Hydra - Developer Guide

This guide is for the people who want to write Sequences for Hydra. It describes the formats, keywords and concepts needed to create and run sequences.
To be able to develop Hydra Sequence, you need some PowerShell notions.

Sequences definition

Hydra Sequences are one or a succession of steps, all ps1 files, set in a defined order and that will run sequentially on an object.
The steps and settings relative to a sequence are stored in a file with the extension .sequence.xml.
As a best practice, it is highly recommended to save this .sequence.xml file in the same subfolder than the relative ps1 and potential other files associated (installation sources, configuration files…).
It is also recommended to create one folder per sequence.

.sequence.xml files

.sequence.xml file define the process of tasks (or steps) to be performed: it is mainly composed of parameters and paths to the tasks to execute. Optionally, other settings like variables or modules can be defined.
The format used is XML: each task, parameter or variable definition has to be defined in its own XML node. A simple example of a sequence is:

```
<sequence>

<parameter name="sequencename" value="Check Gemalto Installation" > </parameter>
<parameter name="warning" value="No"> </parameter>
<parameter name="maxthreads" value="50"> </parameter>

<task path="..\IsOnline.ps1" comment="Check if the machine is online"> </task>
<task path="GemaltoCheck.ps1" comment="Gemalto Installation check"> </task>

</sequence>
```

A Sequence must always be encapsulated in the keyword <sequence> </sequence>. Each single line defines a parameter, a task or a variable (covered later in this document).

Keyword: parameter

To ease the reading of sequence files, Parameters should be placed at the beginning of the file.
The format of a parameter is:

```
<parameter name="[defined parameter]" value="[value assigned]"> </parameter>
```

The parameters are all optional.

sequencename

The parameter **sequencename** will display the name of the sequence in the sequence panel. It is highly recommended to set it.

```
<parameter name="sequencename" value="Install 7Zip" > </parameter>
```

warning

The parameter **warning** can be set to Yes or No (default is No). If it is set to Yes, a warning pops up before the start of the sequence. The user can then cancel the start.

```
<parameter name="warning" value="Yes"> </parameter>
```

securitycode

The parameter **securitycode** sets a value that the user will have to re-type to confirm the start of the sequence. It is a higher security level than the **warning** keyword. If set, the value of the **warning** keyword is ignored. The **securitycode** is case sensitive and the user has 3 retries.

```
<parameter name="securitycode" value="IwanttoInstall"> </parameter>
```

maxthreads

The parameter **maxthreads** defines the number of concurrent threads for the sequence. Try to define a good value for your sequence: a sequence mainly composed of simple queries can easily manage up to 100 threads at once. A sequence made to perform installations using the machine as a central deployment point will need a low number of threads: too high calculates maximum threads may result in bottleneck effects, high CPU or RAM usage,… The default of maximum threads is 10.

```
<parameter name="maxthreads" value="10"> </parameter>
```

maxobjects

The parameter **maxobjects** restricts the number of objects that can be selected for a sequence. This is particularly needed for sequences using variables (see below) applying to only one or a very small number of objects. If the number of selected objects exceeds the maxobjects value, the sequence doesn’t start.

```
<parameter name="maxobjects" value="1"> </parameter>
```

sendmail

The parameter **sendmail** generates a protocol sent per mail at the end of the sequence run. The default is No. This parameter is rarely used, as users can generate emails from the grid, whenever they want.

```
<parameter name="sendmail" value="Yes"> </parameter>
```

This parameter needs additional options:

```
<parameter name="mailserver" value="vip-m-parpop.fr.net.intra"> </parameter>
<parameter name="mailfrom" value="hydra@test.com"> </parameter>
<parameter name="mailto" value="user.name@bnpparibas.com"> </parameter>
<parameter name="mailreplyto" value="user.name@bnpparibas.com"> </parameter>
```

timer

The parameter **timer** defines the interval between the start of new threads, as well as GUI refreshes. It is set in ms, between 500 and 30000 (default is 1000). This parameter is generally used for debugging purposes.

```
<parameter name="timer" value="5000"> </parameter>
```

An example with the main parameters types could be:

```
<parameter name="sequencename" value="Uninstall KB3177725" ></parameter>
<parameter name="warning" value="Yes"> </parameter>
<parameter name="maxthreads" value="10"> </parameter>
```

Keyword: task

The most important keyword is "*task*". It needs two parameters:

- path: the path of the ps1 script to execute
- comment: the comment displayed in the sequence panel, on the left of the interface

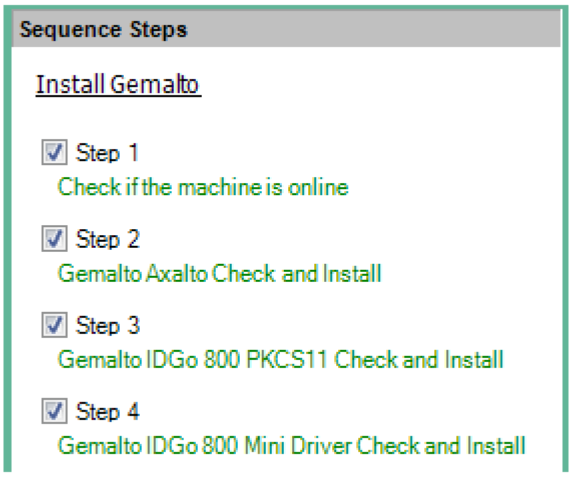
The format of a task is:

```
<task path="[path_to_ps1]" comment="[task comment in the GUI]"> </task>
```

The path can be relative to the sequence.xml itself (1st search), relative to the Hydra startup path (2nd search) or can be a full path.

```
<task path="..\..\IsOnline.ps1" comment="Check if the machine is online"> </task>
<task path="Gemalto_Axalto_Install.ps1" comment="Gemalto Axalto Check and Install"> </task>
<task path="Gemalto_IDGo800_PKCS11_Install.ps1" comment="Gemalto IDGo 800 PKCS11 Check and Install"> </task>
<task path="Gemalto_IDGo800_Minidriver_Install " comment="Gemalto IDGo 800 Mini Driver Check and Install"> </task>
```

The steps are then displayed in the same order in the sequence panel:


Sequence Steps

Install Gemalto

☑ Step 1
Check if the machine is online

☑ Step 2
Gemalto Axalto Check and Install

☑ Step 3
Gemalto IDGo 800 PKCS11 Check and Install

☑ Step 4
Gemalto IDGo 800 Mini Driver Check and Install

Keyword: variable

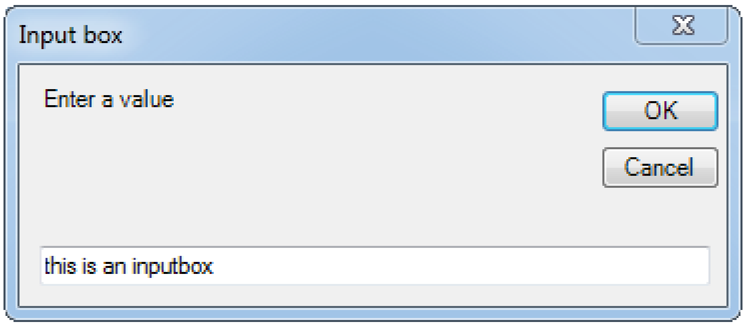
Variables are defined by the user at the start of a sequence and can be used in some or all ps1 files of this sequence. The developer defines the variables necessary. Variables can be a text, a selected file or folder, a dropdown entry, …
The format of a variable is:

