

Bash Interactive Command Modules (Bash-ICM)

Document #PLPC-180058

Version 0.1

December 12, 2018

This Document is Available on-line at:

<http://www.by-star.net/PLPC/180058>

Neda Communications, Inc.

Email: <http://www.by-star.net/contact>

Contents

I	About Bash-ICM: This Document, The Software And History	5
1	About BASH-ICM	7
1.1	The Concept Of Interactive Command Modules (ICM)	7
1.2	Bash-ICMs as a subset of the Python-ICM Framework	7
1.3	History Of Bash-ICMs and Python-ICMs	7
1.4	About This Document	7
II	Concept and Model	9
2	Open Services Management Tools	11
2.1	Server To Services Transformation	11
2.2	Open Services Management Tools	11
2.3	GOALS	11
2.4	Common Features	12
2.5	Obtaining LSIP	13
2.6	LSIP License	13
2.7	LSIP Overview	13
III	Libre Platform Base	15
3	Open Platform Libraries	17
3.1	doLib	17
3.2	visLib	17
3.3	ocp-lib	18
3.4	ocp-general	18
3.5	ocp-lineNu	20
3.6	ocpLibUse	20

3.7	opRunEnvLib	20
3.8	opWrappersLib	20
3.9	itemsLib	21
3.9.1	Visibility Rules	21
3.10	opDoAtAsLib	22
4	Seed Scripts	23
4.1	seedActions.sh	23
4.1.1	Description	23
4.1.2	Example	27
4.2	seedSubjectAction.sh	27
4.2.1	Description	27
4.2.2	Example	32

Part I

About Bash-ICM: This Document, The Software And History

Chapter 1

About BASH-ICM

1.1 The Concept Of Interactive Command Modules (ICM)

The concept and python realization of Interactive Command Modules (ICM) is described in:

Unified Python Interactive Command Modules (ICM) and ICM-Players
A Framework For Development Of Expectations-Complete Commands
A Model For GUI-Line User Experience
<http://www.by-star.net/PLPC/180050> — [1]

You should continue reading this document after having read that document.

1.2 Bash-ICMs as a subset of the Python-ICM Framework

Bash-ICMs are a sub-set of Python-ICM Framework. The Remote-Operations model of Python-ICM is not implemented in Bash-ICMs. Bash-ICM-Players are less complete than the Python-ICM-Players.

Bash functions whose names start with “vis_” become automatically visible at Bash-ICM’s command-line-interface. This is similar to how the “Cmnd” classes become automatically visible at Python-ICM’s command-line-interface.

1.3 History Of Bash-ICMs and Python-ICMs

Bash-ICMs predate Python-ICMs and can be considered the origin of Python-ICMs.

Bash-ICMs used to be called IIMs (Interactively Invokable Modules). The collection of Bash-ICMs that are now called BISOS (ByStar Intenernet Services OS) used to be called OSMT (Open Services Management Tools) and LSIP (Libre Service Integration Platform).

1.4 About This Document

Most of this document was written before the evolutions that we mentioned in “History Of Bash-ICMs and Python-ICMs”, hence their terminology predates the Python-ICMs.

We intend to update this document in the future. Since the basic information that is included in the current version of this document still reflects the architecture of the implementation, the current document remains of value.

Part II

Concept and Model

Chapter 2

Open Services Management Tools

2.1 Server To Services Transformation

GNU/Linux demonstrated that large a complete Operating System can be put together purely in the Free Software model.

Various forms of dedicated servers have been integrated based on GNU/Linux. Such server constructs are ad-hoc integrations demanding much expertise.

Collective collaboration towards transformation of ad-hoc servers based on Free Software into mass usable agents for delivery of Libre Services is the next challenge.

Construction of a set of Application Services requires an important extension beyond the underlying software layer. Construction of a set of Application Services requires the integration of a set of software components together to provide useful functionality to the user.

This integration layer must conform to correct principles of **structure** and **consistency**. Thus Free Services represent an extension of the Free Software model based on structured and consistent integration.

The versatile “Glue” needed to bring about the needed structure and consistency is a crucial element for realization of Libre Services. Much effort has been devoted to creation of the initial implementation of this Glue. See “Open Systems Management Tools”, [?] for more details.

2.2 Open Services Management Tools

OSMT (Open Services Management Tools) are a set of tools on top of which various consistent policies can be implemented.

This is a collection tools that collectively lets you consistently manage Unix and Linux systems and some of the tools will also manage Windows system.

