Coding guidelines for WFA

OnCommand Workflow Automation 5.1

NetApp
August 30, 2024

Table of Contents

Coding guidelines for WFA

You should understand the general OnCommand Workflow Automation (WFA) coding guidelines, naming conventions, and recommendations on creating various building blocks such as filters, functions, commands, and workflows.

Guidelines for variables

You must be aware of the guidelines for PowerShell and Perl variables in OnCommand Workflow Automation (WFA) when you create a command or a data source type.

PowerShell variables

Guidelines	Example
For script input parameters: • Use Pascal case. • Do not use underscores. • Do not use abbreviations.	`$VolumeName` `$AutoDeleteOptions` `$Size`
For script internal variables: • Use Camel case. • Do not use underscores. • Do not use abbreviations.	`$newVolume` `$qtreeName` `$time`
For functions: • Use Pascal case. • Do not use underscores. • Do not use abbreviations.	`GetVolumeSize`
Variable names are not case-sensitive. However, to improve readability, you should not use different capitalization for the same name.	`$variable` is the same as `$Variable`.
Variable names should be in plain English and should be related to the functionality of the script.	Use `$name` and not `$a`.
Declare the data type for each variable, explicitly.	[string]name [int]size
Do not use special characters (! @ # & % , .) and spaces.	None

Guidelines	Example
Do not use PowerShell reserved keywords.	None
Group the input parameters by placing the mandatory parameters first followed by the optional parameters.	`param(` `[parameter(Mandatory=$true)]` `[string]$Type,` `[parameter(Mandatory=$true)]` `[string]$Ip,` `[parameter(Mandatory=$false)]` `[string]$VolumeName` `)`
Comment all input variables using *`HelpMessage`* annotation with a meaningful help message.	`[parameter(Mandatory=$false,HelpMessage="LUN to map")]` `[string]$LUNName`
Do not use “Filer” as a variable name; use “Array” instead.	None
Use *`ValidateSet`* annotation in cases where the argument gets enumerated values. This automatically translates to Enum data type for the parameter.	`[parameter(Mandatory=$false,HelpMessage="Volume state")]` `[ValidateSet("online","offline","restricted")]` `[string]$State`
Add an alias to a parameter that ends with “_Capacity” to indicate that the parameter is of capacity type.	The “Create Volume” command uses aliases as follows: `[parameter(Mandatory=$false,HelpMessage="Volume increment size in MB")]` `[Alias("AutosizeIncrementSize_Capacity")]` `[int]$AutosizeIncrementSize`

Guidelines	Example
Add an alias to a parameter that ends with “_Password” to indicate that the parameter is of password type.	``` param ([parameter(Mandatory=$false, HelpMessage="In order to create an Active Directory machine account for the CIFS server or setup CIFS service for Storage Virtual Machine, you must supply the password of a Windows account with sufficient privileges")] [Alias("Pwd_Password")] [string]$ADAdminPassword) ```

Perl variables

Guidelines	Example
For script input parameters: • Use Pascal case. • Do not use underscores. • Do not use abbreviations.	`$VolumeName` `$AutoDeleteOptions` `$Size`
Do not use abbreviations for script internal variables.	`$new_volume` `$qtree_name` `$time`
Do not use abbreviations for functions.	`get_volume_size`
Variable names are case-sensitive. To improve readability, you should not use different capitalization for the same name.	`$variable` is not the same as `$Variable`.
Variable names should be in plain English and should be related to the functionality of the script.	Use `$name` and not `$a`.
Group the input parameters by placing the mandatory parameters first, followed by the optional parameters.	None

Guidelines	Example
In GetOptions function, explicitly declare the data type of each variable for input parameters.	`GetOptions( "Name=s"=>\$Name, "Size=i"=>\$Size )`
Do not use “Filer” as a variable name; use “Array” instead.	None
Perl does not include the *ValidateSet* annotation for enumerated values. Use explicit “if” statements for cases where argument gets enumerated values.	`if (defined$SpaceGuarantee&&!($SpaceGuaranteeeq'none'||$SpaceGuaranteeeq'volume'||$SpaceGuaranteeeq'file')) { die'Illegal SpaceGuarantee argument: \''.$SpaceGuarantee.'\''; }`
All Perl WFA commands must use the “strict” pragma to discourage the use of unsafe constructs for variables, references, and subroutines.	`use strict; # the above is equivalent to use strictvars; use strictsubs; use strictrefs;`
All Perl WFA commands must use the following Perl modules: • Getopt This is used for specifying input parameters. • WFAUtil This is used for utility functions that are provided for command logging, reporting command progress, connecting to array controllers, and so on.	`use Getopt::Long; use NaServer; use WFAUtil;`

Guidelines for indentation

You must be aware of the guidelines for indentation when writing a PowerShell or Perl script for OnCommand Workflow Automation (WFA).

