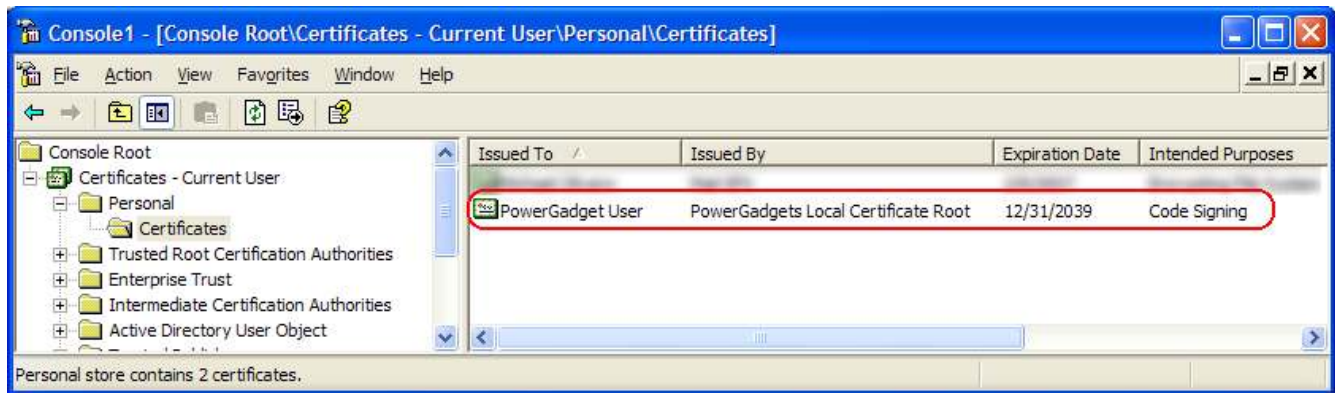
Notice the certificate has been designated for "Code Signing" purposes as shown in the last column.

## Using Certificate To Sign PowerGadgets File

PowerGadgets files can now be digitally signed using the certificate we created in the previous section.  Since PowerGadgets was designed from the ground up with security in mind, the designer tightly integrates script signing mechanism into the designer.

Launch the PowerGadgets designer from the start menu and use the various wizards and options to configure your power gadget as desired.
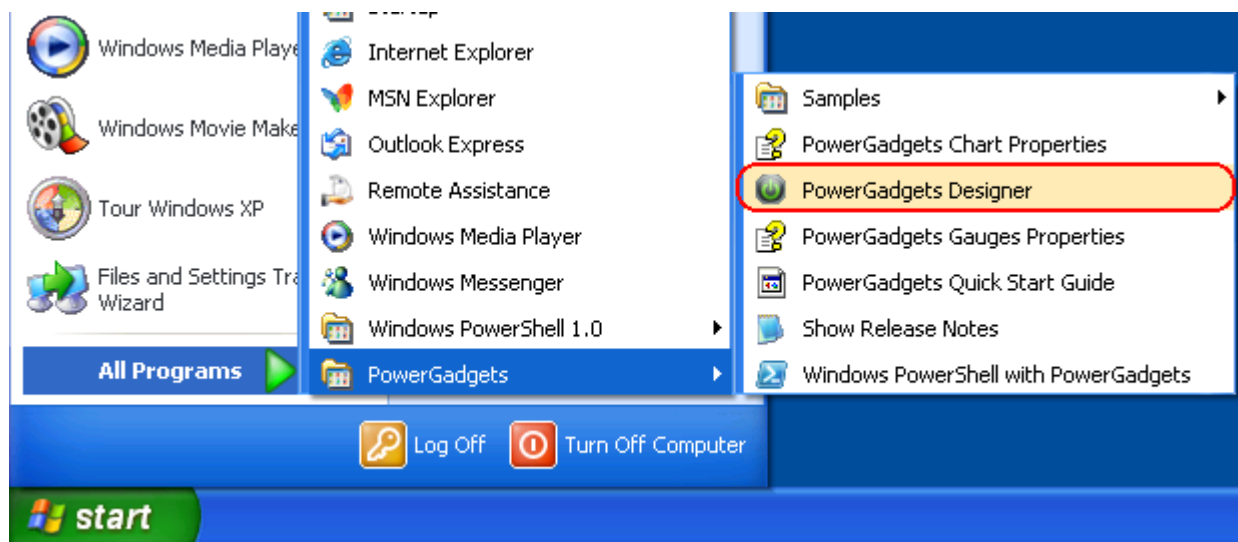


When ready to save, simply choose Digital Signature... from the File Menu
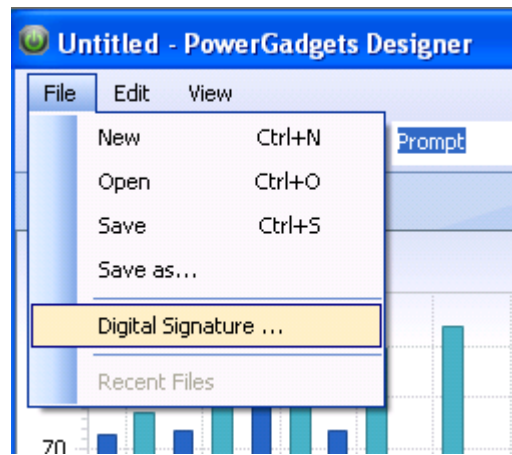
And choose the PowerGadgets certificate created earlier prior to saving the PowerGadgets File.
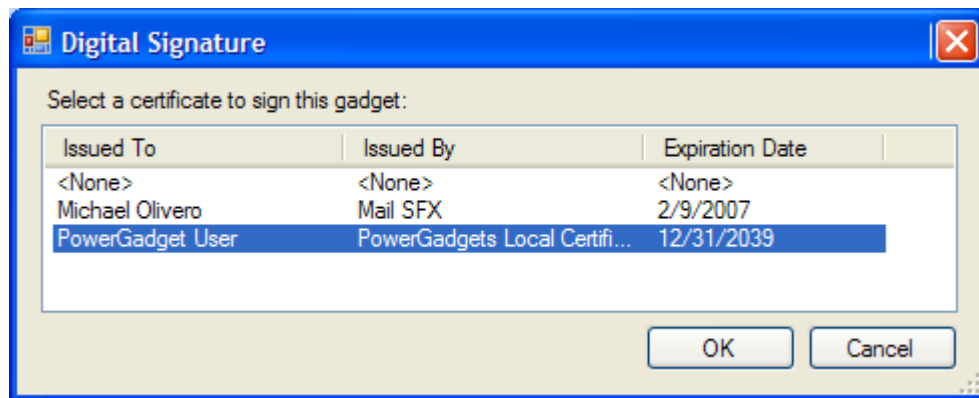


The final PowerGadgets file will automatically be signed with the certificate for trusted execution when saved.

**Executing a PowerGadgets With Signed Security**

Since the local computer has the digital certificate installed, you can already execute signed PowerGadgets files.  The PGF file is executed differently depending on the execution policy set by the client machine.

