



PERL-based development

SnapCenter Software

Nirupama Sriram, Soumik Das
June 16, 2021

Table of Contents

- PERL-based development 1
 - General plug-in handling 1

PERL-based development

You must follow certain conventions while developing the plug-in using PERL.

- Contents must be readable
- Must implement mandatory operations setENV, quiesce, and unquiesce
- Must use a specific syntax to pass results back to the agent
- The contents should be saved as <PLUGIN_NAME>.pm file

Available operations are

- setENV
- version
- quiesce
- unquiesce
- clone_pre, clone_post
- restore_pre, restore
- cleanup

General plug-in handling

Using results object

Every custom plug-in operation must define the results object. This object sends messages, exit code, stdout, and stderr back to the host agent.

Results object:

```
my $result = {
```

```
    exit_code => 0,  
    stdout => "",  
    stderr => "",  
};
```

Returning the results object:

```
return $result;
```

Preserving data consistency

It is possible to preserve data between operations (except cleanup) as part of same workflow execution. This is

done using key-value pairs. The key-value pairs of data are set as part of result object and are retained and available in the subsequent operations of same workflow.

The following code sample sets the data to be preserved:

```
my $result = {
    exit_code => 0,
    stdout => "",
    stderr => "",
};
$result->{env}->{'key1'} = 'value1';
$result->{env}->{'key2'} = 'value2';
...
return $result
```

The above code sets two key-value pairs, which are available as input in the subsequent operation. The two key-value pairs are accessible using the following code:

```
sub setENV {
    my ($self, $config) = @_ ;
    my $first_value = $config->{'key1'} ;
    my $second_value = $config->{'key2'} ;
    ...
}
```

=== Logging error messages

Each operation can send messages back to the host agent, which displays and stores the content. A message contains the message level, a timestamp, and a message text. Multiline messages are supported.

```
Load the SnapCreator::Event Class:
my $msgObj = new SnapCreator::Event();
my @message_a = ();
```

Use the msgObj to capture a message by using the collect method.


```
$msgObj->collect(\@message_a, INFO, "My INFO Message");
$msgObj->collect(\@message_a, WARN, "My WARN Message");
$msgObj->collect(\@message_a, ERROR, "My ERROR Message");
$msgObj->collect(\@message_a, DEBUG, "My DEBUG Message");
$msgObj->collect(\@message_a, TRACE, "My TRACE Message");
```


Apply messages to the results object:

```
$result->{message} = \@message_a;
```

Using plug-in stubs

Custom plug-ins must expose plug-in stubs. These are methods that the SnapCenter Server calls, based on a workflow.

Plug-in Stub	Optional/Required	Purpose
setENV	required	<p>This stub sets the environment and the configuration object.</p> <p>Any environment parsing or handling should be done here. Each time a stub is called, the setENV stub is called just before. It is only required for PERL-style plug-ins.</p>
Version	Optional	<p>This stub is used to get application version.</p>
Discover	Optional	<p>This stub is used to discover application objects like instance or database hosted on the agent or host.</p> <p>The plug-in is expected to return discovered application objects in specific format as part of the response. This stub is only used in case the application is integrated with SnapDrive for Unix.</p> <div><p>Linux file system (Linux Flavors) is supported. AIX/Solaris (Unix Flavors) are not supported.</p></div>

Plug-in Stub	Optional/Required	Purpose
discovery_complete	Optional	<p>This stub is used to discover application objects like instance or database hosted on the agent or host.</p> <p>The plug-in is expected to return discovered application objects in specific format as part of the response. This stub is only used in case the application is integrated with SnapDrive for Unix.</p> <div>  <p>Linux file system (Linux flavors) is supported. AIX and Solaris (Unix flavors) are not supported.</p> </div>
Quiesce	required	<p>This stub is responsible for performing a quiesce, which means placing application into a state where you can create a Snapshot copy. This is called before Snapshot copy operation. The metadata of application to be retained should be set as part of response, which shall be returned during subsequent clone or restore operations on corresponding storage Snapshot copy in the form of configuration parameters.</p>
Unquiesce	required	<p>This stub is responsible for performing a unquiesce, which means placing application into a normal state. This is called after you create a Snapshot copy.</p>
clone_pre	optional	<p>This stub is responsible for performing preclone tasks. This assumes you are using the built-in SnapCenter Server cloning interface and is triggered when performing clone operation.</p>