2.3 GOALS

Key goals for the design has been:

- Be very Unix centric. Focus on Solaris and Linux

- Limit use of the tools to what is minimally and generically available on plain Unix systems. Namely Korn Shell.
- Be consistent in use of the tools. View this work as a collection. Not bits and pieces here and there.
- Don't view the tools as host management tools, view them as domain management and system management tools.
- Support consistent and simultaneous management of multiple domains. Detection of Sites, Domains and Hosts is an integral part of these tools.
- Tools should be location independent.

2.4 Common Features

The following features are available to all scripts based on `seedActions.sh` and `seedSubjectActions.sh`

```
Tracing:      -T <runLevelNumber>  -- Ex: mmaQmailHosts.sh -T 9 ...
Run Mode:     -n <runMode>          -- Ex: mmaQmailHosts.sh -n runSafe ...
Verbose:      -v                    -- Ex: mmaQmailHosts.sh -v
Force Mode:   -f                    -- Ex: mmaQmailHosts.sh -f
Check Mode:   -c                    -- Ex: mmaQmailHosts.sh -c fast
```

Tracing
=====

DEFAULT: -T 0

Trace Number Conventions:

```
0: No Tracing
1: Application Basic Info
2: Application General Info
3: Application Function Entry and Exit
  4: Application Debugging
5: Wrappers Library
6: Seed Script
7: Seed Supporting Libraries (eg, doLib.sh)
8: ocp_library
9: Quick Debug, usually temporary
```

Run Mode:
=====

DEFAULT: runOnly

```
G_runMode=
showOnly:      at opDo* just show the args always return 0
runOnly:       at opDo* just execute
showRun:       at opDo both runOnly and showOnly
runSafe:       at opDo both show and run, but if protected
                then just show
```

```

showProtected:      Run everything and don't show except for
                    show only protected
showRunProtected:   Run everything and don't show except for
                    run and show roTECTED

runSafe = unprotected: showRun, protected: show
showProtected = unprotected: run, protected: show
showRunProtected = unprotected: run, protected: showRun

Verbose Mode:
=====

G_verbose=
  verbose          When Set, verbose format (eg, line nu, time tag, ...)
                    of Tracing and RunMode are selected.

Force Mode:
=====

G_forceMode=
  force            When Set, force/overwrite mode of operation
                    is selected.

Check Mode:
=====

G_checkMode={fast,strict,full}
  fast:            1) Skip asserting and consistency checks.
                    2) Do less than default, invoker will
                       compensate
  strict:          Do asserts and consistency checks.
  full:            1) Do more than default

```

2.5 Obtaining LSIP

<http://www.neda.com/libre/lpGenesis.sh>

2.6 LSIP License

Afero GPL V3.

2.7 LSIP Overview

Take from presentation.

Part III

Libre Platform Base

Chapter 3

Open Platform Libraries

3.1 doLib

The doLib.sh is a place for common features for script that used the seedSubjectAction. This common features includes:

<i>vis_ls</i>	list all of the functions (hence, equivalent to items) inside the itemsFile.
<i>do_list</i>	
<i>do_describe</i>	describing each items in the itemsFile if opItem_description function exist within the item.
<i>do_itemActions</i>	if the item has a list of itemActions, then it will perform all of them.
<i>doLibExamplesOutput</i>	list all of the common examples for the seedSubjectAction script which include common examples (showMe, seedHelp, ls, list, describe) and common debugging.

To use this feature, put the following in each of the seedSubjectAction script:

```
function vis_examples {
    typeset doLibExamples=`doLibExamplesOutput ${G_myName}`
    cat << _EOF_
EXAMPLES:
${doLibExamples}
--- EVERYTHING ELSE
.....
.....
.....
_EOF_
}
```

3.2 visLib

This library function the same as doLib except this lib is for seedActions script.

3.3 ocp-lib

The ocp-lib loads all of the osmt library. Each of these libraries will be covered in the following sections.

3.4 ocp-general

ocp-general is a collection of several functions which can be used by any scripts. This library will most probably grow over time to simplify tasks.