<variable name="[variable name]" type="[variable type]" value="[list of parameters for this type of variable]"> </variable>

There are currently 9 types of variables, each with its own number of parameters.

inputbox

The variable **inputbox** will prompt for a string:


Input box

Enter a value

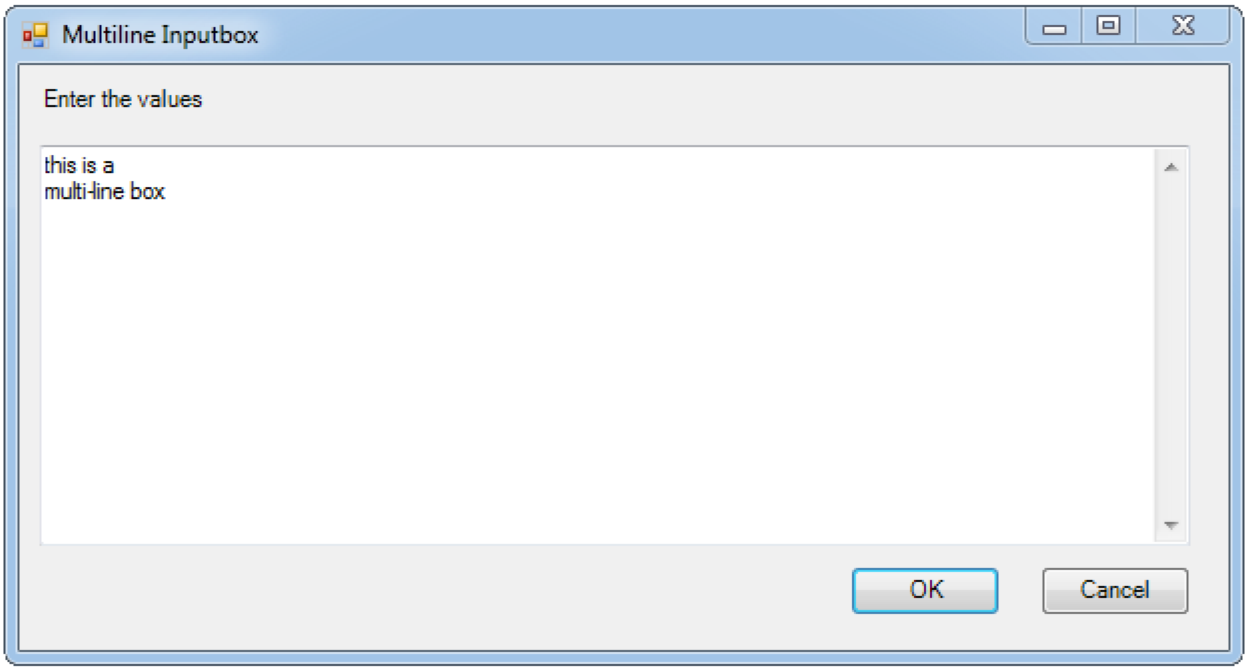
this is an inputbox

inputbox needs 3 parameters, separated by semicolons: Label, Text to display, Default value

<variable name="inputvar" type="inputbox" value="Input box;Enter a value;Default"> </variable>

multilineinputbox

The variable **multilineinputbox** will prompt for strings on multiple lines:



Enter the values

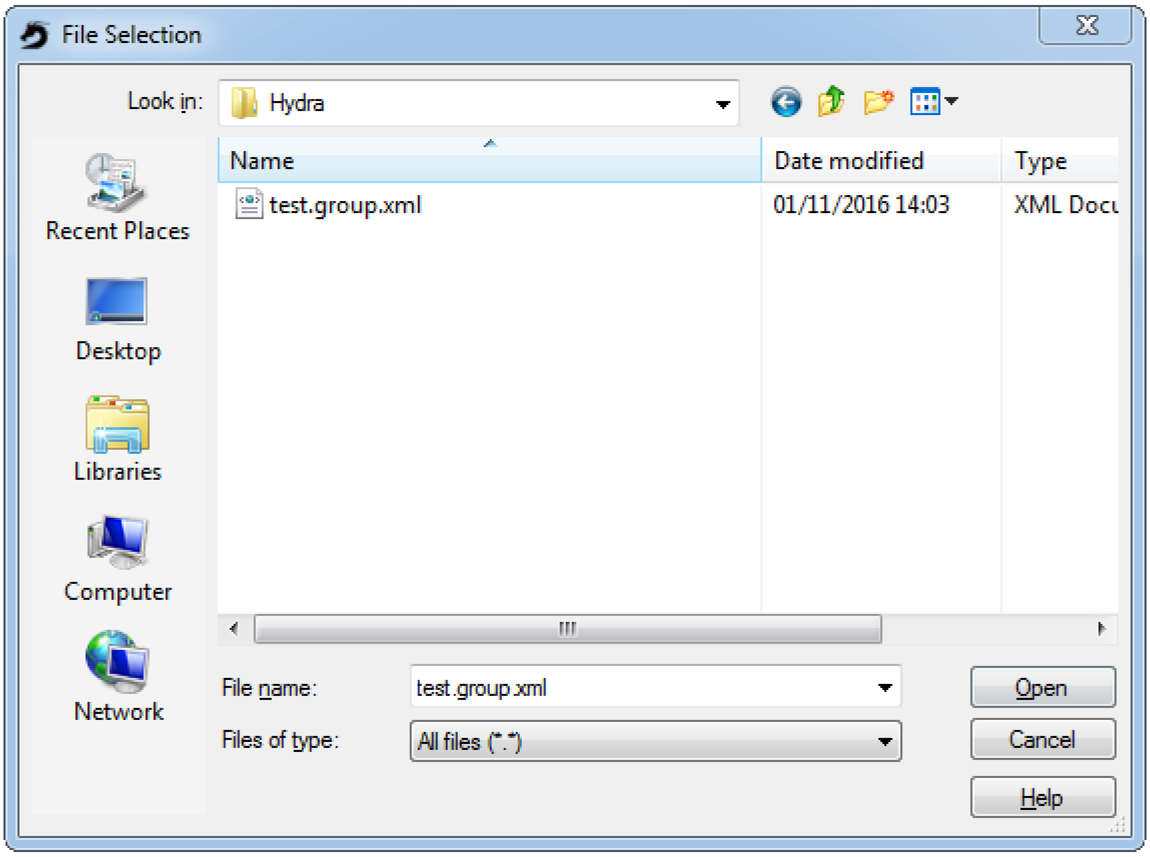
this is a
multi-line box

multilineinputbox needs 2 parameters, separated by semicolons: Label, Default value

<variable name="multilineinputvar" type="multilineinputbox" value="Multiline Inputbox;Enter the values"> </variable>

selectfile

The variable **selectfile** lets the user select a file. The path of this file is stored in the variable.