Guidelines	Example
A tab is equal to four empty spaces.	
Use tabs and braces to show the beginning and end of a block.	PowerShell script `if ($pair.length-ne 2) { throw "Got wrong input data" }` Perl script `if (defined $MaxDirectorySize) { # convert from MBytes to Bytes my $MaxDirectorySizeBytes = $MaxDirectorySize * 1024 * 1024; }`
Add blank lines between sets of operations or chunks of code.	`$options=$option.trim(); $pair=$option.split(" "); Get-WFAlogger -Info -messages $("split options: "+ $Pair)`

Guidelines for comments

You must be aware of the guidelines for PowerShell and Perl comments in your scripts for OnCommand Workflow Automation (WFA).

PowerShell comments

Guidelines	Example
Use the # character for a single line comment.	`# Single line comment` `$options=$option.trim();`
Use the # character for an end of line comment.	`$options=$option.trim(); # End of line` `comment`
Use the <# and #> characters for a block comment.	`<#` `This is` `a` `block comment` `#>` `$options=$option.trim();`

Perl comments

Guidelines	Example
Use the # character for single line comment.	`# convert from MBytes to Bytes` `my $MaxDirectorySizeBytes =` `$MaxDirectorySize *` `1024 * 1024;`
Use the # character for end of line comment.	`my $MaxDirectorySizeBytes =` `$MaxDirect` `orySiZe * 1024 * 1024; # convert` `to Bytes`

Guidelines	Example
Use the # character in every line with an empty # at the beginning and end to create a comment border for multi-line comments.	``` # # This is a multi-line comment. Perl 5, unlike # Powershell, does not have direct support for # multi-line comments. Please use a '#'in every line # with an empty '#' at the beginning and end to create # a comment border # ```
Do not include commented and dead code in WFA commands. However, for testing purposes, you can use the Plain Old Documentation (POD) mechanism to comment out the code.	``` =begin comment # Set deduplication if(defined $Deduplication && $Deduplication eq "enabled") { $wfaUtil- >sendLog("Enabling Deduplication"); } =end comment =cut ```

Guidelines for logging

You must be aware of the guidelines for logging when writing a PowerShell or Perl script for OnCommand Workflow Automation (WFA).

PowerShell logging

Guidelines	Example
Use the Get-WFALogger cmdlet for logging.	``` Get-WFALogger -Info -message “Creating volume” ```

Guidelines	Example
Log every action that requires interaction with internal packages such as Data ONTAP, VMware, and PowerCLI.All the log messages are available in Execution Logs in the execution status history of workflows.	None
Log every relevant argument that is passed to internal packages.	None
Use appropriate log levels when using the Get-WFALogger cmdlet, depending on the usage context. -Info, -Error, -Warn, and -Debug are the various available log levels. If a log level is not specified, then the default log level is Debug.	None

Perl logging

Guidelines	Example
Use the WFAUtil sendLog for logging.	`my wfa_util = WFAUtil->new();` `eval {` `$wfa_util->sendLog('INFO',` `"Connecting to the` `cluster: $DestinationCluster");` `}`
Log every action that requires interaction with anything external to the command such as Data ONTAP, VMware, and WFA. All the log messages that you create using the WFAUtil sendLog routine are stored in the WFA database. These log messages are available for the executed workflow and command.	None
Log every relevant argument passed to the routine that was called.	None
Use appropriate log levels.-Info, -Error, -Warn, and -Debug are the various available log levels.	None

Guidelines	Example
When logging at the -Info level, be precise and concise. Do not specify implementation details such as class name and function name in log messages. Describe the exact step or the exact error in plain English.	The following code snippet shows an example of a good message and a bad message: `$wfa_util->sendLog('WARN', "Removing volume: '.$VolumeName); # Good Message` `$wfa_util->sendLog('WARN', 'Invoking volume-destroy ZAPI: '.$VolumeName); # Bad message`

Guidelines for error handling

You must be aware of the guidelines for error handling when writing a PowerShell or Perl script for OnCommand Workflow Automation (WFA).