Function name convention:

- MA_: mail addressing parsing
- ATTR_: Attribute value parsing
- FN_: File Name Manipulation
- USER_: passwd file related activities
- PN_: Path name

The functions included in this library are:

<i>MA_domainPart</i>	Mail address parsing. Print out the domain part. Example: MA_domainPart vendors@neda.com will output neda.com.
<i>MA_localPart</i>	Mail address parsing. Print out the local part. Example: MA_localPart vendors@neda.com will output vendors.
<i>ATTR_leftSide</i>	Attribute value parsing. Print out the left side of the equal sign (=). Example: ATTR_leftSide variable1=value1 will output variable1.
<i>ATTR_rightSide</i>	Attribute value parsing. Print out the right side of the equal sign (=). Example: ATTR_rightSide variable1=value1 will output value1.
<i>FN_prefix</i>	Print out only the basename of a file without the extension. Example: FN_prefix /opt/public/osmt/bin/mmaQmailHosts.sh will output mmaQ-mailHosts.
<i>FN_extension</i>	Print out only the extension of a basename file. Example: FN_extension /opt/public/osmt/bin/mmaQmailHosts.sh will output sh.
<i>FN_dirsPart</i>	Print out only the directory of a specific file location. Example: FN_dirsPart /opt/public/osmt/bin/mmaQmailHosts.sh will output /opt/public/osmt/bin.
<i>FN_nonDirsPart</i>	Print out only the basename of a specific file location. Example: FN_nonDirsPart /opt/public/osmt/bin/mmaQmailHosts.sh will output mmaQmailHosts.sh.
<i>FN_fileDefunctMake</i>	Make a specific file become no longer active in the system by moving the file into another file and chmod to 0000. It requires 2 arguments. First arg is the name of the file that we want to defunct and second arg is the new name and it should not have existed.

<i>FN_dirDefunctMake</i>	Same as the above except it applies to a directory instead of a file.
<i>FN_FileCreateIfNotThere</i>	Create a null file if it does not exist.
<i>FN_dirCreateIfNotThere</i>	Create a directory if it does not exist using the mkdir command.
<i>FN_dirCreatePathIfNotThere</i>	Create a directory path if it does not exist using mkdir -p command.
<i>FN_fileSymlinkSafeMake</i>	Requires 2 arguments: source/origin of a file (should exist) and the target name. If the target exist, skip the symlink process.
<i>FN_fileSymlinkUpdate</i>	Same as <i>FN_fileSymlinkSafeMake</i> except if the target exist, it will remove the old symlink and make a new one.
<i>FN_fileSafeCopy</i>	Required 2 arguments: a source name and a target name. If the target exist, it will skip the copy process.
<i>FN_fileCopy</i>	Same as <i>FN_fileSafeCopy</i> except if the target exist, it will overwrite the old file. Use with caution.
<i>FN_fileSafeKeep</i>	Move a file and rename it with a dateTag extension.
<i>FN_dirSafeKeep</i>	Move a directory and rename it with a dateTag extension.
<i>FN_lineIsInFile</i>	Required 2 arguments: string to check and the filename. It will return 0 if the string is found in the file specified and 1 otherwise.
<i>FN_lineAddToFile</i>	Required 3 arguments: string to check, string to be added, the filename.
<i>FN_textReplace</i>	Required 3 arguments: regexp of text to replace, replacement text, and the filename. The regexp of text to replace has to be in the format of <code>text.*\$</code> .
<i>FN_textReplaceOrAdd</i>	If the text to be replaced exist in the file, it will call <i>FN_textReplace</i> otherwise the replacement text will be added to the file.
<i>FN_fileInstall</i>	This is to ensure that we use FSF's install command. In SunOS the location is in /opt/sfw/bin/install.
<i>FN_grep</i>	This is to ensure that we use grep command that supports "-F", "-v", and "-q". In SunOS, the location is /usr/xpg4/bin/grep.
<i>FN_egrep</i>	This is to ensure that we use egrep command that support "-v", "-q".
<i>_opDoRunOnly</i>	
<i>_opDoShowOnly</i>	
<i>_opDoShowRun</i>	
<i>_opDo</i>	
<i>_opDoAssert</i>	
<i>opDoProtectedBegin</i>	
<i>opDoProtectedEnd</i>	
<i>opDoProtected</i>	
<i>USER_isInPasswdFile</i>	Return 0 if a user is in the /etc/passwd file.
<i>USER_loginGivenHomeDir</i>	Required 1 argument: the path to home directory. If the home directory is found in /etc/passwd, it will output his/her loginName and return 0 otherwise it will return 1.
<i>USER_nextLoginNameGet</i>
<i>PN_fileVerify</i>	List information about file.
<i>FN_fileRmIfThere</i>	Calling <i>PN_rmIfThere</i> .
<i>PN_rmIfThere</i>	If -v is specified, it will enable the verbose mode. You can specified more than 1 file to be removed.
<i>IS_inList</i>	Required 2 arguments: a string to be checked and a list of strings. Return 0 if the string is in the list of strings otherwise return 1.