test.group.xml 01/11/2016 14:03 XML Docu

File name: test.group.xml

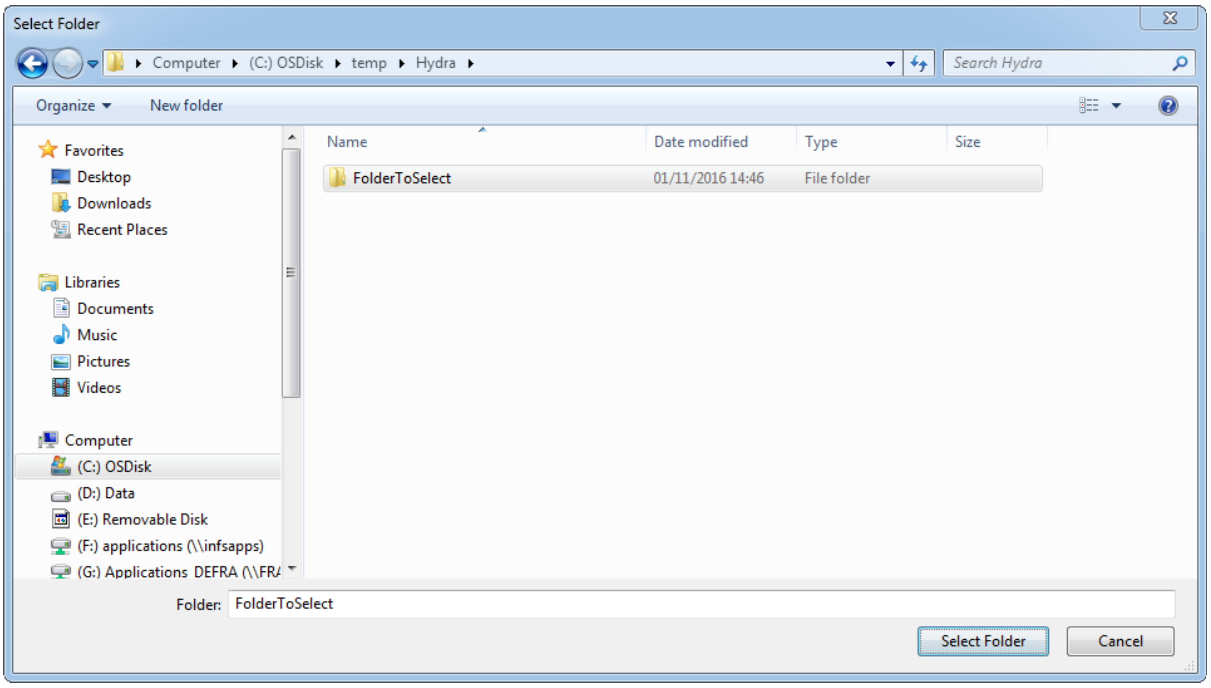
Files of type: All files (*.*)

selectfile needs 3 parameters: Label, Default folder, list of extensions to filter

```
<variable name="selectedfile" type="selectfile" value="File Selection;C:\temp;All files (*.*)|*.*">
</variable>
```

selectfolder

The variable **selectfolder** lets the user select a folder. The path of this folder is stored in the variable.

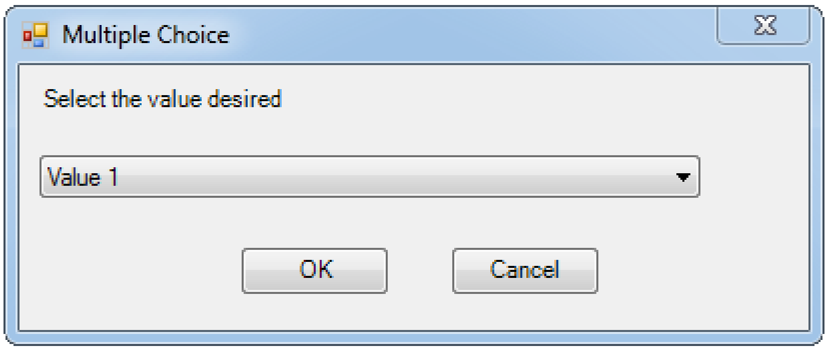


selectfolder needs 1 parameter: Default folder

<variable name="selectedfolder" type="selectfolder" value="C:\temp"> </variable>

combobox

The variable **combobox** lets the user choose an option in a dropdown menu.



combobox needs 3 parameters: Label, Text to display, list of entries in the combobox
The list of entries can be a list separated by commas, or a definition file.

<variable name="combovalue" type="combobox" value="Multiple Choice;Select the value desired;Value 1,Value 2,Value 3"> </variable>

or

<variable name="combovalue" type="combobox" value="Multiple Choice;Select the value desired;.\Bin\Demos\Variables\clOptions.txt"> </variable>

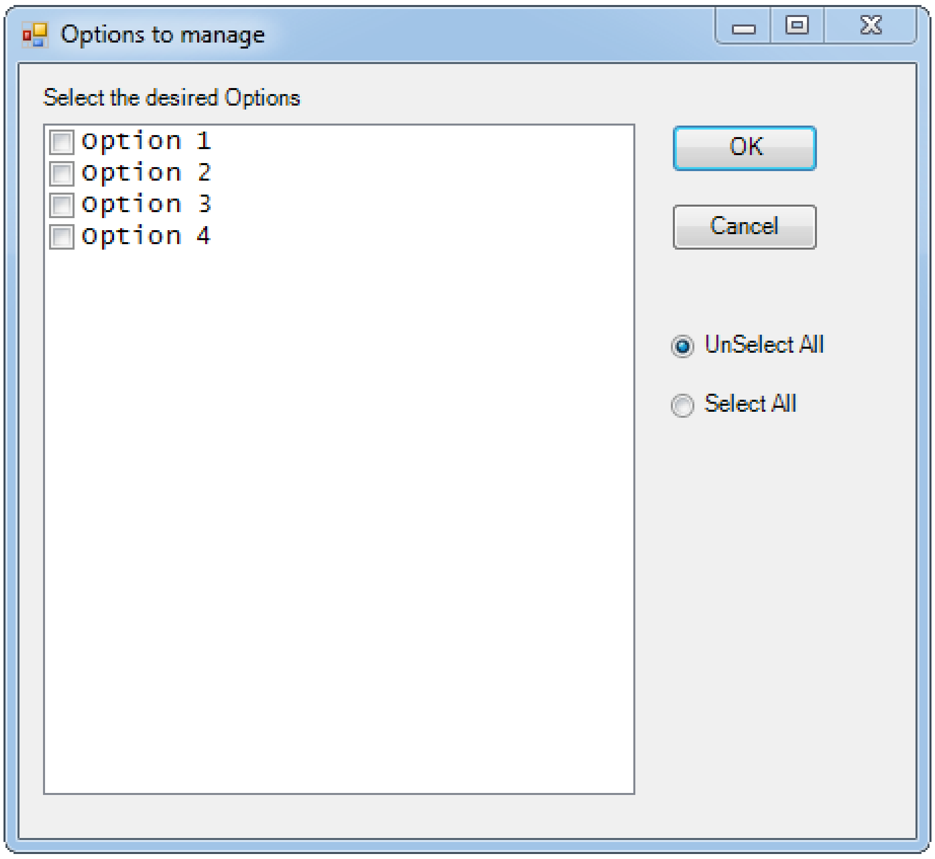
The definition file (clOptions.txt in the example) is an ASCII file with the name of each entry:

clOptions.txt

```
Value 1
Value 2
Value 3
```

multicheckbox

The variable **multicheckbox** displays a list of checkboxes:



multicheckbox needs 3 parameters: Label, Text to display, list of entries
The list of entries can be a list separated by commas, or a definition file.