RemoteSigned is more flexible than AllSigned.  Remote signed requires digital signatures to be check ONLY if the file has been obtained from a remote source such as web site, messenger, or other means supporting the remote attribute mark.

NOTE:  The remote attribute mark is unique to Windows and NTFS where additional file attribute information is stored on additional streams related to the file identifying the file as being obtained from remote sources.

As a benefit, scripts are generally not considered as "remote" if they are obtained from local network or internal means such as Outlook, messenger, etc. so no certificate check will be necessary to execute these PGF files.

If strict security is required, then we recommend setting PowerGadgets execution model to AllSigned.  This setting requires all PGF files to validate their digital signature prior to execution.  When executing a digitally signed PGF file with this security model, the client will be prompted for authorization prior to accepting.  If the client does not have the certificate installed on his machine, then the script will not be executed as shown below:
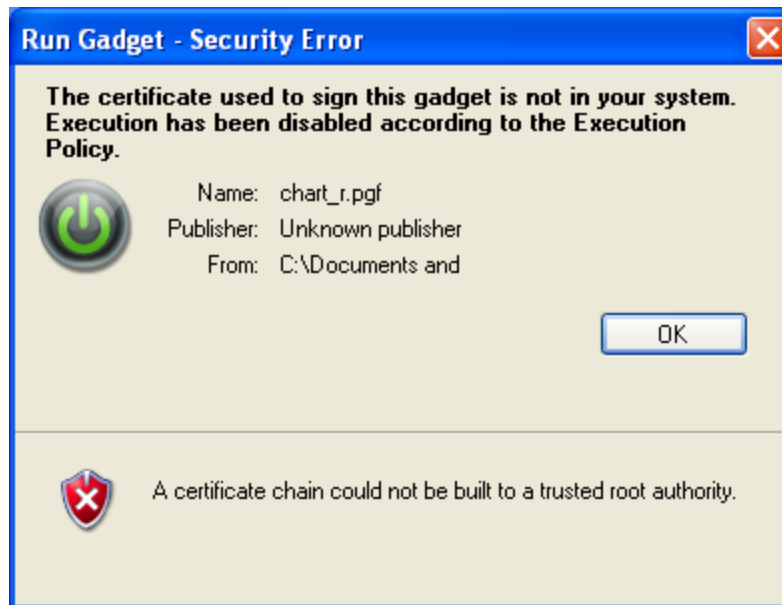


In this mode, all client machines expecting to run PGF files from a specific trusted source, should have the certificate for the trusted source installed on their machines.

## Installing Certificates on Client Machines

To install a certificate on a client machine, simply launch the MMC console as described previously and right click the certificate under the following branch:

Trusted Root Certification Authorities\YourCertificateName


and chose the Export... as shown below.
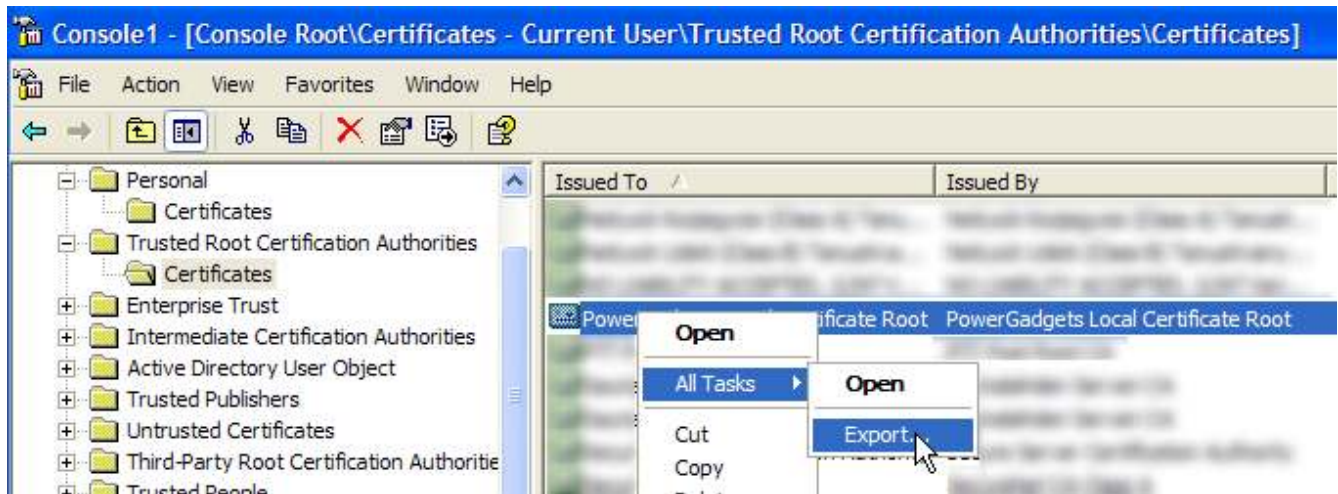
Click next accepting all the defaults, including the default format of "DER encoded binary X.509 (.CER)" option and specify the file name such as c:\TrustedRootCert.cer

If all goes well, you should receive a confirmation such as the following indicating the certificate was successful created.



On the client machine, launch the MMC console navigating to the same tree branch and right click the Certificates node selecting "All Tasks -> Import" as shown below.
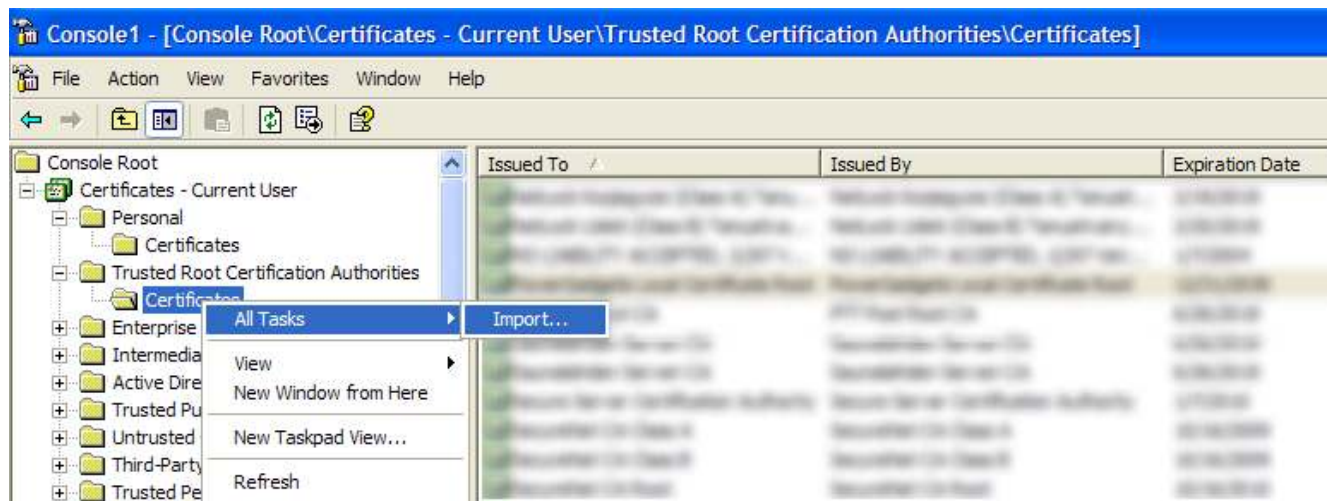


Specify the .cer file created in the previous section and click next accepting all defaults. After a few screens you will arrive at a security warning. This warning is normal because the certificate authority is local – click Yes to accept the certificate. Now the client machine should be able to run PGF files signed by the authority.