PowerShell error handling

Guidelines	Example
Common parameters added to cmdlets by PowerShell runtime include error handling parameters such as ErrorAction and WarningAction: • The ErrorAction parameter determines how a cmdlet should react to a non-terminating error from the command. • The WarningAction parameter determines how a cmdlet should react to a warning from the command. • Stop, SilentlyContinue, Inquire, and Continue are the valid values for the ErrorAction and WarningAction parameters. For more information, you can use the `Get-Help about_CommonParameters` command in PowerShell CLI.	ErrorAction: the following example shows how to handle a non-terminating error as a terminating error: `New-NcIgroup-Name $IgroupName-Protocol $Protocol-Type$OSType-ErrorActionstop` WarningAction `New-VM-Name $VMName-VM $SourceVM-DataStore$DataStoreName-VMHost$VMHost-WarningActionSilentlyContinue`

Guidelines	Example
Use the general “try/catch” statement if the type of the incoming exception is unknown.	`try` `{` `"In Try/catch block"` `}` `catch` `{` `"Got exception"` `}`
Use the specific “try/catch” statement if the type of the incoming exception is known.	`try` `{` `"In Try/catch block"` `}` `catch[System.Net.WebExceptional],` `[System.IO.` `IOException]` `{` `"Got exception"` `}`
Use the “finally” statement to release resources.	`try` `{` `"In Try/catch block"` `}` `catch` `{` `"Got exception"` `}` `finally` `{` `"Release resources"` `}`

Guidelines	Example
Use PowerShell automatic variables to access information about exceptions.	`try` `{` `Get-WFALogger -Info -message` `$("Creating` `Ipspace: " + $Ipspace)` `New-NaNetIpspace-Name $Ipspace` `}` `catch` `{` `Throw "Failed to create Ipspace.` `Message:` `" + $_.Exception.Message;` `}`

Perl error handling

Guidelines	Example
Perl does not include native language support for try/catch blocks. Use eval blocks for checking and handling errors. Keep eval blocks as small as possible.	``` eval { $wfa_util->sendLog('INFO', "Quiescing the relationship : $DestinationCluster://$Destination Vserver /$DestinationVolume"); $server->snapmirror_quiesce('destination-vserver' => $DestinationVserver, 'destination-volume' => $DestinationVolume); $wfa_util->sendLog('INFO', 'Quiesce operation started successfully.'); }; $wfa_util->checkEvalFailure("Failed to quiesce the SnapMirror relationship $DestinationCluster://$Destination Vserver /$DestinationVolume", $@); ```

General PowerShell and Perl conventions for WFA

You must understand certain PowerShell and Perl conventions that are used in WFA to create scripts that are consistent with existing scripts.

- Use variables that help to clarify what you want the script to do.
- Write readable code that can be understood without comments.
- Keep the scripts and commands as simple as possible.
- For PowerShell scripts:
  - Use cmdlets whenever possible.
  - Invoke .NET code when there is no cmdlet available.
- For Perl scripts:

- Always end “die” statements with newline characters.

  In the absence of a newline character, the script line number is printed, which is not useful for debugging Perl commands executed by WFA.

- In the “GetOpt” module, make the string arguments to a command mandatory.

Perl modules bundled with Windows

Some Perl modules are bundled with the Windows Active state Perl distribution for OnCommand Workflow Automation (WFA). You can use these Perl modules in your Perl code for writing commands, only if they are bundled with Windows.

The following table lists the Perl database modules that are bundled with Windows for WFA.

Database module	Description
DBD::mysql	Perl5 database interface driver that enables you to connect to the MySQL database.
Try::Tiny	Minimizes common mistakes with evaluation blocks.
XML::LibXML	Interface to libxml2 that provides XML and HTML parsers with DOM, SAX, and XMLReader interfaces.
DBD::Cassandra	Perl5 database interface driver for Cassandra that uses the CQL3 query language.

Considerations for adding custom PowerShell and Perl modules

You must be aware of certain considerations before adding custom PowerShell and Perl modules to OnCommand Workflow Automation (WFA). Custom PowerShell and Perl modules enable you to use custom commands for creating workflows.

- During the execution of WFA commands, all custom PowerShell modules are added to the WFA install directory `/Posh/modules` are automatically imported.
- All custom Perl modules added to the `WFA/perl` directory are included in the *@INC* library.
- Custom PowerShell and Perl modules are not backed up as part of the WFA backup operation.
- Custom PowerShell and Perl modules are not restored as part of the WFA restore operation.

You must manually back up custom PowerShell and Perl modules in order to copy them to a new WFA installation.

The folder name in modules' directory must be same as that of the module name.

WFA cmdlets and functions

OnCommand Workflow Automation (WFA) provides several PowerShell cmdlets as well as PowerShell and Perl functions that you can use in your WFA commands.