<i>LIST_getLast</i>	Get the last argument/string in a list.
<i>LIST_getFirst</i>	Get the first argument.
<i>LIST_set</i>	
<i>LIST_minus</i>	
<i>LIST_setMinusResult</i>	
<i>doStderrToStdout</i>	Put standard error to standard output.
<i>G_validateOption</i>	Required 2 arguments: target and a list. If the target is in the list, it will set <code>targetIsValid="TRUE"</code> .
<i>G_abortIfNotSupportedOs</i>	Abort the running script if the OS is not supported. The currently supported OS are SunOS and Linux.
<i>G_abortIfNotRunningAsRoot</i>	Abort the running script if the current user is not root.
<i>G_returnIfNotRunningAsRoot</i>	Return 1 if the current user is not root.
<i>G_validateRunOS</i>	Required 1 argument: a list of OS. If the current OS is in the given list, it will set <code>isValid="TRUE"</code> otherwise it will set <code>isValid="FALSE"</code> and exit.
<i>DOS_toFrontSlash</i>	Convert DOS filename to UNIX system filename.
<i>DOS_toBackSlash</i>	
<i>RELID_extractInfo</i>	Information about product's release ID
<i>logActivitySeparator</i>	
<i>buildAndRecord</i>	

3.5 ocp-lineNu

This library contains functions for debugging purposes.

<i>tm_trace</i>	Depending on what the trace level is, will print out information for debugging purposes. For more complete information, see section ??.
<i>log_event</i>	For logging purposes.
<i>eh_problem</i>	Give out PROBLEM message and continue.
<i>eh_fatal</i>	Give out a FATAL message and exit.

3.6 ocpLibUse

3.7 opRunEnvLib

To setup and verifying the environment configuration on the system.

3.8 opWrappersLib

This script includes these functions:

<i>opNetCfg_paramsGet</i>	Required 2 parameters: <code>clusterName</code> and <code>hostName</code> . Given these 2 parameters, the <code>nedaIPAddr.sh</code> is called and the network setting for this particular cluster and hostname are set.
<i>i_nedaNetParamsGet</i>	Used by the <code>opNetCfg_paramsGet</code> to set all of the network setting as global variables. These global variables are: <code>opNetCfg_ipAddr</code> , <code>opNetCfg_domainName</code> , <code>opNetCfg_netmask</code> , <code>opNetCfg_networkAddr</code> , <code>opNetCfg_defaultRoute</code> .

3.9 itemsLib

itemsLib ia a set of facilities that operate on any item files.

<i>opItem_description</i>	Whenever -i describe is executed, it will call <i>opItem_description</i> and this function will look for <i>iv_descriptionFunction</i> in each of the item in the itemsFile. If it exist, the description will be printed out.
<i>opItem_selectClusterFiles</i> <i>opItem_ifAvailableInvoke</i> <i>opItem_isAvailable</i>	It will check whether the item is available to hostMode (by calling <i>opItem_isAvailableToHostMode</i>) and if it is within the cluster (by calling <i>opItem_isWithinClusterScope</i>). It will return 0 if everything is correct.
<i>opItem_isAvailableToHostMode</i> <i>opItem_isAvailableToOs</i> <i>opItem_isWithinClusterScope</i>	Subject variables should be all set (iv_itemScopeVisibleHosts, iv_itemScopeVisibleClusters, iv_itemScopeHiddenHosts). Returns: 0 if disk within scope and should be acted upon 1 if disk is tagged to be hidden 2 if disk not in the cluster and also not tagged as visible

3.9.1 Visibility Rules

items Visibility

By adding

```
iv_itemScopeVisibleHosts -- List of hosts outside of the clusters
                           item is visible to
iv_itemScopeVisibleClusters -- List of clusters, item is visible to
iv_itemScopeHiddenHosts -- List of hosts inside of the clusters
                           item is visible to
```

you can then use *opItem_isWithinClusterScope* to check the visibility of the item.

By adding

```
iv_itemAvailableToHostModes
```

you can then use *opItem_isAvailableToHostMode*.