<variable name="Options" type="multiCheckbox" value="Options to manage;Select the desired Options;Option 1,Option 2,Option 3,Option 4"> </variable>

or

<variable name="Options" type="multiCheckbox" value="Options to manage;Select the desired Options;.\Bin\Demos\Variables\cloptions.txt"> </variable>

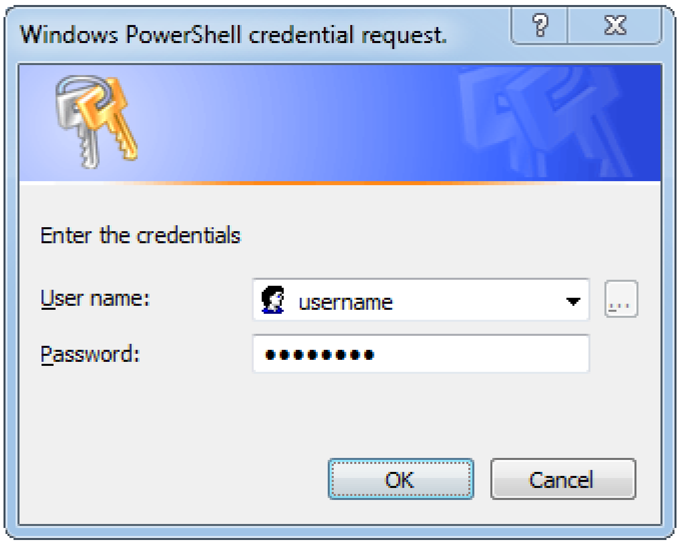
The definition file (clOptions.txt in the example) is an ASCII file with the name of each checkbox:

clOptions.txt

```
Option 1
Option 2
Option 3
Option 4
```

credentials

The variable **credentials** displays a credentials window and save them in a PSCredential object.

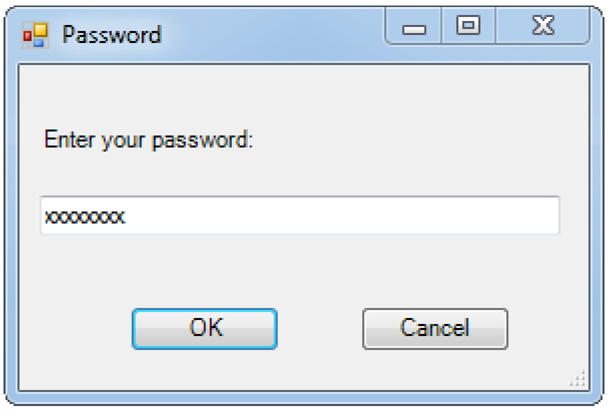


credentials needs one parameter: Text to display

<variable name="cred" type="credentials" value="Enter the credentials"> </variable>

secretinputbox

The variable **secretinputbox** prompts for a hidden string, where the characters typed are replaced with x. It is typically used to prompt for a password.

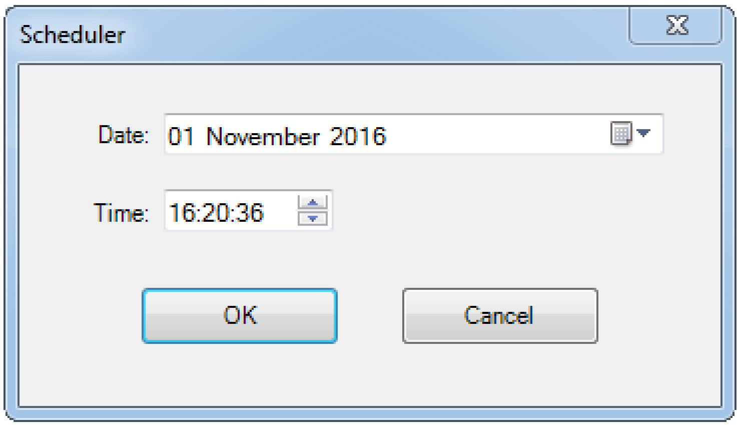


secretinputbox needs 2 parameters: Label, Text to display

<variable name="secret" type="secretinputbox" value="SecretBox;Enter the password for test:"> </variable>

scheduler

The variable **scheduler** displays a window to let the user choose the date and time of the sequence start.



Date: 01 November 2016

Time: 16:20:36

scheduler needs one parameter: Label

```
<variable name="sched" type="scheduler" value="Scheduler"> </variable>
```

Keyword: preload

In some situations, it may be helpful to have steps running once, independent of the objects, before the "real" steps are launched. This can be done with the **preload** keyword. Preload act exactly the same than tasks, except that they run only once, at the beginning of the sequence: during this phase, the objects are on hold and will start with their own steps only when the preloads will have finished. Preload can use the variables defined in the sequence.xml file. If shared variables are returned by a preload sequence (see the shared variables later in this document), they are passed to all objects pending.
Preload usually load modules, create and pass variables to the objects pending, generate files that the further steps will rely on, …

```
<preload path="first_operation.ps1" comment="First thing to do"> </preload>
```

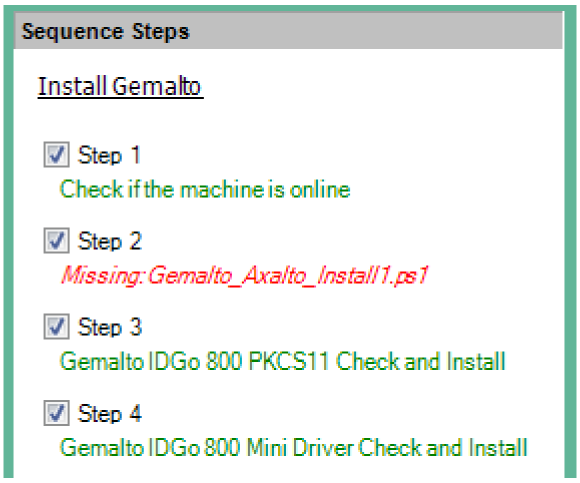
Keyword: importmodule

Some tasks need (3rd party) PowerShell snap-ins or modules that need to be loaded. Most of these modules can't be loaded parallelly when several tasks are running. This is a restriction of the Runspaces technology on which Hydra is based.
To use such snap-ins or modules, simply define them in the sequence.xml and use their commands in the tasks without re-importing the modules.
Depending on the type of the module you need to load, use one of the 3 importmodule type: ImportPSSnapIn, ImportPSModulesFromPath or ImportPSModule:

```
 <importmodule type="ImportPSSnapIn" name="Citrix.Common.Commands"> </importmodule>
 <importmodule type="ImportPSSnapIn" name="Citrix.XenApp.Commands"> </importmodule>
 <importmodule type="ImportPSSnapIn" name="Citrix.Common.GroupPolicy"> </importmodule>
 <importmodule type="ImportPSModulesFromPath" name="testpath"> </importmodule>
 <importmodule type="ImportPSModule" name="DellPEPowerShellTools"> </importmodule>
```

Processing

Once loaded, the paths of all tasks are first verified. If a path is not valid or can’t be found, it is displayed in red in the sequence panel and the sequence can't be started.