You can view all the PowerShell cmdlets and functions provided by the WFA server using the following PowerShell commands:

- `Get-Command -Module WFAWrapper`
- `Get-Command -Module WFA`

You can view all the Perl functions provided by the WFA server in the `WFAUtil.pm` module. The help sections, WFA PowerShell cmdlets help and WFA Perl methods help, of the WFA Help module Support Links enables access to all the PowerShell cmdlets and functions and the Perl functions.

PowerShell and Perl WFA modules

You must be aware of the PowerShell or Perl modules for OnCommand Workflow Automation (WFA) to write scripts for your workflows.

PowerShell modules

Guidelines	Example
Use the Data ONTAP PS Toolkit to invoke APIs whenever the toolkit is available.	The `Add VLAN` command uses the toolkit as follows: `Add-NaNetVlan-Interface $Interface-Vlans$VlanID`
If there are no cmdlets available in the Data ONTAP PS Toolkit, use the `Invoke-SSH` command to invoke the CLI on Data ONTAP.	`Invoke-NaSsh-Name $ArrayName-Command "ifconfig -a"-Credential $Credentials`

Perl modules

The NaServer module is used in WFA commands. The NaServer module allows the invocation of Data ONTAP APIs, which are used in active management of Data ONTAP systems.

Guidelines	Example
Use the NaServer module to invoke APIs whenever the NetApp Manageability SDK is available.	The following example shows how the NaServer module is used for a resume SnapMirror operation: `eval {` `$wfa_util->sendLog('INFO', "Connecting to the cluster: $DestinationCluster" );` `my $server = $wfa_util->connect($DestinationClusterIp, $DestinationVserver);` `my $sm_info = $server->snapmirror_get( 'destination-vserver' => $DestinationVserver, 'destination-volume' => $DestinationVolume );` `my $sm_state = $sm_info->{'attributes'}->{'snapmirror-info'}->{'mirror-state'};` `my $sm_status = $sm_info->{'attributes'}->{'snapmirror-info'}->{'relationship-status'};` `$wfa_util->sendLog('INFO', "SnapMirror relationship is $sm_state ($sm_status)");` `if ($sm_status ne 'quiesced') {` `$wfa_util->sendLog('INFO', 'The status needs to be quiesced to resume transfer.');` `} else {` `my $result = $server->snapmirror_resume( 'destination-vserver' => $DestinationVserver, 'destination-volume' => $DestinationVolume );` `$wfa_util->sendLog('INFO', "Result of resume: $result");`

Guidelines	Example
If a Data ONTAP API is not available, invoke the Data ONTAP CLI using the executeSystemCli utility method. executeSystemCli is not supported and is currently available only for Data ONTAP operating in 7-Mode.	None

Considerations while converting PowerShell commands to Perl

You must be aware of certain important considerations when you convert PowerShell commands to Perl because PowerShell and Perl have different capabilities.

Command input types

OnCommand Workflow Automation (WFA) allows workflow designers to use arrays and hash as input for the command when defining a command. These input types cannot be used when the command is defined using Perl. If you want a Perl command to accept array and hash inputs, you can define the input as a string in the designer. The command definition can then parse the input, which is passed to create an array or hash as required. The description for the input describes the format in which the input is expected.

```
my @input_as_array  = split(',', $InputString); #Parse the input string of
format val1,val2 into an array

my %input_as_hash = split /[;=]/, $InputString; #Parse the input string of
format key1=val1;key2=val2 into a hash.
```

PowerShell statement

The following examples show how an array input can be passed into PowerShell and Perl. The examples describe the input CronMonth, which specifies the month when the cron job is scheduled to run. The valid values are integers -1 to 11. A value of -1 indicates that the schedule executes every month. Any other value denotes a specific month, with 0 being January and 11 being December.

```
[parameter(Mandatory=$false, HelpMessage="Months in which the schedule
executes. This is a comma separated list of values from 0 through 11.
Value -1 means all months.")]
  [ValidateRange(-1, 11)]
  [array]$CronMonths,
```

Perl statement

```
GetOptions(
    "Cluster=s"         => \$Cluster,
    "ScheduleName=s"    => \$ScheduleName,
    "Type=s"            => \$Type,
    "CronMonths=s"      => \$CronMonths,
) or die 'Illegal command parameters\n';

sub get_cron_months {
    return get_cron_input_hash('CronMonths', $CronMonths, 'cron-month',
-1,
        11);
}

sub get_cron_input_hash {
    my $input_name   = shift;
    my $input_value  = shift;
    my $zapi_element = shift;
    my $low          = shift;
    my $high         = shift;
    my $exclude      = shift;

    if (!defined $input_value) {
        return undef;
    }

    my @values = split(',', $input_value);

    foreach my $val (@values) {
        if ($val !~ /^[+-]?\d+$/) {
            die
                "Invalid value '$input_value' for $input_name: $val must
be an integer.\n";
        }
        if ($val < $low || $val > $high) {
            die
                "Invalid value '$input_value' for $input_name: $val must
be from $low to $high.\n";
        }
        if (defined $exclude && $val == $exclude) {
            die
                "Invalid value '$input_value' for $input_name: $val is not
valid.\n";
        }
    }
    # do something
}
```

Command definition

A one-line expression in PowerShell using a pipe operator might have to be expanded into multiple blocks of statements in Perl in order to achieve the same functionality. An example from one of the wait commands is shown in the following table.

