NSGUI Script interface Specification

Draft

By Shui Hung Kwok, March 24, 2015

# Introduction

NSGUI is the GUI that executes NS scripts. NS scripts contain commands and conditions that control the telescope and guider. To extend the capability of the NSGUI, new commands can be added by means of this interface. The NSGUI invokes these new commands and processes the output of these commands to determine the results of their execution.

The terms “external commands” or “external scripts” are used in this document to refer to the commands that are not built in into NSGUI.

## Revision History

|  |  |  |
| --- | --- | --- |
| Date | Author | Description |
| 2015-03-23 | S. H. Kwok | Initial draft |
| 2015-03-24 | S. H. Kwok | Added example implementation, possible scripts.  Values cannot have spaces. |

# General requirements:

* External scripts can trigger one or multiple actions or query statuses.
* Execution of external scripts shall be short, i.e. they shall provide a ‘no-wait’ option or have a timeout option if the execution is expected to be long.
* External scripts shall not spawn background processes that run after the script terminates.
* External scripts must allow named parameters, such that they can be given in any order.
* External scripts parameters must have default options, such that the parameters can be omitted.
* External scripts must provide an option to query the parameter list, i.e. similar to help, but formatted for NSGUI to read.
* External scripts outputs must be strictly formatted according to the syntax defined below.
* External scripts must be registered with the NSGUI.

# Theory of Operation

A list of external scripts is defined in the NSGUI configuration. Using this list and a help program, an external script configuration file is generated, which contains the names of the scripts and their parameter lists. This configuration file can be created manually as well. However, it is recommended to generate this configuration file by invoking the scripts’ parameter query option. This way, changes in the scripts can be easily propagated to NSGUI, i.e. via a Makefile.

A NS script can use any of the registered external scripts as long as the parameters used in the NS script match the ones defined by the external scripts. Before the NSGUI invokes external scripts via the standard system call, it must build the command line with the script name and its parameters. This command line is then passed to a standard shell for execution. The NSGUI reads the outputs of the external script and waits for its termination. The external script includes an explicit execution status. NSGUI checks this execution status as well as the exit and error status from the shell execution, in case of abnormal termination.

For action commands, the execution status can be success or failure. For query or condition commands, the output contains additional information depending on the query.

# Interface Description

## External Script Command Line

The syntax of the external script command line is:

scriptName param1=value1 param2=value2 param3=value3

where:

* scriptName is the name of the script, full path or implicit via PATH
* param1, param2, etc, are the names of the parameters
* value1, value2, etc, are the parameter values

Values can be numbers or strings. White spaces within values are not allowed. No white spaces are allowed before and after the ‘=‘ symbol.

Example:

startExposure TTIME=20

## External Script Output

The output of the external script consists of a set of keywords/value pairs, each pair on a separate line, where the keywords must match the parameter list. A mandatory keyword, EXECSTATUS must be included to indicate if the execution was successful, OK, or failed, ERROR. An optional keyword, STATUSMSG, contains a message string to be passed to the GUI.

EXECSTATUS=OK|ERROR

STATUSMSG=‘Status or error message if any’

kwd1=value1

kwd2=value2

kwd3=value3

where values include everything up to end of line.

Also leading and trailing white spaces are discarded.

If EXECSTATUS is ERROR, the keyword/value pairs are ignored.

For action commands, the kwd/value pairs are ignored. For query or condition commands, the keywords correspond to the parameter values in the command line. For example:

queyScript KWD1=TEMP1 KWD2=TEMP2 KWD3=FILTNAME

The output of the command above would be:

EXECSTATUS=OK

STATUSMSG=“Everything is fine”

TEMP1=-2

TEMP2=-42

FILTNAME=OPEN

## Parameters query

An external script must provide a list of allowed parameters, if the special query parameter ‘queryparam=1’ is given. No other parameters must be given when this parameter is used.

The output of the parameter query is a list of keyword/value pairs, where the keywords are the names of the allowed parameters and the values are comma separated lists defined as follows:

type,unit,default,range,description

where

type is integer, float or string

unit is any unit of measurement, can be empty

default is the default value if the parameter is omitted

range is a ‘:’ separated list of identifiers or a pair of numbers in the form “low:high”

description is a string describing this parameter

For example, the parameters query output for ‘startExposure’ would be:

EXECSTATUS=OK

STATUSMSG=“No problem”

TTIME=float,s,1,0:3600,Exposure time

DEMOPARAM=string,,Demo,Example1:Example2:Demo,Example parameter

Note that unit is empty for DEMOPARAM.