## Appendix A. General Guidelines for using the PowerGadgets Properties in  PowerShell

PowerGadgets provides an extensive and powerful API with thousands of .NET compliant properties to fully customize charts and gauges and their aesthetics directly from the PowerShell prompt. These properties help you customize virtually any chart setting from pipelined data from sources like databases, web services, processes, Windows Management Instrumentation, event logs, and much more…

Note: Please note most of these properties have been made available through graphical wizards available when the –configure switch is added to the PowerShell script. Therefore, we recommend you configure charts and gauges using this methodology instead of unnecessarily complicating the PowerShell script with these properties.

In addition, a copy of the PowerGadgets API Reference has been installed in the PowerGadgets installation directory. This CHM file contains the API that you can use within the shell. Please note this is a pre-release version of the API documentation and properties listed in this help file have not been thoroughly tested in PowerShell.

When reading the documentation please keep in mind the following modifications for PowerShell, this information will be properly updated in the released product documentation.

*Enumerations* use the property and the enumeration text
```
ps | out-chart  –gallery Lines
```

*Complex Properties*: simply use the underscore to dive in the object model
```
ps | out-chart  –AxisY_Max
```

*Boolean properties*: simply set to true or false
```
ps | out-chart  –toolbar true
```

*Numeric Properties* simply set the appropriate value
```
ps | out-chart –AxisY_Max 200
```

*Array Properties*: use underscore for brackets and the sero-based index of the appropriate element
```
ps | out-chart –Series_1_Color Red
```

*Text Properties*: enclose in "" when there are blank spaces, otherwise set text
```
ps | out-chart –titles_0_Text "My Processes"
```

# Appendix B. Obtaining additional help and documentation from Windows PowerShell

PowerGadgets includes numerous materials and resources for making sure you make most of your PowerGadgets investment.  The installation includes both an electronic HTML help file containing this resource as well as a resource for the full api spectrum available for the various cmdlets.

## Electronic Help

The electronic help files are located in the start menu.  There is a PDF file containing the contents of this resource, in addition to traditional CHM help files typically found with windows applications.

## PowerShell Help

Windows PowerShell includes a useful get-help cmdlet where you can use to obtain useful brief information on any cmdlet.  We have summarized the typical features and functionalities commonly used with each of the core cmdlets in their respective help files.  You can view these help files with the following:

```
get-help out-chart
get-help out-gauge
get-help out-map
```

## PowerGadgets Tab Completion

When authoring Gadgets via a PowerShell prompt you can enable Auto Tab completion by copying the WindowsPowerShell folder (containing the profile.ps1) to your \My Documents folder. This feature will allow you better explore the API for the out-chart, out-gauge, out-map or any other cmdlet provided by PowerGadgtes.
To test, type a cmdlet (e.g. out-chart) the - switch and start pressing the Tab key to see the available properties for that cmdlet. You can even see enumeration types after selecting the desired property. For example, you could type the following in a PowerShell script: out-chart -gallery and start pressing the tab key to see the available chart types provided by PowerGadgets.

Note: The PowerGadgets installer does not copy this file into the \My Documents folder to avoid accidentally overriding other default configurations that you may have performed to this file prior to the PowerGadgets installation. If you have already edited this file manually you may need to merge the contents of the PowerGadgets profile.ps1 file in your configuration file, you may do this manually or with any file merging utility.

## Appendix C. Windows PowerShell & PowerGadgets General Scripting FAQ.

### How do I generate and load a gadget template?

PowerGadgets exposes the same design-time experience found in graphical design surfaces like Visual Studio 2005. Simply use the -configure switch and PowerGadgets will activate a powerful graphical Wizard allowing advanced customization for chart and gauge settings. When used, a configuration template file is saved in the local user's My Documents folder and can then be use with the –template switch to load and apply these settings quickly and easily.

To setup a template simply specify the -configure parameter optionally followed by the name of the template as follows:

```
get-process | out-chart -configure -template MyProcesses
```

Note: By specifying the input command to be used with the template, the template editor can assist with associating the various available properties to Chart elements via the "Adjust Data to Chart" option.

When launching the PowerGadgets template editor, you will notice a dialog appear for fully customizing gadgets both cosmetically as well as associating the input to various chart elements.

Once all the settings have been applied, you simply have choose "Save" or "Save As..." from the file menu to save your settings. Later you can use the –template property to load and apply the template.

```
get-process | out-chart -template MyProcesses
```

Note:  To ensure consistency, a saved template should only be used with the same input command.  So if the template was configured for the Get-Process command, the Get-Process command should be used as input when using the template.

### How do I size a gadget on the desktop?

By default, Gadgets are automatically sized.  This is useful if you wish to create various cmdlets all having a specific size without having to worry about passing additional size-related parameters as described below. In some cases you may need to set gadgets to a desired size.
To size a gadget, you need to use the -Size parameter.  Size specifies the actual dimension of the final rendered chart. The values are passed with the -size width,height parameter as shown below:

```
out-chart -size 150,100
```

Note:  Passing a value of 0 for width or height indicates PowerGadgets should calculate the best possible width for the supplied height given the chart or gauge type displayed.  In some instances, automatic proportions also apply as in Horizontal and Vertical gauges.

## How do I make the gadget float?

You may use the -floating parameter to launch a gadget without a window frame – making the chart or gauge appear as it were floating over the desktop.

This, in combination with PowerShell script files (.ps1), provides enormous power and options for creating full feature dashboards by simply launching multiple floating and modeless charts and gauges for monitoring your data.

To close a floating chart or gauge you should right click and choose the close option from the context menu.

To demonstrate this, the following PowerShell command creates a floating non-modal gauge reporting the megabytes available on a Windows desktop.

```
get-wmiobject Win32_PerfRawData_PerfOS_Memory | out-gauge -value AvailableMBytes
-MainScale_Max 1000 –floating
```

## How do I position a gadget on the desktop?

With the ability to show charts, maps and gauges in a floating format, you will most likely want to position them as well as size them to fit according to your dashboard needs. For this PowerGadget cmdlets support the -location, -anchor, and -size parameters.

When used in unison, they provide complete control over the positioning of the control of the gadget on your desktop.

**Anchor**
Anchoring allows you to specify the general location of the chart or gauge with respect to your desktop. There are four locations where it may be anchored: topleft (default), topright, bottomleft, bottomright as shown below.