PowerShell statement	Perl statement
`# Get the latest job which moves the specified volume to the specified aggregate.` `$job = Get-NcJob -Query $query \| where {$_.JobDescription -eq "Split" + $VolumeCloneName} | Select-Object -First 1`	`my $result = $server->job_get_iter(` `'query' => {'job-type' => 'VOL_CLONE_SPLIT'},` `'desired-attributes' => {` `'job-type' => '',` `'job-description' => '',` `'job-progress' => '',` `'job-state' => ''` `}` `);` `my @jobarray;` `for my $job (@{ $result->{'attributes-list'}})` `{` `my $description = $job->{'job-description'};` `if($description =~ /$VolumeCloneName/)` `{` `push(@jobarray, $job)` `}` `}`

Guidelines for WFA building blocks

You must be aware of the guidelines for using Workflow Automation building blocks.

Guidelines for SQL in WFA

You must be aware of the guidelines for using SQL in OnCommand Workflow Automation (WFA) to write SQL queries for WFA.

SQL is used in the following places in WFA:

- SQL queries to populate user inputs for selection
- SQL queries for creating filters to filter objects of a specific dictionary entry type

- Static data in tables in the playground database
- A custom data source type of SQL type where the data has to be extracted from an external data source such as a custom configuration management database (CMDB).
- SQL queries for reservation and verification scripts

Guidelines	Example
SQL reserved keywords must be in uppercase characters.	`SELECT` `    vserver.name` `FROM` `    cm_storage.vserver vserver`
Table and column names must be in lowercase characters.	Table: aggregate Column: used_space_mb
Separate words with an underscore (_) character. Spaces are not allowed.	array_performance
Table name is defined in singular. A table is a collection of one or more entries.	“function”, not “functions”
Use table aliases with meaningful names in SELECT queries.	`SELECT` `    vserver.name` `FROM` `    cm_storage.cluster cluster,` `    cm_storage.vserver vserver` `WHERE` `    vserver.cluster_id =` `cluster.id` `    AND cluster.name =` `'${ClusterName}'` `    AND vserver.type = 'cluster'` `ORDER BY` `    vserver.name ASC`

Guidelines	Example
If you have to refer to a filter input parameter or user input parameter in a filter query or user query, use the syntax as '${inputVariableName}.You can also use the syntax to refer to a command definition parameter in reservation scripts and verification scripts.	`SELECT` `    volume.name AS Name,` `    aggregate.name as Aggregate,` `    volume.size_mb AS 'Total Size (MB)',` `    voulme.used_size_mb AS 'Used Size (MB)',` `    volume.space_guarantee AS 'Space Guarantee'` `FROM` `    cm_storage.cluster,` `    cm_storage.aggregate,` `    cm_storage.vserver,` `    cm_storage.volume` `WHERE` `    cluster.id = vserver.cluster_id` `    AND aggregate.id = volume.aggregate_id` `    AND vserver.id = voulme.vserver_id` `    AND vserver.name = '${VserverName}'` `    AND cluster.name = '${ClusterName}'` `ORDER BY` `    volume.name ASC`
Use comments for complex queries. Some of the supported comment styles in queries are as follows: • “--” until the end of the line A space is mandatory after the second hyphen in this comment style. • From a “#” character until the end of the line • From a “/**” to the following "**`/`"sequence	`/*` `multi-line` `comment` `*/` `--line comment` `SELECT` `    ip as ip, # comment till end of this line` `    NAME as name` `FROM --end of line comment` `    storage.array`

Guidelines for WFA functions

You can create functions to encapsulate commonly used and more complex logic in a named function, and then reuse the function as command parameter values or filter parameters values in OnCommand Workflow Automation (WFA).

Guidelines	Example
Use Camel case for a function name.	calculateVolumeSize
Variable names should be in plain English and related to the functionality of the function.	splitByDelimiter
Do not use abbreviations.	calculateVolumeSize, *not* calcVolSize
Functions are defined using MVFLEX Expression Language (MVEL).	None
The function definition should be specified according to the official Java Programming Language guidelines.	None

Guidelines for WFA dictionary entries

You must be aware of the guidelines for creating dictionary entries in OnCommand Workflow Automation (WFA).