For a query or condition command, the output could look as follows:

EXECSTATUS=OK

STATUSMSG=“OK”

KWD1=string,,TEMP1,,Temperature device 1

KWD2=string,,TEMP2,,Temperature device 2

KWD3=string,,FILTNAME,,Filter name

# Example Implementation

The following steps describe how to implement a script for starting exposures.

Define the script name:

startExposure

Define the parameters:

TTIME,float,s,1,0:3600,Exposure time DEMOPARAM=string,,Demo,Example1:Example2:Demo,Example parameter

where float is the type, 1 default exposure time, 0:3600 valid range. The last portion is the description.

Find out what keywords are needed:

In this case, the keyword is TTIME from the service “kbds”. For convenience, the parameters and the keywords names are the same. But this is not required.  
A second keyword is required to implement this script: STARTNSX also from kbds. This keyword does not need to be a parameter.   
Parameters do not have to be keywords. In this case, DEMOPARAM can be used for controlling other aspects of the script rather than controlling the instrument.

#!/bin/sh

PARAMDEFINITION="\

TTIME,float,1,0:3600,Exposure Time\n\

NOWAIT,int,1,0:1,NO wait option\n\

DEMOPARAM,String,,Demo,Example1:Example2:Demo,Example parameter\

"

reportStatus()

{

echo "EXECSTATUS=$1"

shift

echo 'STATUSMSG="'$\*'"'

}

setDefaults()

{

TTIME="1"

DEMOPARAM="Demo"

NOWAIT="1"

export TTIME DEMOPARAM NOWAIT

}

handleQueryParam()

{

for p in $\*

do

case $p in

queryparam\*)

reportStatus OK No problem

/bin/echo "$PARAMDEFINITION"

exit 0

;;

esac

shift

done

}

checkParameters()

{

for p in $\*

do

case $p in

TTIME\*|ttime\*)

TTIME=`echo $p | awk -F= '{print $2}'`

;;

DEMOPARAM\*)

DEMOPARAM=`echo $p | awk -F= '{print $2}'`

;;

\*)

# noop

;;

esac

shift

done

}

performAction()

{

# Perform the action

if [ $DEMOPARAM == 'FakeError' ]

then

cat nofile 2> /dev/null # this will cause an error

echo "Bad command"

echo "Somethign missing"

echo "Or invalid parameters"

else

echo

echo "This simulates the action"

echo "Calling modify -s kbds TTIME=$TTIME DEMOPARAM=$DEMOPARAM"

echo

echo "Output of the action comes after the status message"

echo

echo "Done"

echo

fi

}

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Main part

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

handleQueryParam $\*

setDefaults

checkParameters $\*

OUTPUT="`performAction`"

if [ $? != 0 ]

then

reportStatus ERROR Something went wrong

echo "$OUTPUT"

else

reportStatus OK Setting keywords

echo "Output is"

echo "$OUTPUT"

fi

The script above demonstrates the structure of the interface script. It is written in Bourne Shell, the most basic of all shell scripts, i.e. the smallest common denominator and fully compatible across platforms. It also allows subroutines to help separate the tasks. There are four important subroutines:

* handleQueryParam  
  This subroutine handles the “queryparam” case. The parameter definitions are all contained in one single environment variable, PARAMDEFINTION, which would be different for different scripts. A possible additional option could be “help”, which could give a human readable help text, maybe defined in HELPTEXT.
* setDefaults  
  Each script defines its own defaults.
* checkParameters  
  This subroutine parses the parameter list.
* performAction  
  This is the main subroutine that performs the action. This subroutine can call other scripts or any other available command, locally or remotely.

The error status is reported first and then the captured output of the performed action is released. This allows the NSGUI to distinguish the error case from the normal case.

Query scripts are similar. The expected output format corresponds to the output of “show”, namely keyword/value pairs.

# Possible Scripts

CCD Setup  
Prepares CCD for an exposure sequence

Start Exposure  
Starts the exposure

Check Exposure  
Checks if exposure is complete

CCD Shuffle  
Shuffles CCD

CCD Readout  
Performs CCD readout

CCD Cleanup  
Cleans up CCD