**Location**
Location specifies the horizontal and vertical offset from the screen edge relative to the anchor position. For example, if you anchor topleft and wish to offset the chart from the edges 10 pixes from the top and 10 pixes from the left, you can use the -location parameter with a value of 10,10 as shown below:

```
get-process | out-chart -anchor topleft -location 10,10
```

## How do I make the gadget disappear from the available Windows list during Alt-Tab ?

When there are many gadgets visible at the same time on the desktop, such as dashboard-like implementations it may be impractical to show all gadgets that have been created in the desktop. You can instruct PowerGadgets to remove a gadget using the HideFromAlTab property as follows:

```
get-process | out-chart -HideFromAltTab
```

Please note this property is also available in the PowerGadgets Creator ( Edit -> Desktop Settings menu entry).

**How do I make the gadget appear on top of other Windows applications?**

I some cases you will want to have a gadget always displayed on top of all other windows or applications you are using. This enables you to have realtime information visible regardless of which application you are utilizing – even if the application is in full-screen mode. To ensure a gadget is always displayed on top of all other windows, use the TopMost parameter as shown.

```
Out-gauge -TopMost
```

Note: Gadgets which are always visible usually automatically refresh themselves to show the most to date view. For this we recommend using the Refresh parameter as described below.

**How do I construct my own .NET object in PowerShell to populate a gadget?**

Since Windows PowerShell works with objects and list of objects piping from one cmdlet to another, there are clearly times where you would like to have a simple object with specific members exposed representing the values you wish to work with. This facilitates working with out-chart and out-gauge cmdlets as they have native attributes designed to work with the properties of piped objects.

First, to define an object and the members it will contain, it is recommended you work within a PowerShell script file (.ps1). You can define additional properties on existing .NET types, however this example will the base System.Object type for simplicity.

The following script declares an object using the new-object cmdlet and creates a set of properties using the add-member cmdlet. Finally the write-ouput cmdlet is used to let the object through the pipe:

```
$newobject = new-object object
Add-member -inputobject $newobject -membertype NoteProperty -Name EmployeeName -Value $name
Add-member -inputobject $newobject -membertype NoteProperty -Name Salary -Value $salary
Write-output $newobject
```

If the script shown above is saved into a ps1, you can easily use any of the PowerGadgets cmdlets to visualize the data, as follows:

```
Employee | out-chart -Labels EmployeeName -values Salary
```

The following FolderSize.ps1 scripts calculates the size of folders in the current directory recursively adding a Length property that can be used later to create a useful chart:

```
$items = get-childitem
foreach($item in $items) {
     if ($item -is [System.IO.DirectoryInfo]) {
          add-member -inputobject $item -membertype ScriptProperty -Name
FolderSize -Value { $a = get-childitem $this.FullName -include *.* -recurse |
measure-object -sum Length; if ($a -eq $null) {return 0;} else {return $a.Sum} }
          write-output $item
     }
}
```

Once you have saved this ps1 script. You can create a chart by simply typing the following script:

**How do I enable my PowerShell for PowerGadgets AutoTab completion?**

When authoring Gadgets via a PowerShell prompt you can enable Auto Tab completion by copying the WindowsPowerShell folder (containing the profile.ps1) to your \My Documents folder. This feature will allow you better explore the API for the out-chart, out-gauge, out-map or any other cmdlet provided by PowerGadgtes.

To test, type a cmdlet (e.g. out-chart) the - switch and start pressing the Tab key to see the available properties for that cmdlet. You can even see enumeration types after selecting the desired property. For example, you could type the following in a PowerShell script: out-chart -gallery and start pressing the tab key to see the available chart types provided by PowerGadgets.

Note: The PowerGadgets installer does not copy this file into the \My Documents folder to avoid accidentally overriding other default configurations that you may have performed to this file prior to the PowerGadgets installation. If you have already edited this file manually you may need to merge the contents of the PowerGadgets profile.ps1 file in your configuration file, you may do this manually or with any file merging utility.

**How do I specify the value to be plotted on a gadget?**

Depending on the gadget, there are various ways for specifying values to be plotted.

Since out-chart works with a collection of objects as input, out-chart includes a -Values parameter where one can specify the property to use for the Yaxis value for each point plotted. For example, here we plot the number of handles as the value and the name of the process as the XAxis labels:

```
get-process | out-chart -Values Handles -Label ProcessName
```

Out-gauge also has a -Value parameter, however since out-gauge typically only requires a single value to plot, out-gauge also accepts a value from the pipe.  Since out-gauge accepts a single value, out-gauge also makes some assumptions on which gauge to display depending on the type of the value.  Numerical types will load a radial gauge.  String types will display a digital panel which is designed to handle and display text based values.

Here are some samples for gauges:

```
get-date | out-gauge
```

```
"Some String Value" | out-gauge
```

These to out-gauge invocations are equivalent:

```
get-process | measure-object -sum Handles | select sum | out-gauge
```

```
get-process | measure-object -sum Handles | out-gauge -Values sum
```

**How can I write a complex script (.ps1) and pipe results into a PowerGadget within the script?**

In some instances, you need to write multiple lines of script code to achieve a particular objective. Many of the included samples are such.

To accomplish this, we recommend referencing the block of code with a PowerShell variable and then piping the results of the block to one of the gadgets. To create a block of code, simply wrap the block of code with the { and } characters and assign this block to a variable. For example:

```
$block =
{
    get-date
    'rest of the script goes here…
}
```

Now, you can use the results of this block and pipe it to the out-gauge gadget.

```
$block =
{
    get-date
    'rest of the script goes here
}

&$block | out-gauge
```

Notice when referencing the block, you must prefix the block variable with an ampersand character (&).

**How do I perform Sorting, filtering and grouping?**

Windows PowerShell provides powerful ways to sort, filter and group data by using cmdlets in your scripts. Some of which are significant when monitoring data with PowerGagdets. For example, when creating a chart from the get-process cmdlet you may want to limit or select only a few indicators rather than the whole set of data return by the get-process cmdlet. To do this, you can simply use the select cmdlet and then "pipe" the out-chart cmdlet to produce the chart, as follows:

```
ps | select WorkingSet,Handles| out-chart
```

Here are other similar basic scripts:


Sort:
Sorts the process by number of handles (least first)
```
get-process | sort Handles | out-chart
```

Filter:
The following filters the process and includes only those with handle count of less than 100
```
get-process | where { $_.Handles -lt 100 } | out-chart
```

Group:
```
dir *.* | group Extension | out-chart -Values Count -Label Name
```

Please note the use of the select cmdlet is imperative when creating gadgets that receive a single value (i.e. Gauges). For example, the following script extracts the Processor usage Percentage from a WMI class and pass it to the out-gauge cmdlet.
In addition, you can use the where and measure cmdlets to filter and sort data before passing it to any of the PowerGadget cmdlets. For additional information, please refer to the PowerShell documentation for select, where and meaure cmdlets.