Guidelines	Example
Dictionary entry names must contain only alphanumeric characters and underscores.	Cluster_License Switch_23
Dictionary entry names must start with an uppercase character. Begin every word in the name with an uppercase character and separate each word with an underscore (_).	Volume Array_License
Dictionary entry attribute names should not include the name of the dictionary entry.	None
Attributes and references in a dictionary entry must be in lowercase characters.	aggregate, size_mb
Separate words with an underscore. Spaces are not allowed.	resource_pool

Guidelines	Example
Dictionary entries cannot include references that are from a different scheme. When a dictionary entry requires cross-reference to an object in a different scheme, ensure that all the natural keys of the object being referred to are present in the dictionary entry.	Array_Performance dictionary entry requires all the natural keys of the Array dictionary entry as direct attributes in it.
Use appropriate data types for attributes.	None
Use Long data type for size or space-related attributes.	size_mb and available_size_mb in storage.Volume dictionary entry
Use Enum when an attribute has a fixed set of values.	raid_type in storage.Volume dictionary entry
Set “To be Cached” as true for an attribute or reference when a data source provides value for that attribute or reference.For Active IQ Unified Manager data source, add cacheable attributes if the data source can provide the value to it.	None
Set “Can be Null” as true if the data source providing the value for this attribute or reference can return NULL.	None
Provide a meaningful description to each attribute and reference.The description is displayed in command details when designing a workflow.	None
Do not use “id” as the name of an attribute in dictionary entries.It is reserved for internal WFA usage.	None

Related information

References to learning material

Guidelines for commands

You must be aware of the guidelines for creating commands in OnCommand Workflow Automation (WFA).

Guidelines	Example
Use an easily identifiable name for commands.	`Create Qtree`
Use spaces to delimit words and each word must start with an uppercase character.	`Create Volume`

Guidelines	Example
Provide a description to explain the functionality of the command, including the expected outcome of the optional parameters.	None
By default, the timeout for standard commands is 600 seconds. The default timeout is set while creating the command. Change the default value only if the command might take a longer time to complete.	`Create Volume` command
In case of long-running operations, create two commands—one to invoke the long-running operation and another to report the progress of the operation periodically. The first command should be a `Standard Execution` command type and the second should be `Wait for Condition` command type.	`Create VSM` and `Wait for VSM` commands
Prefix the `Wait for condition` command names with “Wait” for easy identification.	`Wait for CM Volume Move`
Use an appropriate waiting interval for the “Wait for condition” commands. The specified value governs the interval at which the polling command gets executed to check if the long-running operation is complete.	60s sampling interval for the `Wait for VSM` command
For the `Wait for condition` commands, use an appropriate timeout based on the expected time for the long-running operation to complete. The expected time might be considerably longer if the operation involves data transfer over a network.	A VSM baseline transfer can take many days to complete. Therefore, the specified timeout is 6 days.

String representation

The string representation for a command displays the details of a command in a workflow design during planning and execution. Only the command parameters can be used in the string representation for a command.

Guidelines	Example
Avoid using attributes that do not have any value. An attribute without a value is displayed as NA.	VolName 10.68.66.212[NA]aggr1/testVol7
Separate different entries in the string representation using the following delimiters: [ ] , / :	*`ArrayName[ArrayIp]`*

Guidelines	Example
Provide meaningful labels to every value in string representation.	`Volume name=VolumeName`

Command definition language

Commands can be written using the following supported scripting languages:

- PowerShell
- Perl

Command parameter definition

The command parameters are described by Name, Description, Type, a default value for the parameter, and whether the parameter is mandatory. The parameter type can be String, Boolean, Integer, Long, Double, Enum, DateTime, Capacity, Array, Hashtable, Password, or an XmlDocument. While the values for most of the types are intuitive, the values for Array and Hashtable should be in a particular format as described in the following table:

Guidelines	Example
Ensure that the value for an Array input type is a list of values, separated by comma.	`[parameter(Mandatory=$false, HelpMessage="Months in which the schedule executes.")] [array]$CronMonths` Input is passed as following: 0,3,6,9
Ensure that the value for a Hashtable input type is a list of key=value pairs, separated by semicolon.	`[parameter(Mandatory=$false, HelpMessage="Volume names and size (in MB)")] [hashtable]$VolumeNamesAndSize` Input is passed as following: Volume1=100;Volume2=250;Volume3=50

Guidelines for workflows

You must be aware of the guidelines for creating or modifying a predefined workflow for OnCommand Workflow Automation (WFA).