By adding

```
iv_itemAvailableToOsType -- matched against opRunOsType
iv_itemAvailableToMachineArch -- matched against opRunMachineArch
```

you can then use *opItem_isAvailableToOs*.

runMode Visibility

Cluster Visibility

Binary Visibility

3.10 opDoAtAsLib

Chapter 4

Seed Scripts

4.1 seedActions.sh

4.1.1 Description

NAME

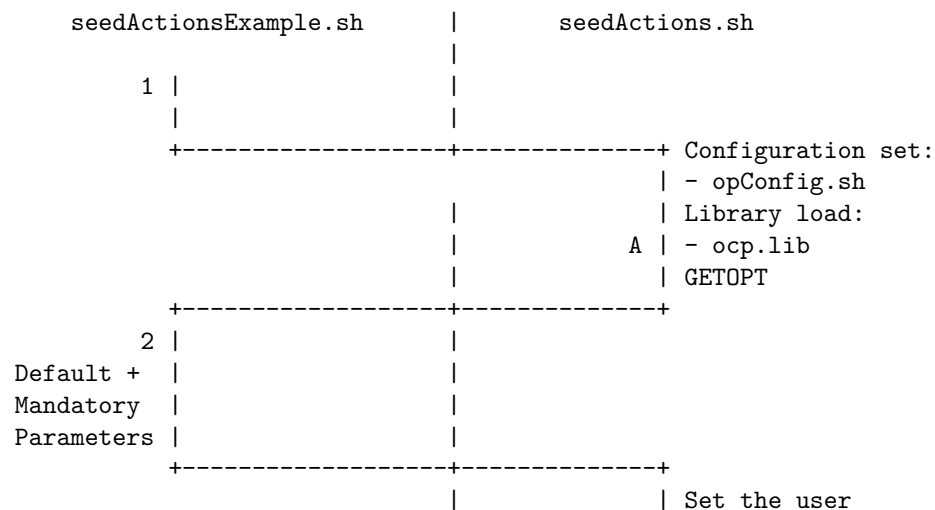
seedAction.sh

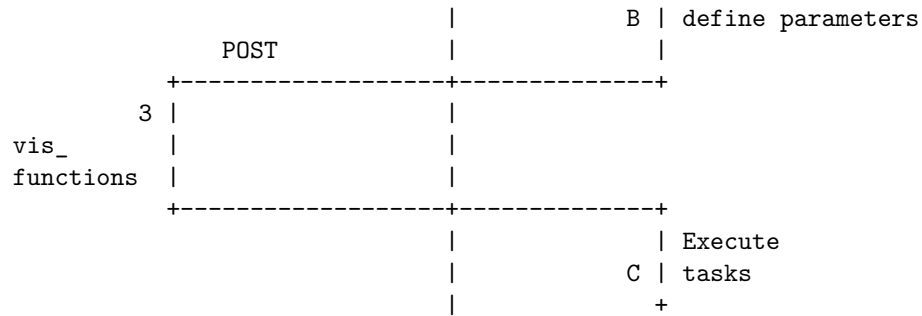
DESCRIPTION

seedActions.sh is the basis of a tool for grouping a number of functions within a shell script and allowing for their execution and maintenance in a consistent way.

A large number of common features are provided by simply loading seedActions.sh. seedActions.sh integrates itself with your script in three stages.

Below is the diagram of how this seedActions.sh works:





In this description, the routine is:

```

part 1 called --> part A executed -->
part 2 called --> part B executed -->
part 3 called --> part C executed.

```

First, `mmaExampleActions.sh` is calling part 1:

```

if [ "${loadFiles}X" == "X" ] ; then
    seedActions.sh -l $0 $@
    exit $?
fi

```

As a result, the `seedActions.sh` is executed and the first thing that `seedActions.sh` do is execute Part A:

```

- load opConfig.sh
- load ocp-lib.sh (OCP Library)
- process GETOPT (get options)

```

After Part A is executed, `mmaExampleActions.sh` declare the default parameter with tags (`typeset -t`) if any. This is also known as PRE loading.

```

typeset -t FirstName=MANDATORY
typeset -t LastName=MANDATORY
typeset -t SubsSelector=""
.....

```

This is where all of the necessary parameters are set, including the default and mandatory parameters.

parameter=value from the command line must match a `typeset -t`.
The initial value of mandatory variables is MANDATORY

After all the parameters are set, `seedActions.sh` executes Part B:

```

- set all of the user's define parameters.