## Appendix D. PowerGadgets out-chart cmdlet Scripting FAQ.

### How do I set a chart title?

Identifying each gadget with what they're actually measuring is not only a good practice but may be critical for end user readability when displaying multiple gadgets in your desktop. You can quickly display a title in your chart, as follows:

```
out-chart -Title "My Title"
```

Please note charts support more than just a single title. In fact, you can title axes and other chart elements using the out-chart parameters. For example, if you want to describe the Y Axis with the "In millions" string, you can do it as follows:

```
out-chart -AxisX_Title_Text "In Millions"
```

Note: If you are using Windows Vista, most titles and labels will be automatically removed when the chart is docked in Windows Sidebar. They will be redisplayed when the chart is detached from the Windows Vista Sidebar.

### How do I change the chart type?

The out-chart cmdlet lets you choose from a wealth of 2D/3D chart types to display your data. You can explore chart types and their capabilities using the PowerGadgets Graphical Wizard. However, if you want to change the chart gallery type from your PowerShell script you can use the –Gallery property as follows:

```
out-chart -gallery Lines
out-chart -gallery Area
```

Please note some chart types may require data in a specific format or with a specific number of series in order to properly display some gallery types, we strongly encourage you to explore the graphical configuration options before setting a chart type via scripting.

### How do I enable the chart's interactive Toolbar?

As fully interactive objects, charts let end users perform data analysis with UI tools that allow zooming, scrolling, highlighting and other interactive features that will empower users to analyze data. The chart tool bar will even allow end users to export chart to other productivity applications such as Word and Outlook for further redistribution of gadgets.

To enable the tool bar for end users, you can add the –toolbar property to your script as follows:

```
out-chart -toolbar_Visible true
```

## How do I change colors in a chart?

When configuring chart aesthetics, you can quickly setup colors by changing the chart palette. Palettes will change ALL elements in a chart based on a pre-defined and consistent set of colors. PowerGadgets provides over a dozen color palettes for all gadgets, including charts, gauges and maps.

The following script changes a chart palette:

```
get-process | out-chart -palette "Schemes.Classic"
```

In addition, you can change colors for individual objects in a chart (e.g. Series, labels, etc) by accessing each object's Color property. For example, the following script changes the color of the first series in a chart:

```
get-process | out-chart -Series_0_Color Blue
```

Note: If you change colors for individual objects in a chart, these objects will not respond to palette changes.

## How do I choose which values of the data set to plot in a chart?

PowerGadgets is capable of creating charts from a collection of objects.  Some cmdlets format these objects in convenient to read columns for the most common properties referenced.  For example, the get-process cmdlet returns a series of columns for the various useful properties of the ProcessInfo type resembling a tabular data format.  Similarly, the PowerGadgets invoke-sql cmdlet returns the content of a table in a database in a collection of .NET where a property is created for each column requested in the invoke-sql.

With this data, you can now use the –values and –labels properties to instruct the out-chart cmdlet which columns to plot as different series or x-axis labels. For example, the following script uses the –values and –labels properties to describe the content to be displayed on the chart:

```
invoke-sql -server DBServer -sql ="Select Product,QtySold,QtyInInventory FROM
Products" | out-chart Label Product -Values QtySold,QtyInInventory
```

## Conditionals

Another interesting aspect of the out-chart cmdlet is the ability to apply conditions to highlight data in the chart. The syntax is very similar to other PowerShell cmdlets. For example if you want to highlight processes that are using more than 600 handles, you can easily do it by typing the following command:

```
ps | out-chart –values Handles –condition {$_.Handles –gt 600} –condition_color Red
```

This script will render the following chart:
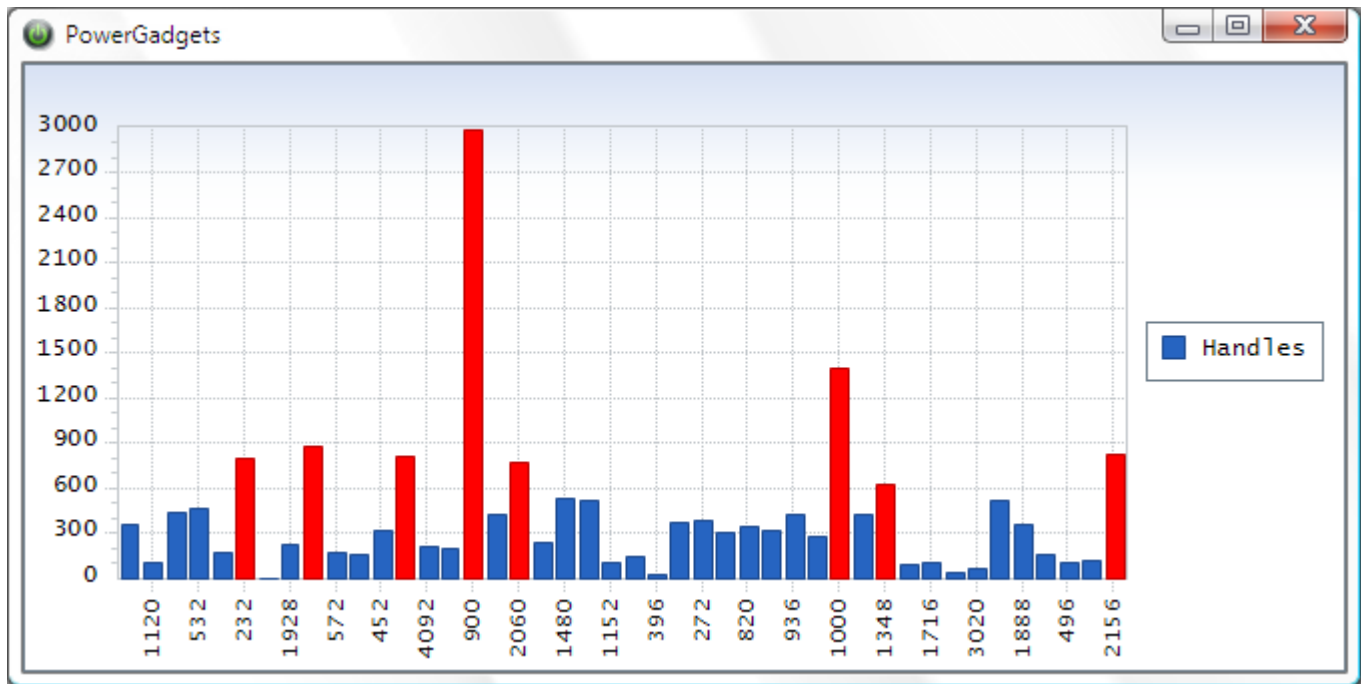


## What are chart smarts and how do I disable them?

For your convenience, PowerGadgets supports a DefaultViews.xml file that lets you control how PowerShell data will be used by the out-chart cmdlet. In essence, this file lets you simplify your scripts because no additional parameters will be required for the out-chart cmdlet when certain types are used. For example, for a collection of File objects, you may want the Length of the files to be the values in the chart while the Name of the file to be the Label.