General guidelines

Guidelines	Example
Name the workflow such that it reflects the operation that is executed by the storage operator.	`Create a CIFS Share`
For workflow names, capitalize the initial letter of the first word and every word that is an object. Capitalize letters for abbreviations and acronyms.	Volume Qtree Create a Clustered Data ONTAP Qtree CIFS Share
For workflow descriptions, include all of the important steps of the workflow, including any prerequisites, result of the workflow, or conditional aspects of execution.	See the description of the sample workflow `Create VMware NFS Datastore on Clustered Data ONTAP Storage`, which includes the prerequisites.
Set “Ready For Production” to `true` only when the workflow is ready for production and can be displayed in the portal page.	None
By default, set “Consider reserved elements” to true. When previewing a workflow for execution, the WFA planner considers all of the objects that are reserved along with the existing objects in the cache database. Effects of other scheduled workflows or workflows executing in parallel are considered when planning a specific workflow if this option is set to `true`.	• Scenario 1 Workflow 1 creates a volume, and is scheduled to execute one week later. Workflow 2 creates qtrees or LUNs in volumes that are searched for, and if workflow 2 is executed within a day or so, you should turn off “Consider reserved elements” for workflow 2 to prevent it from considering the volume that is to be created in a week. • Scenario 2 Workflow 1 uses the `Create Volume` command. If there is a scheduled workflow 2 that consumes 100 GB from an aggregate, then workflow 1 must consider the requirements for workflow 2 during planning.

Guidelines	Example
By default, “Enable element existence validation” is set to `true`.	• Scenario 1 If you create a workflow that first removes a volume by name using the command `Remove Volume` only if the volume exists, and the re-creates it using another command such as `Create Volume` or `Clone Volume`, then the workflow should not use this flag. The effect of removing the volume will not be available to the `Create volume` command, thereby causing the workflow to fail. • Scenario 2 The `Create Volume` command is used in a workflow with a specific name as “vol198”. If this option is set to true, WFA planner checks during planning to see if a volume by that name exists in the given array. If the volume exists, the workflow fails during planning.
When the same command is selected more than once in a workflow, provide appropriate display names for the command instances.	The “Create, map, and protect LUNs with SnapVault” sample workflow uses the `Create Volume` command twice. However, it uses the display names as `Create Primary Volume` and `Create Secondary Volume` appropriately for the primary volume and the mirrored destination volume.

User inputs

Guidelines	Example
Names: • Start the name with the “$” character. • Use an uppercase letter at the beginning of each word. • Use uppercase letters for all terms and abbreviations. • Do not use underscores.	`$Array` `$VolumeName`

Guidelines	Example
Display names: • Use an uppercase letter at the beginning of each word. • Separate words with spaces. • If inputs have specific units, specify the unit in brackets in the display name directly.	`Volume Name` `Volume Size (MB)`
Descriptions: • Provide a meaningful description for each user input. • Provide examples when required. You should do this especially when the user input is expected to be in a specific format. The user input descriptions are displayed as tooltips for the user inputs during workflow execution.	Initiators to be added to an “iGroup”. For example, IQN or WWPN of the initiator.
Type: Select Enum as the type if you want to restrict the input to a specific set of values.	Protocol: “iscsi”, “fcp”, “mixed”
Type: Select Query as the type when the user can select from values available in the WFA cache.	$Array: QUERY type with query as follows: `SELECT` `  ip, name` `FROM` `  storage.array`
Type: Mark the user input as locked when the user input should be restricted to the values that are obtained from a query or should be restricted to only the supported Enum types.	$Array: Locked Query type: Only arrays in the cache can be selected.$Protocol: Locked Enum type with valid values as iscsi, fcp, mixed. No other value than the valid value is supported.
Type: Query TypeAdd additional columns as return values in the query when it helps the storage operator to make the right choice of user input.	$Aggregate: Provide name, total size, available size so that the operator knows the attributes before selecting the aggregate.