Plug-in Stub	Optional/Required	Purpose
clone_post	optional	This stub is responsible for performing post clone tasks. This assumes you are using the built-in SnapCenter Server cloning interface and is triggered only when performing clone operation.
restore_pre	optional	This stub is responsible for performing prerestore tasks. This assumes you are using the built-in SnapCenter Server restore interface and is triggered while performing restore operation.
Restore	optional	This stub is responsible for performing application restore tasks. This assumes you are using the built-in SnapCenter Server restore interface and is only triggered when performing restore operation.
Cleanup	optional	This stub is responsible for performing cleanup after backup, restore, or clone operations. Cleanup can be during normal workflow execution or in the event of a workflow failure. You can infer the workflow name under which cleanup is called by referring to configuration parameter ACTION, which can be backup, cloneVolAndLun, or fileOrVolRestore. The configuration parameter ERROR_MESSAGE indicates if there was any error while executing the workflow. If ERROR_MESSAGE is defined and NOT NULL, then cleanup is called during workflow failure execution.
app_version	Optional	This stub is used by SnapCenter to get application version detail managed by the plug-in.

Plug-in package information

Every plug-in must have following information:

```

package MOCK;
our @ISA = qw(SnapCreator::Mod);
=head1 NAME
MOCK - class which represents a MOCK module.
=cut
=head1 DESCRIPTION
MOCK implements methods which only log requests.
=cut
use strict;
use warnings;
use diagnostics;
use SnapCreator::Util::Generic qw ( trim isEmpty );
use SnapCreator::Util::OS qw ( isWindows isUnix getUid
createTmpFile );
use SnapCreator::Event qw ( INFO ERROR WARN DEBUG COMMENT ASUP
CMD DUMP );
my $msgObj = new SnapCreator::Event();
my %config_h = ();

```

Operations

You can code various operations like setENV, Version, Quiesce, and Unquiesce, which are supported by the custom plug-ins.

setENV operation

The setENV operation is required for plug-ins created using PERL. You can set the ENV and can easily access plug-in parameters.

```

sub setENV {
    my ($self, $obj) = @_;
    %config_h = %{$obj};
    my $result = {
        exit_code => 0,
        stdout => "",
        stderr => "",
    };
    return $result;
}

```

Version operation

The version operation returns the application version information.


```

sub version {
    my $version_result = {
        major => 1,
        minor => 2,
        patch => 1,
        build => 0
    };
    my @message_a = ();
    $msgObj->collect(\@message_a, INFO, "VOLUMES
$config_h{'VOLUMES'}");
    $msgObj->collect(\@message_a, INFO,
"$config_h{'APP_NAME'}::quiesce");
    $version_result->{message} = \@message_a;
    return $version_result;
}

```

Quiesce operations

Quiesce operation performs application quiesce operation on resources listed in the RESOURCES parameter.

```

sub quiesce {
    my $result = {
        exit_code => 0,
        stdout => "",
        stderr => "",
    };
    my @message_a = ();
    $msgObj->collect(\@message_a, INFO, "VOLUMES
$config_h{'VOLUMES'}");
    $msgObj->collect(\@message_a, INFO,
"$config_h{'APP_NAME'}::quiesce");
    $result->{message} = \@message_a;
    return $result;
}

```

Unquiesce operation

Unquiesce operation is required to unquiesce the application. The list of resources is available in the RESOURCES parameter.

```
sub unquiesce {
  my $result = {
    exit_code => 0,
    stdout => "",
    stderr => "",
  };
  my @message_a = ();
  $msgObj->collect(\@message_a, INFO, "VOLUMES
$config_h{'VOLUMES'}");
  $msgObj->collect(\@message_a, INFO,
"$config_h{'APP_NAME'}::unquiesce");
  $result->{message} = \@message_a;
  return $result;
}
```

Copyright Information

Copyright © 2021 NetApp, Inc. All rights reserved. Printed in the U.S. No part of this document covered by copyright may be reproduced in any form or by any means-graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system- without prior written permission of the copyright owner.

Software derived from copyrighted NetApp material is subject to the following license and disclaimer:

THIS SOFTWARE IS PROVIDED BY NETAPP "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WHICH ARE HEREBY DISCLAIMED. IN NO EVENT SHALL NETAPP BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

NetApp reserves the right to change any products described herein at any time, and without notice. NetApp assumes no responsibility or liability arising from the use of products described herein, except as expressly agreed to in writing by NetApp. The use or purchase of this product does not convey a license under any patent rights, trademark rights, or any other intellectual property rights of NetApp.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.277-7103 (October 1988) and FAR 52-227-19 (June 1987).

Trademark Information

NETAPP, the NETAPP logo, and the marks listed at <http://www.netapp.com/TM> are trademarks of NetApp, Inc. Other company and product names may be trademarks of their respective owners.