The following is a snippet of the DefaultViews.xml file showing how FileInfo objects should be viewed in a chart.

```xml
<DefaultViews>
  <Views>
    <View Type="System.IO.FileInfo">
      <Fields>
        <Field Name="Length" Style="Value"/>
        <Field Name="Name" Style="Label"/>
      </Fields>
    </View>
```

```
    ...
</DefaultViews>
```

By specifying a default FileInfo view as above, you will only be required to use out-chart cmdlet as follows:

```
dir *.* | out-chart
```

producing the following results:

Similarly, the System.Diagnostics.Process type is configured (DefaultViews.xml) to display two sets of values – WorkingSet and Handles as shown below:

```
...
<View Type="System.Diagnostics.Process">
  <Fields>
    <Field Name="WorkingSet" Style="Value"/>
    <Field Name="Handles" Style="Value"/>
    <Field Name="ID" Style="Label"/>
    <Field Name="ProcessName" Style="ToolTip"/>
  </Fields>
</View>
...
```
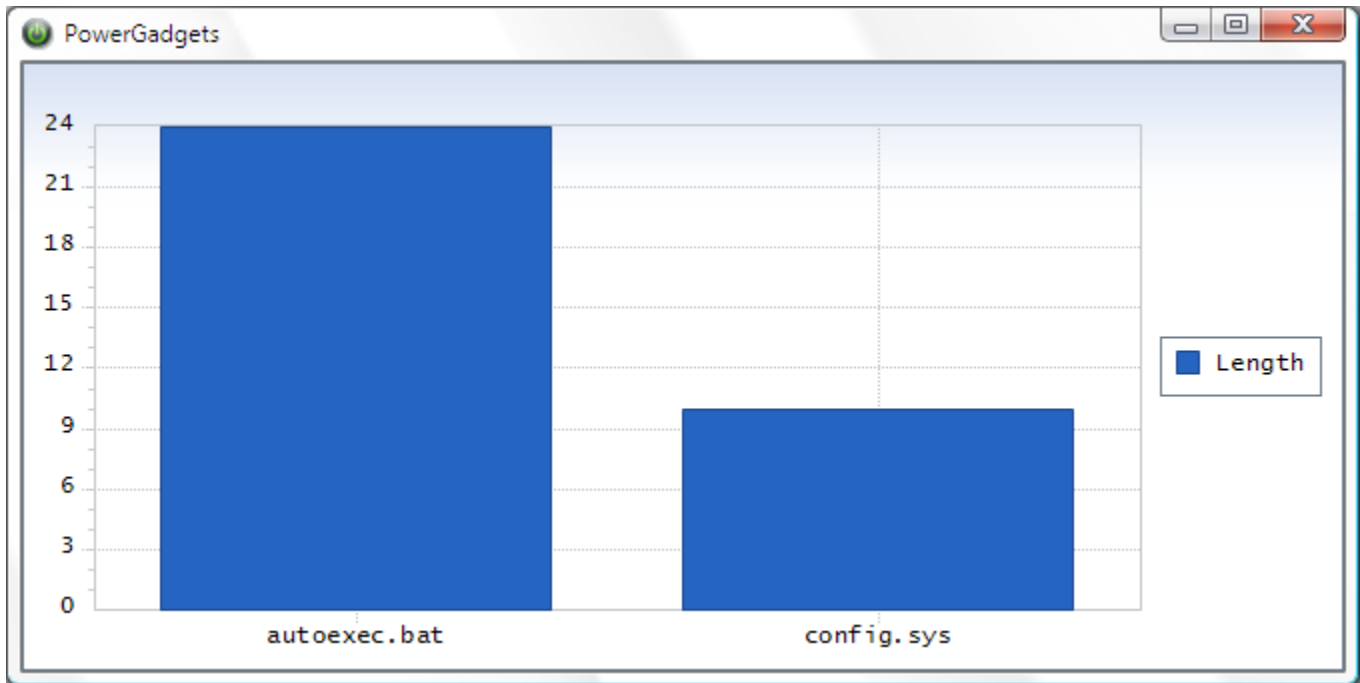
This automatically translates into two series of data within the chart. As an additional feature, Chat FX recognizes the WorkingSet values are significantly larger than the Handles and as a consequence PowerGadgets automatically turns on the multiple y-axis feature to split the values into two discernible ranges – improving the readability of both sets of values.

For completeness, the chart also includes a tool tip showing the process name as well as labeling the XAxis with the process ID number as shown in diagram below.
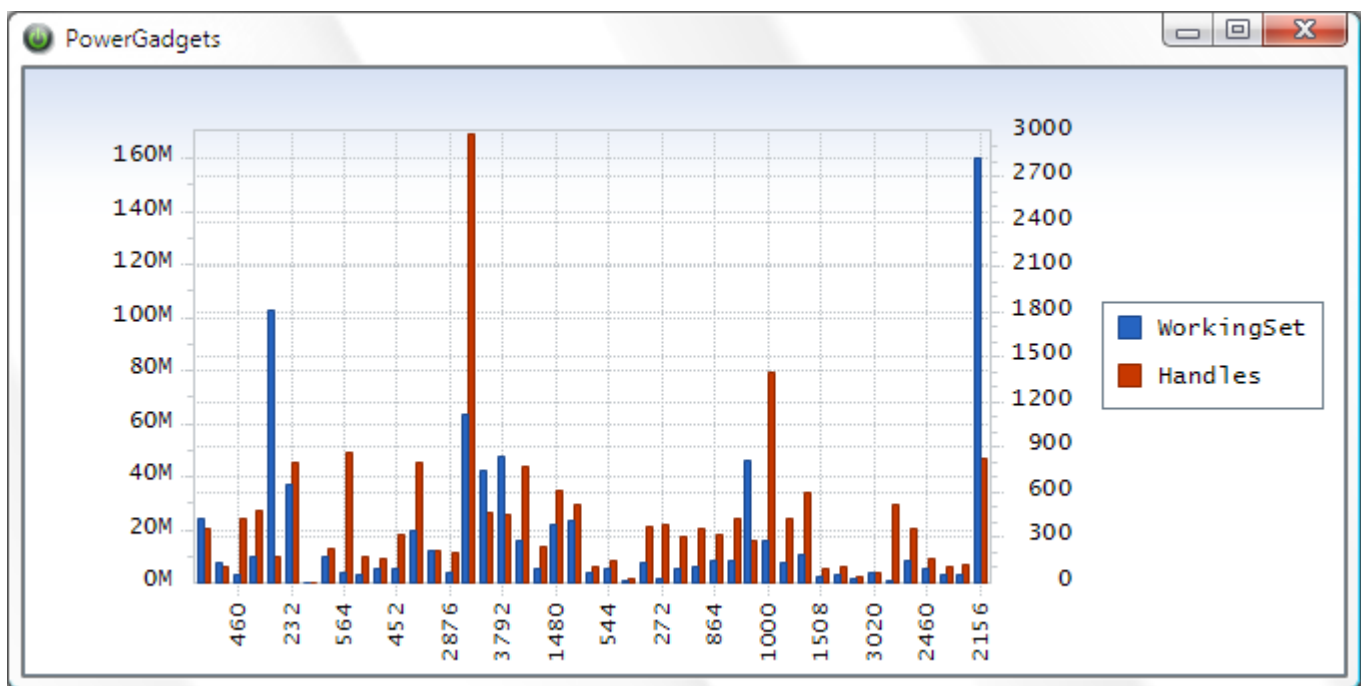


To customize the view for additional types, simply add a new View element node specifying the fully qualified name for the type to customize followed by as many Field elements you would like to customize following the same xml format shown above.