Sequence Steps

Install Gemalto

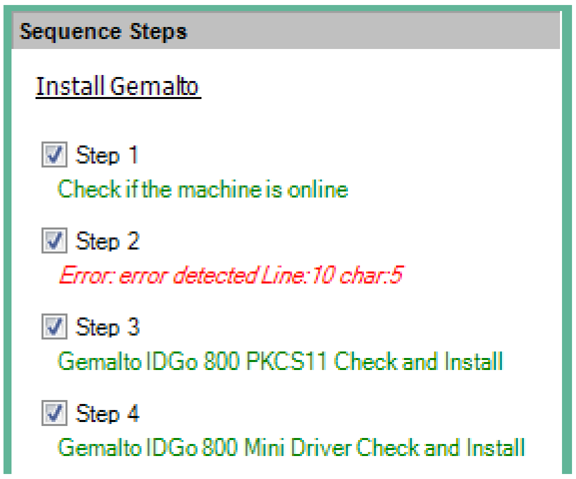
☑ Step 1
Check if the machine is online

☑ Step 2
Missing: Gemalto_Axalto_Install1.ps1

☑ Step 3
Gemalto IDGo 800 PKCS11 Check and Install

☑ Step 4
Gemalto IDGo 800 Mini Driver Check and Install

If a syntax error is detected, it is also displayed in the sequence panel:



Sequence Steps

Install Gemalto

☑ Step 1
Check if the machine is online

☑ Step 2
Error: error detected Line:10 char:5

☑ Step 3
Gemalto IDGo 800 PKCS11 Check and Install

☑ Step 4
Gemalto IDGo 800 Mini Driver Check and Install

The tasks are launched in the order they are defined in the sequence file. The current step of every object being processed is displayed in nearly real-time in the grid.
If a task fails on a specific object, because of a script error, the whole sequence is stopped for this object. The step on which the sequence has stopped is displayed in the grid with the cause of the stop: it can be a runtime error (division by zero, empty variables…), no objects returned, unknown or too much objects returned.
A stop at a specific step doesn't mandatory means that the main part of the sequence has failed or that the ps1 script associated to the task is faulty: for example, if a reboot is needed to complete the sequence, but you don't want to force it when a user is logged on, the sequence can stop (or break) before a reboot is initiated.

Example
This sequence is used for the installation of the Checkpoint Endpoint Security tool:

XML code:

```
<sequence>

  <parameter name="sequencename" value="Install Checkpoint" > </parameter>
  <parameter name="warning" value="Yes"> </parameter>
  <parameter name="maxthreads" value="5"> </parameter>

  <task path="..\..\IsOnline.ps1" comment="Check if the machine is online"> </task>
  <task path="CheckPointPreCheck.ps1" comment="Checkpoint Prechecks"> </task>
  <task path="CheckPointDeploy.ps1" comment="Remote installation of CheckPoint"> </task>
  <task path="CheckPointPostCheck.ps1" comment="Check the Checkpoint services"> </task>
  <task path="LoggedOnUser.ps1" comment="Check if a user is logged on"> </task>
  <task path="Reboot.ps1" comment="Rebooting"> </task>
  <task path="WaitUntilOnline.ps1" comment="Waiting for machine to be up"> </task>

</sequence>
```

Explanations:

Step 1: If the machine is not online: Stop here

Step 2: Requirements: if Checkpoint is already installed or if the registry is not reachable: Stop here
If a specific registry key is found avoiding the software to install correctly, it is removed

Step 3: Installation: the good msi file (laptop or workstation) is remotely copied and remotely executed. The task exits when the cmd process is not running anymore

Step 4: Post Checks: if the Checkpoint services are not installed: Stop here

Step 5: If a user is logged on: Stop here. To force a reboot, even if someone is logged on, the user will have to uncheck this step

Step 6: Start a reboot and wait until the ping doesn't answer anymore

Step 7: Wait that the machine is up again

If the Step 7 successfully completes, the machine is marked as green per default in the grid. In the other cases, it is marked as red, or blue (see the Tasks part for the colors explanations). If other colors are defined in the tasks, the output color may differ.

Sequences Lists

The sequences listed in the sequence tree are defined in a Hydra Sequence List file. Such a file is located in the subfolder Settings. At the start of Hydra, a Sequence List relative to the country (based on the DNS suffix) is searched in this folder. If such a Sequence List file is found, this one will be used. If not, the default file **Hydra_Sequences.lst** is used. A country Sequence List file is called Hydra_Sequences_CC.lst, where CC is the DNS country code of the machine where Hydra is currently run: Hydra_Sequences_DE.lst and Hydra_Sequences_ES.lst are example of Sequences List files for Germany and Spain. To use another Sequence List or force the use of a specific one at the start, define it as parameter at the run of Hydra.

A Sequences List is composed of the Name of the sequence (displayed in the sequences tree) and the location of the respective sequence.xml, separated by a semicolon. Lines containing "--" are ignored and can be used as separators.

Example:

```
----- Software Checks -----
Check CheckPoint;.\bin\Checks\CheckpointVersionCheck\CheckPointVersionCheck.Sequence.xml
Check Firefox Version;.\bin\Checks\FirefoxVersionCheck\FirefoxVersionCheck.Sequence.xml
Check Gemalto Installation;.\bin\Checks\Gemalto\GemaltoCheck.Sequence.xml
Check Internet Explorer Version;.\bin\Checks\IECheck\IECheck.Sequence.xml
----- Installations -----
Install 7-Zip;.\bin\Installation\7Zip\7ZipInstallation.Sequence.xml
Install Adobe Acrobat;.\bin\Installation\Acrobat\AcrobatInstallation.Sequence.xml
Install Checkpoint Endpoint
Security;.\bin\Installation\Checkpoint\CheckpointInstallation.Sequence.xml Install Citrix Receiver
3.4;.\bin\Installation\CitrixReceiver\Citrix34Installation.Sequence.xml
Install Citrix Receiver 4.4;.\bin\Installation\CitrixReceiver\Citrix44Installation.Sequence.xml Install
----- OS Tasks -----
Display the current user logged on;.\bin\Tasks\LoggedOnUser\LoggedOnUser.Sequence.xml
Ping Machines (DNS Query);.\bin\Tasks\PingDNS\PingDNS.Sequence.xml
Reboot Machines;.\bin\Tasks\RebootMachines\RebootMachines.Sequence.xml
```

The paths are relative to the Hydra startup path.