```


After we have all the parameters, part 3 is called (POST Loading). Part 3 only executed if function called `G_postParamHook` exist within the script.

command line "someFunction" maps to function: `vis_someFunction`

OPTIONS

All scripts base on `seedActions.sh` get `getopts` with the following options:

- T traceLevel Use for debugging purposes -- tracing, with traceLevel being a number between 0-9.
- i Run a specific visible function within the script.
- p Specify the required/default parameters. parameter=value from the command line must match a typeset -t. For example:
 -p FirstName=Homer ...
- l Specify the file for loading.
- u Gives USAGE Info. The usage info automatically lists all visible functions without the prefix "vis_".

VISIBLE FUNCTIONS

- The visible functions (indicated by prefix `vis_`) are internal functions which are exposed externally.
- It can accept `ARGS` on command line.

CONVENTIONS

- In every script, `vis_help` is always put on top. The idea being that a description of the script can always be accessed through "`-i help`" in the command line.
- Those based on `seedActions.sh` should end in a category of actions as a VERB. The most generic form is the verb Action itself. For example: `mmaSendmailAction.sh`
- The `noArgsHook` function will be available in some of the script.
If a default action is applicable to a script, the `noArgsHook` is called, if it exists, based on the recognition that a default action will be performed.
If `noArgsHook` is not specified and the script is run with no options, then this warning will be displayed:

"No action taken. Specify options. See -u"

EXAMPLE

Mandatory parameters:

the initial value of mandatory variables is MANDATORY

e.g.

```
typeset -t FirstName=MANDATORY
```

In order to force this parameter to be set (hence MANDATORY) call the `opParamMandatoryVerify` within the function that needs this parameter. When `opParamMandatoryVerify` is executed, it will check all of the parameters that has initial value MANDATORY. If it is not set, return error.

Optional parameters:

the optional parameters has initial value other than MANDATORY.

`vis_help`: the `vis_help` can always be accessed through "-i help" in the command line

Example of usage: `anyScript.sh -i help`

Example of code:

```
vis_help () {
    cat << _EOF_
        Put any text here for information related to this script.
    _EOF_

    exit 1
}
```

`noArgsHook`:

e.g.

```
noArgsHook="noArgsHook"
noArgsHook() {
    # If no args, default action or usage

    if [ "$*X" == "X" ]
    then
        echo "No Defaults Specified"
        echo "Specify Options -- See -u for list of visible actions"
        usage
    fi
}
```