This behavior can be easily turned off by writing the following script:

```
out-chart -DisableAuto AxisY
```

## Appendix E. PowerGadgets out-gauge cmdlet Scripting FAQ.

### How do I select the gauge type to be displayed?

The out-gauge cmdlet wraps four different single-variable displays: Radial Gauges, Vertical Gauges, Horizontal Gauges and Digital Panels. Among these, Digital Panels are more flexible since they can display alphanumeric information.

You can use the –type property in your PowerShell script to select the desired gauge type as shown below.

```
Out-gauge -type radial
Out-gauge -type horizontal
Out-gauge -type Vertical
Out-gauge -type digital
```

Here for example we sum the files of the current directory and display the value through a radial gauge.

```
dir *.* | measure-object -sum | out-gauge -type radial -Value Sum
```

### How do I select the value to be plotted in a gauge?

As a single variable display, a gauge requires a value to be plotted. Based on this value a scale will be automatically created and displayed to give meaning to the value being plotted.  Since PowerGadgets can consume more complex .NET objects to plot data in charts and maps, when creating gauges you must be careful to return a single variable a gauge can used to instantiate itself.

If you are returning a more complex object as a result of your PowerShell script, you must use the –value property to specify which of the properties you want the gauge to display.

For example, the following script displays the CPU Usage by explicitly using the –value to indicate the property of a resulting object:

```
get-wmiobject -class Win32_PerfFormattedData_PerfOS_Processor | where {$_.Name -
eq "_Total"} | out-gauge -value PercentProcessorTime
```

### How do I change the default gauge style?

To facilitate usage, PowerGadgets provides pre-built styles for each gauge type (Radial, Vertical, Horizontal and Digital Panel). Styles provide different aesthetics by varying borders, tickmarks and other settings such as label positioning and scale settings.

You can easily change the default gauge visuals with the –style property by passing one of the available style numbers from 0 through 12 as shown.

```
Out-gauge -style 4
```

Please note the style property WILL NOT change default colors.  This setting will only change properties associated with labels, tickmarks and visuals such as borders and indicators, among others. If you want to change colors or make changes to specific gauge objects, please refer to the out-gauge parameters reference.

## How to access/configure specific objects in a gauge?

Although PowerGadgets attempts to setup appropriate default gauge settings, there maybe situations where you will want to access specific objects in a gauge, whether they are border styles, fonts, colors or even the way tickmarks are displayed. Gauges and other gadgets are fully configurable and provide a rich object model that allows you to configure virtually any setting from your PowerShell script.

You may need to reference the PowerGadgets parameter reference for the specific gadget you want to configure, or you may use the powerful auto tab completion which will let you navigate through the different properties and their settings with just a keystroke. In any case, you can use the underscore (_) convention to access complex properties in a gadget. For example, the following script changes the cap color and style of a radial gauge by accessing its related objects:

```
out-gauge -scales_0_Cap_Color Red -scales_o_Cap_Style Cap18
```

Please note, you can use PowerGadgets Graphical Wizards and templates which will allow you to manipulate these settings in a graphical environment, generate a template and use in your script as an effective way to simplify your PowerShell scripts. For more information on using templates please refer to "PowerGadgets Graphical wizard & PowerGadget Templates" chapter in this Quick Start Guide.

## How do I set the maximum value of a gauge?

When you populate a gauge, PowerGadgets will choose the maximum scale according to the value being passed. This auto-scale mechanism may not work in certain scenarios where the max is fixed or calculated based on other system or environmental settings. For example, the following script creates a Radial Gauge displaying  current page file usage and maximum page file size as the maximum value of the gauge:

```
get-wmiobject Win32_PageFileUsage | select Name,CurrentUsage,AllocatedBaseSize |
out-gauge -MainValue {$_.CurrentUsage} -MainScale_Max {$_.AllocatedBaseSize} -
refresh 0:0:1
```

## How do I change colors in a gauge?

When configuring gauge aesthetics, you can quickly setup colors by changing the gauge palette. Palettes will change ALL elements in a gauge based on a pre-defined and consistent set of colors. PowerGadgets provides over a dozen color palettes for all gadgets, including charts, gauges and maps.

The following script changes a gauge palette:

```
out-gauge -palette Autumn
```

In addition, you can change colors for individual objects by accessing each object's Color property. For example, the following script changes the color of the border and the indicator of a radial gauge:

```
out-gauge -border_color Blue -MainIndicator_Color Blue
```

Note: If you change colors for individual objects in a gauge, these objects will not respond to palette changes.

## How do I set titles in a gauge?

Identifying each gadget with what they're actually measuring is not only a good practice but may be critical for end user readability when displaying multiple gadgets in your desktop. You can quickly display a title in your gauge, as follows:

```
out-gauge -titles_0_Text "First Gauge"
```

```
out-gauge -type Vertical -titles_0_Text "First Gauge"
```

```
out-gauge -type Horizontal -titles_0_Text "First Gauge" titles_0_Layout_Alignment
BottomCenter
```

```
out-gauge -type Digital -titles_0_Text "First Gauge"
```

The PowerGadgets designer lets you visually position titles and images on the gauge surface. To do this, use the –config switch to generate a template that can later be incorporated in your script using the –template property. For more information on using templates please refer to "PowerGadgets Graphical wizard & PowerGadgets Templates" chapter in this Quick Start Guide.

## How do I set color stripes and other visual cues in a gauge?

Most gauges provide visual cues to warn the user about important ranges or values in the scale. For example, in a thermometer display you may want to highlight with colors a range of temperatures. Similarly, you may want to highlight a single value in a gauge to indicate certain condition has been met.

You can easily create a color stripe as follows:

```
out-gauge -type horizontal -border_style Thermometer01 -MainScale_Sections_0_Min
80 -MainScale_Sections_0_Max 100 -MainScale_Sections_0_Color Yellow
```

In addition to color stripes, you may change individual ranges in a scale by creating scale sections that change not just colors but other scale elements such as labels and tick marks. The following script creates a stripe in a horizontal gauge with different label and tick mark settings:

```
out-gauge -type horizontal -border_style Thermometer01 -MainScale_Sections_0_Min
80 -MainScale_Sections_0_Max 100 -MainScale_Sections_0_Color Blue -
MainScale_Sections_0_TickMarks_Major_Step 2 -
MainScale_Sections_0_TickMarks_Major_Label_Color Red
```

## How do I add an inner digital panel to a circular gauge?

PowerGadgets lets you combine gauges in a single display to create powerful gadgets that are easy to read and analyze. A common practice is to add digital panels with information as part of a radial gauge to improve the gauge readability.