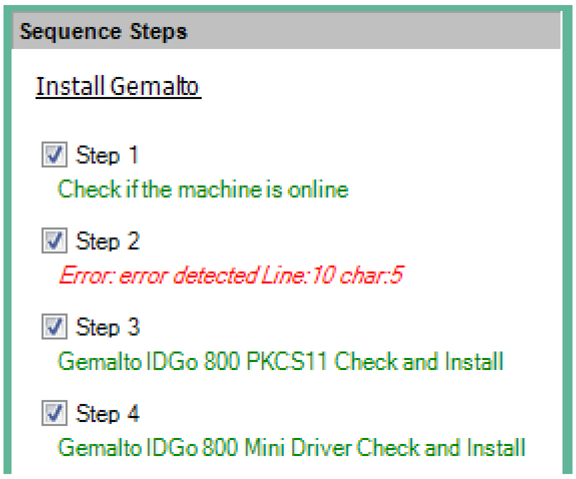
Task Files

The tasks are the core of the sequences: they do the different actions needed and return the result which will influence on the progression of the whole sequence.

Format

Task files can't be fully checked for potential coding mistakes. Be sure to format it correctly:
- They have to be written in PowerShell code
- Tasks are basically PowerShell ScriptBlocks without name definition
- They need to have a ps1 extension
- They have to return 2,3 or 4 values or objects (see below)

If an error in the code is detected, the task is marked in red when it's loaded. If possible, the position of the first error is also displayed.



Sequence Steps

Install Gemalto

☑ Step 1
Check if the machine is online

☑ Step 2
Error: error detected Line:10 char:5

☑ Step 3
Gemalto IDGo 800 PKCS11 Check and Install

☑ Step 4
Gemalto IDGo 800 Mini Driver Check and Install

Try to run the ps1 as stand-alone script to detect the error.

As tasks are working with objects, like machine names, they need this object passed as parameter. A typical task looks like:

```
param ($machine)

 $Online=Test-Connection -ComputerName $machine -Count 1 -Quiet

 if ($Online) {
   return "OK", "$machine is online"
 }
 else {
   return "BREAK", "$machine is offline"
 }
```

The variable name used for the object ($machine in the example), can be freely defined and can be different in every task.

Return values

Every Task has to return correct objects, corresponding to the result state of the script and what has to be displayed in the GUI. If there are some mistakes with the objects returned (unknown, too much, empty,…), the sequence will stop at this step.

The values returned by the tasks must be in the form:

STATE, COMMENT [,CELL STYLE] [,SHARED VARIABLE]

STATE

The state has to be one of the following keywords:

- **OK:** the task has completed successfully, the next one can be executed. This will be displayed as default in green in the grid (or with the color defined as a 3rd parameter, or the one individually set by the user in his own settings).
- **STOP:** the task has failed or some requirements were not met. The sequence will be stopped for the object; the sequence has failed. This will be displayed as default in red in the grid (or with the color defined as a 3rd parameter, or the one individually set by the user).
- **BREAK:** some requirements were met or not met, preventing the sequence to continue. However, this is not related to the deployment itself and is not considered as failed. For example, a reboot is avoided if a user is logged on: the sequence is in state BREAK, not complete, but also not failed. This will be displayed as default in blue in the grid (or with the color defined as a 3rd parameter, or the one individually set by the user).

STOP and BREAK have the same behavior but can be used differently depending on the cases. The proper keyword to use is the decision of the sequence developer.

COMMENT

The comment will be displayed in the 3th column of the grid and will give the information and result of a task. Try to use a human readable string for a best reading.

CELL STYLE

The color of the cell, as well as the color and style of its font can be individually set. This optional returned value is set like this:

Cell Background Color|Font Type|Font Color

The 2nd and 3rd parameters are optional.

If a color is defined for the Cell background, it will override the default one associated to the STATE.

The colors for the cell background and the font have to be a valid official HTML color or defined in HEX format like "#FF90EE90" (for green), "#FFADD8E6" (for blue) or with their official names like described on http://htmlcolorcodes.com/color-names. As it is first interpreted as a String, the whole value has to be set between double quotes ".

The Font Type can be simple or complex.

A simple value is: Regular, Italic, Bold, Underline or Strikethrough. A wrong keyword will be ignored.

A complex value is a .NET Drawing.Font format, like *'Lucida Console',8.25,3,3,0*. Refer to the .NET documentation for the different Font formats allowed.

SHARED VARIABLE

You can use the 4th parameter to pass any variable type (string, number, array, object, hash,…) to the next task. A shared variable is kept and passed to all the next steps until it is replaced by a new one, returned by another task.
To use a shared variable in a task, extend the ***param*** of the task with a 2nd argument:

param ($machine, $sharedvar)

The name of the arguments can be freely set.

If no CELL STYLE has been defined, you <u>need to set</u> the 3rd parameter to $NULL.

Best practice

- ✓ The values generated can be returned with or without the keyword "return".
- ✓ A task script must only return this information! Any other object created and passed for the output will be interpreted and will result in wrong information displayed in the console. Functions or ScriptBlocks in PowerShell return everything that is not assigned to variables: be sure to take this in consideration when you will write your tasks: use variables to store objects that don’t need to be returned, pipe them to out-null or use [void].
- ✓ If the returned values are wrong or can’t be interpreted, an error message will be displayed and the sequence will stop for this object: such errors are marked in grey.

Constants

A few constants relative to the script or sequences paths are available in every task.

$MyScriptInvocation

$MyScriptInvocation is the equivalent of $MyInvocation in PowerShell and has different properties. To use or display the name of a task script, use $MyScriptInvocation.MyCommand or $MyScriptInvocation.InvocationName to get its full path.

$SequencePath

$SequencePath returns the path of the sequence as it is setup in the Hydra_Sequences.lst (or equivalent). If it is set as a relative path, it will be displayed in this form. A manual load of a sequence always returns the full path of the sequence.

$SequenceFullPath

$SequenceFullPath has the same function than $SequencePath, but always displays the full and absolute form of the path.

$CentralLogPath

$CentralLogPath is the path defined in the settings with the name of the user automatically added.

Examples

Logged on user

The following sequence queries the current logged on user, and if no user is logged on, returns the last connected one. A customized color (grey, "#FFCCCCCC") has been defined for machines where nobody is logged on.

LoggedOnUser.Sequence.xml

```
<sequence>

  <parameter name="sequencename" value="Display the current logged on user" > </parameter>
  <parameter name="warning" value="No"> </parameter>
  <parameter name="maxthreads" value="100"> </parameter>

  <task path="..\..\IsOnline.ps1" comment="Check if the machine is online"> </task>
  <task path="LoggedOnUser.ps1" comment="Check who is logged on"> </task>

</sequence>
```