Take a look at `mmaExamplesActions.sh`

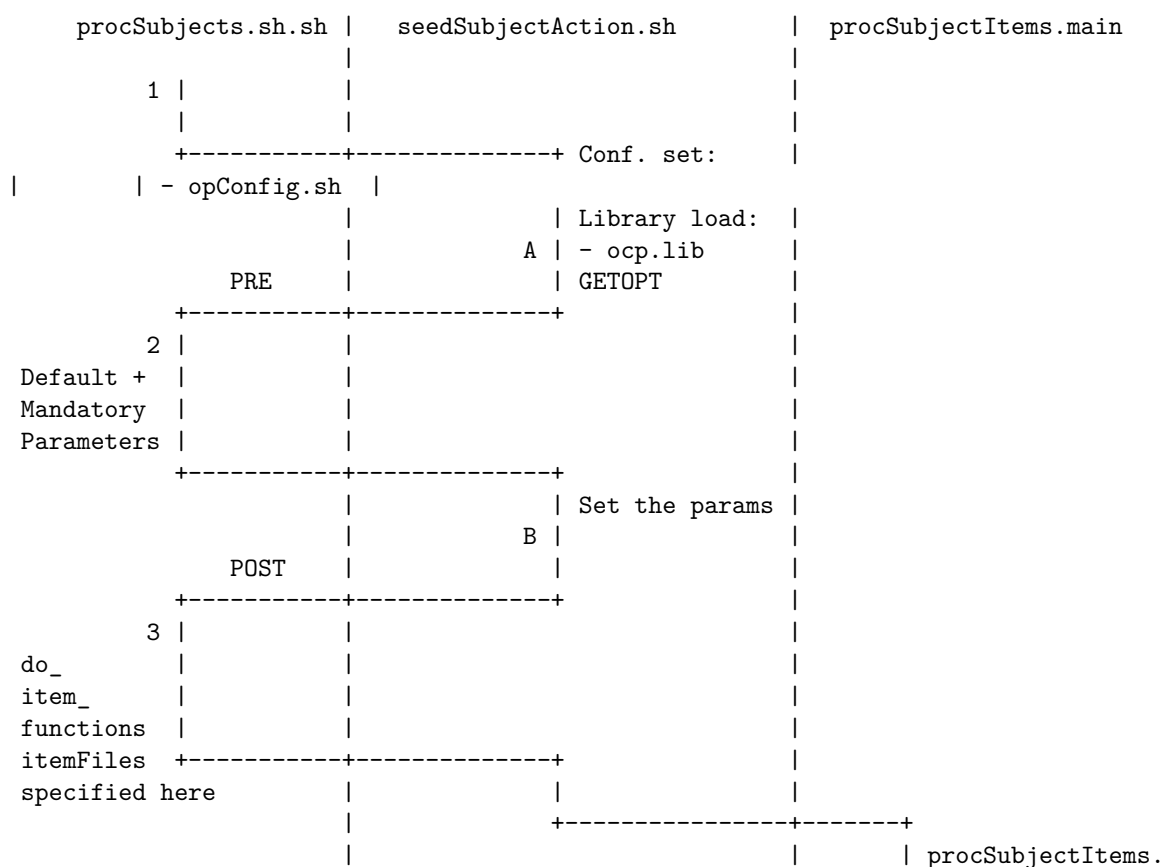
4.2.1 Description

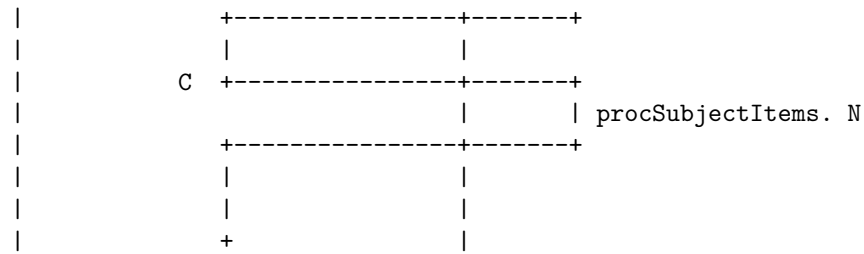
seedSubjectAction.sh

seedSubjectAction.sh is the basis of a tool for grouping a number of functions within a shell script and allowing for their execution and maintenance in a consistent way.

A large number of common features are provided by simply loading `seedSubjectAction.sh`. `seedSubjectAction.sh` integrates itself with your script in three stages.

Below is the diagram of how this seedSubjectAction.sh works:





In this description, the routine is:

```

part 1 called --> part A executed -->
part 2 called --> part B executed -->
part 3 called --> part C executed.
  
```

First, seedSubjectActionExample.sh is calling part 1:

```

if [ "${loadFiles}X" == "X" ] ; then
    seedSubjectAction.sh -l $0 $*
    exit $?
fi
  
```

As a result, the seedSubjectAction.sh is executed and the first thing that seedSubjectAction.sh do is execute Part A:

```

- load opConfig.sh
  - load ocp-lib.sh (OCP Library)
  - process GETOPT (get options)
  
```

After Part A is executed, seedSubjectActionExample.sh declare the default parameter with tags (typeset -t) if any.

This is also known as PRE loading.

```

if [ "${loadSegment}" == "PRE_" ] ; then
    # Mandatory parameters
    typeset -t VirDomRoot=MANDATORY
    typeset -t VirDomTLD=MANDATORY

    # Optional parameter = default value
    typeset -t SiteName=xyzPlus
    .....
  
```

This is where all of the necessary parameters are set, including the optional and mandatory parameters.

parameter=value from the command line must match a typeset -t.

The initial value of mandatory variables is MANDATORY and the optional parameters become the default value.

After all the parameters are set, seedSubjectAction.sh executes Part B:

```

- set all of the user's define parameters.
  
```

After we have all the parameters, part 3 is called (POST Loading). Part 3 only executed if function called `G_postParamHook` exist within the script.

The `setBasicItemsFile` is called here. See CONVENTIONS section for how `setBasicItemsFiles` works.

The `itemsFile` are loaded from the `procSubjectItems` file:

```
procSubjectItems.<specificCluster>
```

where `procSubjetItems` is the corresponding `procSubjects.sh`, `<specificSite>` is one of `main`, `office`, `public`, etc.

When `procSubjectItems` is executed, `itemPre` and `itemPost` are defined, if there is any.

`itemPre` is a place where all the default and mandatory parameters are specified.

`itemPost` derived defaults.

After the `itemsFile` is loaded, `"subject"` and `"action"` are defined.

command line `"subject"` maps to function: `item_subject`

command line `"action"` maps to function: `do_action`

By convention, it calls `itemAction_action`.