The following PowerShell scripts creates and displays a radial gauge with an inner digital panel:

```
out-gauge -InnerGauges_Add Digital -InnerGauges_0_DigitalPanel_Value 20 -
InnerGauges_0_Layout_Alignment BottomCenter
```

## How do I add images to a gauge?

Images are also useful to identify gauges and what are they measuring. For example, you can use icons instead of titles to identify a particular gauge. The following script shows you how to add an image to a gauge via a PowerShell script.

```
out-gauge -Images_Add Bitmap -Images_0_Image "c:\camera.gif"
```

You can also align the image such as:

```
out-gauge -Images_Add Bitmap -Images_0_Image "c:\camera.gif" -
Images_0_Layout_Alignment BottomCenter
```

Images can also be added outside the border, creating interesting mask effects as follows:

```
out-gauge -Images_Add Bitmap -Images_0_Image "c:\camera.gif" -
Images_0_Layout_Alignment TopLeft -Images_0_VerticalPosition OutsideBorder
```

The PowerGadgets designer lets you visually position titles and images on the gauge surface. To do this, use the –config switch to generate a template that can later be incorporated in your script using the –template property. For more information on using templates please refer to "PowerGadgets Graphical wizard & PowerGadgets Templates" chapter in this Quick Start Guide.

## How to change the gauge border?

If you are displaying a gauge floating on the desktop or integrated into the Windows Vista sidebar, the border will play an important role as a windowing effect is created around the gauge border. PowerGadgets has a myriad of borders you can use to differentiate your gauges. The following script changes the gauge border:

```
out-gauge -Type Horizontal -Border_Style LinearBorderStyle01
```

For additional information on borders enumerations and their respective names you can use PowerGadgets AutoTab completion or simply refer to the PowerGadgets documentation or options listed in the PowerGadgets Graphical Wizard.

## How to change a gauge main indicator style?

As vector-based objects, gauges let you choose among a wealth of different indicator types. The following script changes the default border type and its intrinsic properties like colors, size, and style:

```
out-gauge -MainIndicator_Style Needle07 -MainIndicator_Size .8 -
MainIndicator_Color Blue
```

## Appendix F. PowerGadgets out-map cmdlet Scripting FAQ.

### How do I select a different map?

PowerGadgets provides a wealth of maps you can use to display geographical information in addition to associating conditional colors to sections based on the metrics of the information.   In addition you can also create custom geographical maps to suit your needs or even maps for non-geographical situations such as airplane seating charts and more.  To use a specific map, you must use the –mapsource property to load the desired maps from either the PowerGadgets map repository or from any other drive if you fully qualify the source name. For example, you could use the following script with the out-map cmdlet to load a world map:

```
out-map -mapsource World\WorldCountries
```

NOTE:  There are many maps available and to facilitate selecting the appropriate map, we have created a special  profile script for automatic tab completion.  To make use of this, you will need to copy the PSConfiguration folder, residing within the PowerGadgets installation folder, to the My Documents folder and restart power shell with power gadgets shell.  Now, when typing -mapsource followed by a space you can use tab repetitively to view the various maps available. You need to use the path separator '\' in conjunction with tab to see the various nested maps.

You can also load a custom map by qualifying the full path to the map vector file:

```
out-map -mapsource c:\mymaps\mycustomap
```

NOTE:  Depending on your deployment model, you may have your custom maps on a shared network drive.  You can additionally place the custom map in the Maps\Custom Maps sub folder of the PowerGadgets installation folder so it is available in the tab completion.

### How do I identify the map objects and their respective names?

When working with map files it is imperative you know the object names so you can match your data to the objects in the map and allow PowerGadgets to apply conditions to color each particular object according to the conditional attributes. To facilitate this process the  out-map cmdlet provides a –listobjects switch that provides a list of objects within a given map, as follows:

```
out-map -mapsource World\WorldCountries -listobjects
```

In some instances you will not be able to modify your data source so it matches the object names in a map. In these circumstances you must create a LabelLink XML file that provides a bridge between your data source and the objects in a map.  This process starts by recognizing the objects in a map and creating and XML file that the out-map cmdlet can read to make the translation from your original data source to the objects in the map.  You can easily create a LabelLink file by invoking the createlabellinkfile switch in the out-map c,dlet as follows:

```
out-map -mapsource World\WorldCountries -createlabellinkfile countries.xml
```

**How do I color map objects?**

Based on the structure of your data source, PowerGadgets will color map objects according to the following logic:

1)  If the data source has a single series, you must create conditional attributes to color map objects.
2)  If the data source has more than 1 series,  PowerGadgets will determine the winning series based on a summary function (by Default, the Max of all series) and then use the color of the winning series as the object color.

For example, if you have the following temperature data for US States:

| | |
|---|---|
| FL | 92 |
| NY | 78 |
| MA | 73 |
| VA | 78 |
| GA | 83 |

And you want to color states based on a given temperature scale, you must specify conditional attributes as follows:

```
out-map –mapsource US\USA-Regions-StatesAbrev.svg
-conditionalAttributes_0_condition_from 90
-conditionalAttributes_0_condition_To 120
-conditionalAttributes_0_Color Red
-conditionalAttributes_0_Text "Hot"
-conditionalAttributes_1_condition_from 60
-conditionalAttributes_1_condition_To 89
-conditionalAttributes_1_Color Orange
-conditionalAttributes_1_Text "Mild"
-conditionalAttributes_2_condition_from 20
-conditionalAttributes_2_condition_To 59
-conditionalAttributes_2_Color Blue
-conditionalAttributes_2_Text "Cold"
```

Please note you can create conditional attributes using the PowerGadgets Creator and then apply a template containing those conditional attributes. For additional information, please refer to "Using Templates" in previous chapters.
On the other hand, if you have a data source containing multiple series like the following US Election Data:

| | Bush | Gore |
|---|---|---|
| ARIZONA | 781652 | 685341 |
| ARKANSAS | 472940 | 422768 |
| CALIFORNIA | 4567429 | 5861203 |
| COLORADO | 883748 | 738227 |
| DELAWARE | 137288 | 180068 |
| FLORIDA | 2912790 | 2912253 |
| GEORGIA | 1419720 | 1116230 |

PowerGadgets must first determine a winner series and then apply a color based on the series color. Therefore, if you want to change the color of the object in a map you will do it by changing the color series as follows:

```
out-map –mapsource US\USA-Regions-StatesAbrev.svg
-Series_0_Color Red
-Series_1_color Blue
```

Note: You can change the summary function that determines the winning series using the –WinnerSeriesType switch

## How do I create custom maps?

The out-map cmdlet provides a full library of dynamic maps.  These maps display a multitude of commonly used maps such as countries, states, and counties allowing you to plot and diagram various diverse sets of geographically related data.  Additionally, PowerGadgets supports custom diagrams through the use of the SVG standard such as a seating chart or any other diagram where data needs to be associated.  The SVG (Scalable Vector Graphics) format is a World Wide Web Consortium (W3C) specification for a standard, two-dimensional vector graphics language.

Please contact Software FX at support@softwarefx.com for additional information on how to prepare an SVG file to be used with out-map cmdlet.