Guidelines	Example
Type: Query TypeSQL query for user inputs can refer to any other user inputs preceding it. This can be used to limit the results from a query based on other user inputs such as vFiler units of an array, volumes of an aggregate, LUNs in a storage virtual machine (SVM).	In the sample workflow `Create a Clustered Data ONTAP Volume`, the query for VserverName is as follows: `SELECT` `  vserver.name` `FROM` `  cm_storage.cluster cluster,` `  cm_storage.vserver vserver` `WHERE` `  vserver.cluster_id = cluster.id` `  AND cluster.name = '${ClusterName}'` `  AND vserver.type = 'cluster'` `ORDER BY` `  vserver.name ASC` The query refers to ${ClusterName}, where $ClusterName is the name of the user input preceding the $VserverName user input.
Type: Use Boolean type with values as “true, false” for user inputs that are Boolean in nature. This helps in writing internal expressions in the workflow design using the user input directly. For example, $UserInputName rather than $UserInputName == ''`Yes'.	`$CreateCIFSShare`: Boolean type with valid values as “true” or “false”
Type:For string and number type, use regular expressions in the values column when you want to validate the value with specific formats. Use regular expressions for IP address and network mask inputs.	Location-specific user input can be expressed as “[A-Z][A-Z]\-0[1-9]”. This user input accepts values such as “US-01”, “NB-02”, but not “nb-00”.
Type: For number type, a range-based validation can be specified in the values column.	For Number of LUNs to be created, the entry in the Values column is 1-20.
Group: Group related user inputs into appropriate buckets and name the group.	“Storage Details” for all storage-related user inputs. “Datastore Details” for all VMware-related user inputs.

Guidelines	Example
Mandatory: If the value of any user input is necessary for the workflow to execute, mark the user input as mandatory. This ensures that the user input screen mandatorily accepts that input from the user.	“$VolumeName” in the “Create NFS Volume” workflow.
Default value: If a user input has a default value that can work for most of the workflow executions, provide the values. This helps in allowing the user to provide fewer inputs during execution, if the default serves the purpose.	None

Constants, variables, and returns parameters

Guidelines	Example
Constants: Define constants when using a common value for defining parameters to multiple commands.	*AGGREGATE_OVERCOMMITMENT_THRESHOLD* in the `Create, map, and protect LUNs with SnapVault sample workflow`.
Constants:Names • Use an uppercase letter at the beginning of each word. • Use uppercase letters for all terms and abbreviations. • Do not use underscores. • Use uppercase letters for all letters of constant names.	*AGGREGATE_USED_SPACE_THRESHOLD* *ActualVolumeSizeInMB*
Variables: Provide a name to an object defined in one of the command parameter boxes. Variables are automatically generated names and can be changed.	None
Variables: Names Use lowercase characters for variable names.	volume1 cifs_share
Return parameters: Use return parameters when the workflow planning and execution should return some calculated or selected values during planning. The values are made available in the preview mode when the workflow is executed from a web service as well.	Aggregate: If the aggregate is selected using the resource selection logic, then the actual selected aggregate can be defined as a return parameter.

Guidelines for creating validation scripts for remote system types

You must be aware of the guidelines for creating validation scripts that are used to test

the remote system types that you define in OnCommand Workflow Automation (WFA).

- The Perl script that you create must be similar to the sample script provided in the Validation Script window.
- The output of your validation script must be similar to that of the sample script.

Sample validation script

```
# Check connectivity.
# Return 1 on success.
# Return 0 on failure and set $message
sub checkCredentials {
my ($host, $user, $passwd, $protocol, $port, $timeout) = @_;
#
# Please add the code to check connectivity to $host using $protocol here.
#
return 1;
}
```

Guidelines for creating data source types

You must be aware of the guidelines for creating data source types that are used to define custom data sources for OnCommand Workflow Automation (WFA).

You can define a data source type by using one of the following methods:

- SQL: You can use the WFA SQL guidelines to define select queries from data sources based on an external database.
- SCRIPT: You can write a PowerShell script that provides the data for a specific scheme of dictionary entries.

The guidelines for creating data source types are as follows:

- You should use PowerShell language must be used to create script.
- The PowerShell script should provide the output for each dictionary entry in its current working directory.
- The data files should be named `dictionary_entry.csv`, where the name of the dictionary entry should be in lower-case characters.

  The predefined data source type that collects information from Performance Advisor uses a SCRIPT-based data source type. The output files are named `array_performance.csv` and `aggregate_performance.csv`.

- The `.csv` file should include the content in the exact order as that of the dictionary entry attributes.

  A dictionary entry includes attributes in the following order: array_ip, date, day, hour, cpu_busy, total_ops_per_sec, disk_throughput_per_sec.

  The PowerShell script adds data to the `.csv` file in the same order.

```
$values = get-Array-CounterValueString ([REF]$data)
Add-Content $arrayFile ([byte[]][char[]] "\N
t$arrayIP't$date't$day't$hour't$values'n")
```

- You should use Encoding to ensure that the data output from the script is loaded into the WFA cache accurately.
- You should use \N while entering a Null value in the `.csv` file.

Copyright information

Trademark information