OPTIONS

All scripts base on `seedSubjectAction.sh` get `getopts` with the following options:

- T `traceLevel` Use for debugging purposes -- tracing, with `traceLevel` being a number between 0-9.
- a Run the specific action. The `"action"` automatically lists all the action available without the `"do_"` prefix. Also applies to `itemCmd_` as well.
- s Apply the -a `"action"` to a specific `"subject"`. The `"subject"` automatically lists all the subject available without the `"item_"` prefix.
- i Run a specific visible function within the script.
- p Specify the required/default parameters. `parameter=value` from the command line must match a `typeset -t`. For example:

- p FirstName=Homer ...
- l Specify the file for loading.
- u Gives USAGE Info. The usage info automatically lists all visible functions without the prefix "vis_".

CONVENTIONS

- In every script, vis_help is always put on top. The idea being that a description of the script can always be accessed through "-i help" in the command line.
 - Those based on seedSubjectAction.sh should end in the plural of the OBJECT, if there are categories of actions related to the objects those as verbs come before the plural of the object.
For example: opDiskDrives.sh or mmaQmailHosts.sh
- The seed of the items file is the singular of the fileName plus Items. For example opDiskDriveItems.sh or mmaQmailHostItems.sh.
- The noArgsHook function will be available in some of the script.
If a default action is applicable to a script, the noArgsHook is called, if it exists, based on the recognition that a default action will be performed.
If noArgsHook is not specified and the script is run with no options, then this warning will be displayed:
"No action taken. Specify options. See -u"
 - The noSubjectHook function will be available in some of the script.
This function will be executed if there is no subject specified.
 - The firstSubjectHook and lastSubjectHook are typically used when the subject is all. Most of the time, it will be used for printing summary of the itemsFile.
 - setBasicItemsFiles procSubjectItems
Here are the flow how setBasicItemsFiles works:
if there is procSubjectItems.main, then add it.
if there is procSubjectItems.clusterName, then add it.

if there is none of the above then
if there is procSubjectItems.site, then add it.

if there is procSubjectItems.otherName, just ignore it.

- Here is a scenario:
- For example, suppose we have all of these files:
procSubjectItems.main, procSubjectItems.office,
procSubjectItems.home, procSubjectItems.otherCluster
and we are running from an office machine environment
then only procSubjectItems.main and procSubjectItems.office
are loaded and the other are ignored.
 - The itemsFile policy:
item_SSSS (SSSS is the subject)
itemPre
iv_specialize
itemPost
itemCmd_
 - Built in function:
list -- built in action
all -- built in subject
Example of use in command line:
anyScript.sh -s all -a list
This command will enumerate all the subject item_ entries from
the ItemsFile and list all of the paramaters corresponding to
each subject item_.

EXAMPLE

Mandatory parameters:

the initial value of mandatory variables is MANDATORY
e.g.
typeset -t FirstName=MANDATORY

Optional parameters:

typeset -t FirstName=homer

vis_help:

the vis_help can always be accessed through "-i help"
in the command line

Example of usage: anyScript.sh -i help

Example of code:

```
vis_help () {
    cat << _EOF_
        Put any thext here for information related to this script.
    _EOF_

    exit 1
}
```

noArgsHook:

e.g.

```
noArgsHook="noArgsHook"
noArgsHook() {
    # If no args, default action or usage

    if [ "$*X" == "X" ]
    then
        echo "No Defaults Specified"
        echo "Specify Options -- See -u for list of visible actions"
        usage
    fi
}
```

Use of parameters in vis_ function:

print \${FirstName} will give result "homer".

ItemsFile Selection:

There are 2 ways to load the procSubjectItems:

1. Automatic ItemsFile Selection

```
setBasicItemsFiles procSubjectItems
```

2. Manual ItemsFile Selection

```
ItemsFile=${opSiteControlBase}/${opSiteName}/procSubjectItems.main
```

do_ description:

The do_AAA function is the AAA "action" taken to some
"subject" item_.

By convention it calls itemAction_AAA.

itemCmd_ description:

4.2.2 Example

Take a look at mmaExamplesObjects.sh

Bibliography

- [1] "Neda Communications Inc". "interactive command modules (icm) and players a framework for cohesive generalized scripting a model for gui-line user experience ". Permanent Libre Published Content "180050", Autonomously Self-Published, "July" 2017. <http://www.by-star.net/PLPC/180050>.