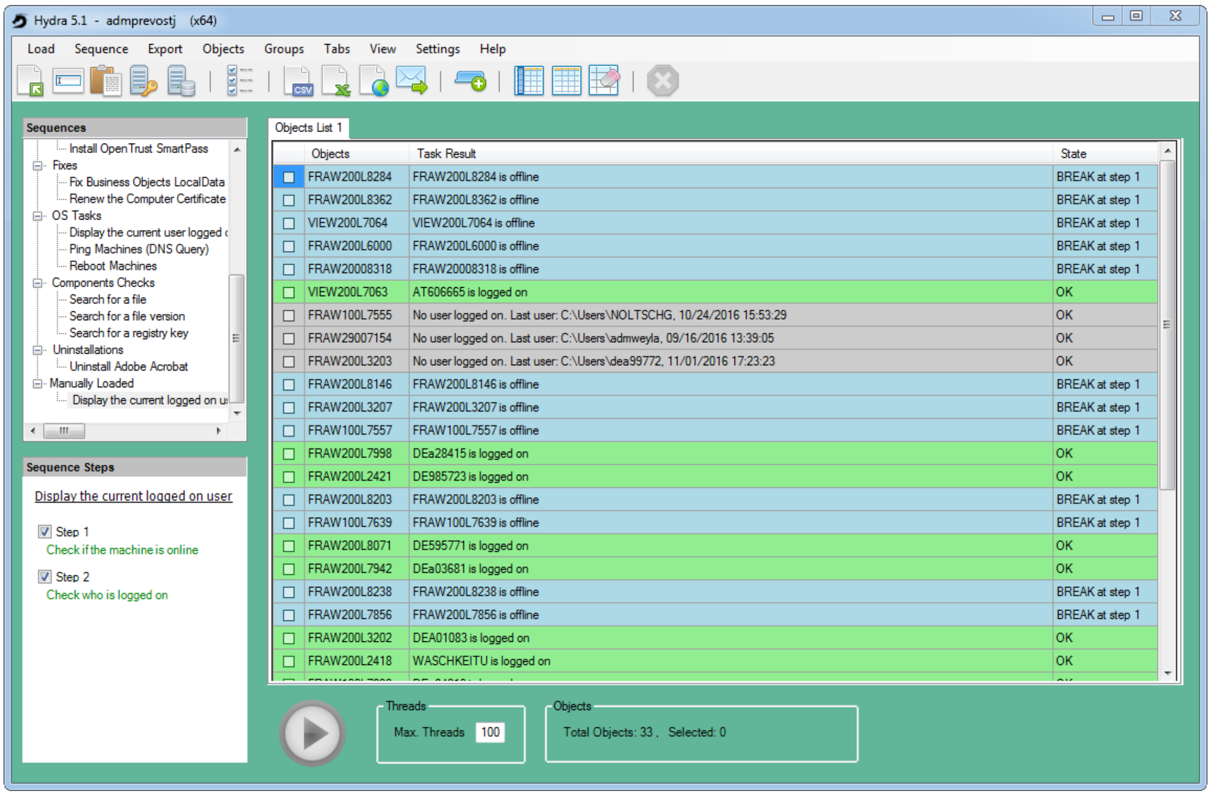
IsOnline.ps1

```
Param ($wks)

$Online=Test-Connection -ComputerName $wks -Count 1 -Quiet

if ($Online) {
  return "OK", "$wks is online"
}
else {
  return "STOP", "$wks is offline"
}
```

LoggedOnUser.ps1

```
Param ($wks)

  try {
    $User=(Get-WmiObject -Class win32_process -Computer $wks -Filter "name='explorer.exe'" | Foreach-
Object { $_.GetOwner() }).User
  }

  catch {
    return "STOP", "Unable to connect to Win32_Process"
  }

  if ($User -ne $NULL) {
    return "OK", "$User is logged on"
  }


  try {
    $Win32User=Get-WmiObject -Class Win32_UserProfile -ComputerName $Wks -ErrorAction Stop
  }

  catch {
    return "STOP", "Unable to connect to Win32_UserProfile"
```

```
 }

  $LastUserLoggedOn=($Win32User | Where { $_.LocalPath -like "C:\users\*" } | Sort -Descending
LastUseTime | Select @{ l="LastUser" ; e={ $_.LocalPath + ", " +
$_.ConvertToDateTime($_.LastUseTime)} }) | Select -First 1 -ExpandProperty LastUser

  return "OK", "No user logged on. Last user: $LastUserLoggedOn", "#FFCCCCCC"
```



Hydra 5.1 - admprevostj (x64)
Load
Sequence
Export
Objects
Groups
Tabs
View
Settings
Help
Sequences
Install OpenTrust SmartPass
Fixes
Fix Business Objects LocalData
Renew the Computer Certificate
OS Tasks
Display the current user logged
Ping Machines (DNS Query)
Reboot Machines
Components Checks
Search for a file
Search for a file version
Search for a registry key
Uninstallations
Uninstall Adobe Acrobat
Manually Loaded
Display the current logged on u
Sequence Steps
Objects List 1
Threads
Objects

Display the current logged on user

Step 1
Check if the machine is online

Step 2
Check who is logged on

Objects	Task Result	State
FRAW200L8284	FRAW200L8284 is offline	BREAK at step 1
FRAW200L8362	FRAW200L8362 is offline	BREAK at step 1
VIEW200L7064	VIEW200L7064 is offline	BREAK at step 1
FRAW200L6000	FRAW200L6000 is offline	BREAK at step 1
FRAW20008318	FRAW20008318 is offline	BREAK at step 1
VIEW200L7063	AT606665 is logged on	OK
FRAW100L7555	No user logged on. Last user: C:\Users\NOLTSCHG, 10/24/2016 15:53:29	OK
FRAW29007154	No user logged on. Last user: C:\Users\admweyla, 09/16/2016 13:39:05	OK
FRAW200L3203	No user logged on. Last user: C:\Users\dea99772, 11/01/2016 17:23:23	OK
FRAW200L8146	FRAW200L8146 is offline	BREAK at step 1
FRAW200L3207	FRAW200L3207 is offline	BREAK at step 1
FRAW100L7557	FRAW100L7557 is offline	BREAK at step 1
FRAW200L7998	DEa28415 is logged on	OK
FRAW200L2421	DE985723 is logged on	OK
FRAW200L8203	FRAW200L8203 is offline	BREAK at step 1
FRAW100L7639	FRAW100L7639 is offline	BREAK at step 1
FRAW200L8071	DE595771 is logged on	OK
FRAW200L7942	DEa03681 is logged on	OK
FRAW200L8238	FRAW200L8238 is offline	BREAK at step 1
FRAW200L7856	FRAW200L7856 is offline	BREAK at step 1
FRAW200L3202	DEA01083 is logged on	OK
FRAW200L2418	WASCHKEITU is logged on	OK

Max. Threads 100

Total Objects: 33 , Selected: 0

Variables

The following example shows how to use variables defined in the .sequence.xml as well as shared variables between the steps. It is a dummy sequence doing nothing than manipulating variables.

DemoVariables.xml

```
<sequence>

  <parameter name="sequencename" value="Variables Demo" > </parameter>
  <parameter name="warning" value="No"> </parameter>
  <parameter name="maxthreads" value="100"> </parameter>

  <variable name="inputvar" type="inputbox" value="Input box;Enter a value for the demo;">
</variable>
  <variable name="combovalue" type="combobox" value="Multiple Choice;Select the value
desired;Demo Value 1,Demo Value 2,Demo Value 3"> </variable>

  <task path="DemoVariables1.ps1" comment="Demo Variables Step 1"> </task>
  <task path="DemoVariables2.ps1" comment="Demo Variables Step 2"> </task>
  <task path="DemoVariables3.ps1" comment="Demo Variables Step 3"> </task>
```

</sequence>

DemoVariables1.ps1

```
param ($wks)
  $VariableToPass="*** $wks ***"

  return "OK", "Variable to pass: $VariableToPass", $null, $VariableToPass
```

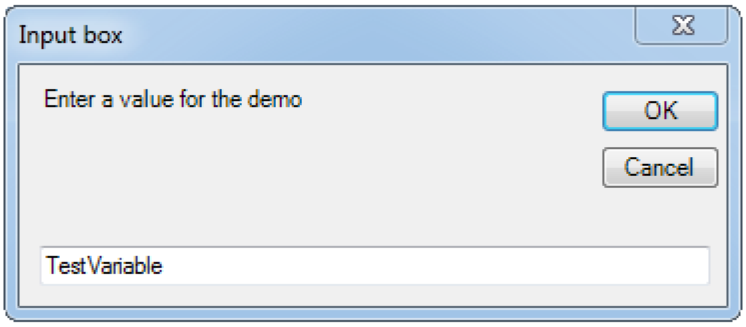
DemoVariables2.ps1

```
param ($wks, $PassedVariable)

  $NewVariable=Get-date
  return "OK", "Variable received: $PassedVariable, Next Variable to pass: $NewVariable", $null,
$NewVariable
```

DemoVariables2.ps1

```
param ($wks, $Date)

  return "OK", "Last step run: $Date, Users variables: $inputvar and $combovalue"
```

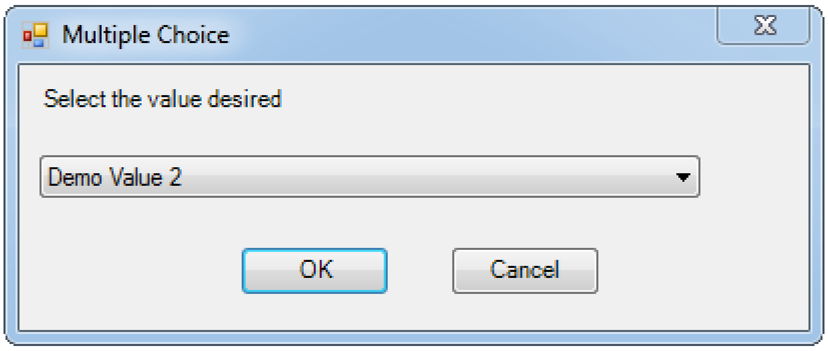
When the sequence is started, 2 variables are first queried:



Input box

Enter a value for the demo

TestVariable

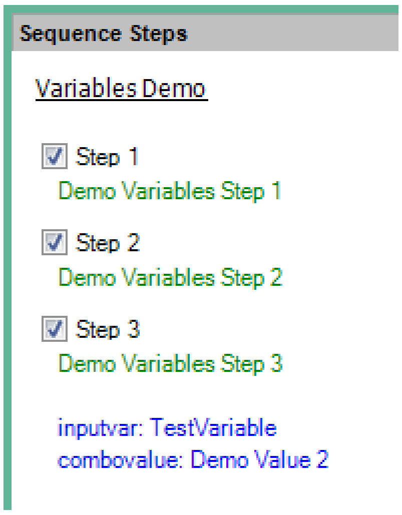


Multiple Choice

Select the value desired

Demo Value 2

Their values are displayed in the sequence panel:



Sequence Steps

Variables Demo

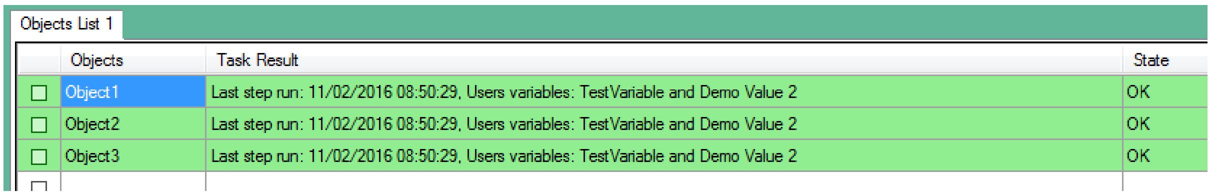
☑ Step 1
Demo Variables Step 1

☑ Step 2
Demo Variables Step 2

☑ Step 3
Demo Variables Step 3

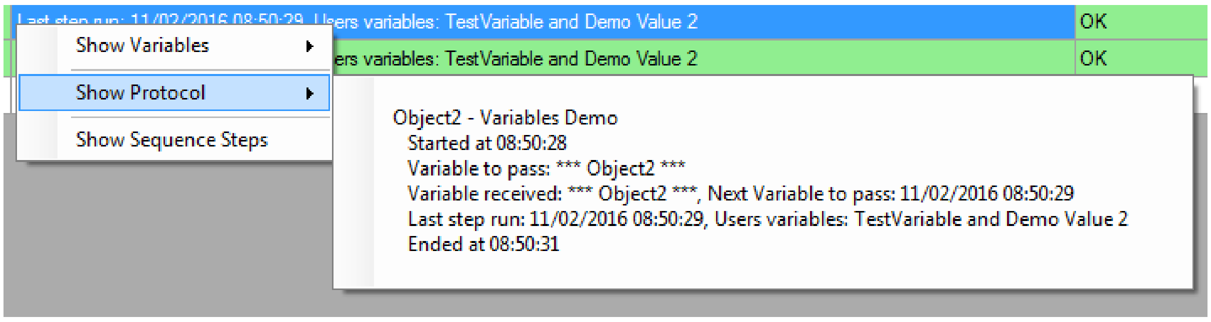
inputvar: TestVariable
combovalue: Demo Value 2

Once finished, the different variables are displayed in the state column, or in the task protocol:



Objects List 1

	Objects	Task Result	State
☐	Object1	Last step run: 11/02/2016 08:50:29, Users variables: TestVariable and Demo Value 2	OK
☐	Object2	Last step run: 11/02/2016 08:50:29, Users variables: TestVariable and Demo Value 2	OK
☐	Object3	Last step run: 11/02/2016 08:50:29, Users variables: TestVariable and Demo Value 2	OK



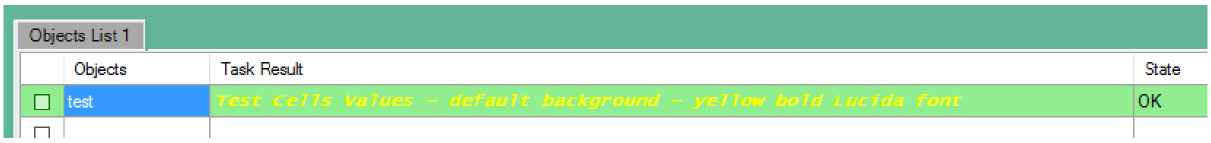
```
Object2 - Variables Demo
  Started at 08:50:28
  Variable to pass: *** Object2 ***
  Variable received: *** Object2 ***, Next Variable to pass: 11/02/2016 08:50:29
  Last step run: 11/02/2016 08:50:29, Users variables: TestVariable and Demo Value 2
  Ended at 08:50:31
```

Cell font

See below an example of a complex cell font:

param ($wks)

return "OK", "Test Cells Values - default background - yellow bold Lucida font", "$Null|'Lucida Console',8.25,3,3,0|yellow"



Objects List 1

	Objects	Task Result	State
☐	test	Test Cells Values - default background - yellow bold Lucida font	OK