Contents

[Operator Instruction 2](#_Toc503885179)

[Perquisites 2](#_Toc503885180)

[Steps to launch the script 2](#_Toc503885181)

[Argument Descriptions 2](#_Toc503885182)

[Syntax Examples 3](#_Toc503885183)

[Developer 4](#_Toc503885184)

[Theory 4](#_Toc503885185)

[Main Definition 4](#_Toc503885186)

[Upgrade Class 4](#_Toc503885187)

[Initial 4](#_Toc503885188)

[Upgrade Class Main 5](#_Toc503885189)

[Logging 5](#_Toc503885190)

[Errata 6](#_Toc503885191)

[Change Log 7](#_Toc503885192)

# Operator Instruction

This section of the documentation provides instructions on how to launch the script and the options the user has access to. For information on editing functionality or fixing bugs please see the developer section of the instructions.

## Perquisites

The user first needs to open an interactive qconsole before launching the script. A minimum of 8GBs of memory is recommended. If the user is upgrading larger projects for Stratix or Arria more memory may be required. **NOTE:** If Quartus pro is used only pro users will be able to open the project.

### Steps to launch the script

1. arc shell python\_altera/2.7.3
2. arc shell acds/<Quartus Version> (or whichever version is preferred \*\*\*see the note above\*\*\*)
3. Navigate to the directory storing the python script.
4. python dustinRewrite.py --<argument>=<string>

**NOTE:** Steps 1 and 2 only needed to be completed before the initial launch of the script. Steps 3 and 4 can be repeated as many times as needed after first time steps 1 and 2 are completed.

## Argument Descriptions

--single\_upgrade

The input for this argument is a string. The string needs to contain the path to the par file that needs to be upgraded. Only one par file can is allowed to be contained by this directory.   
Example: --single\_upgrade=/path/to/dir

--multiple\_upgrade

The input for this argument is a string. The string needs to contain the path to the directory that contains the par files that needs to be upgraded. This argument will create individual files for each par found in the directory passed to the multiple\_upgrade file. This is done to support the class that is used to perform the individual upgrades.  
Example: --multiple\_upgrade=/path/to/dir

--package

The input for this argument in a string. The string needs to contain the path to the directory that contains the project that needs to be packaged. This argument will package the project of the design store. Only one project is allowed to be contained by the directory otherwise the script will not operate properly.  
Example: --package=/path/to/dir

## Syntax Examples

*python* dustinRewrite*.py --single\_upgrade=/data/dustinhe/autoUpgrade/project*

-will update the project in the folder specified

*python* dustinRewrite*.py --s=/data/dustinhe/autoUpgrade/project*

-will update the project in the folder specified

*python* dustinRewrite*.py --multiple\_upgrade=/data/dustinhe/autoUpgrade/multiProject*

-will update the projects in the directory specified

*python* dustinRewrite*.py --m=/data/dustinhe/autoUpgrade/multiProject*

-will update the projects in the directory specified

*python* dustinRewrite*.py --package=/data/dustinhe/autoUpgrade/packageProject*

-will package the project in the directory for the design store

*python* dustinRewrite*.py --p=/data/dustinhe/autoUpgrade/packageProject*

-will package the project in the directory for the design store

# Developer

This section of the documentation exists to help future PSG employees with understanding how the script works and how to edit it. The section starts out with the theory which gives a high-level overview of how the script operates. Next is a section detailing the main Definition of the script. The upgrade class is a more detailed overview of the class structured used to upgrade each project. Last, the logging section details what is logged and how to use the log for debugging purposes.

## Theory

The Quartus projects are stored over multiple source and settings files (qpf, qsf, sdc, qip, ect). The majority of the settings files use the tcl syntax and reference the design files used in the project. The script takes advantage of this structure and uses it to both build a file list of all the source and settings files in addition to upgrading any IP used in the design.

The script starts with the Quartus settings file (.qsf). This file contains all the settings in addition to any source, qip, and qsys files used by the project. It uses a list of tags to find all relevant files. These tags are stored in the initial of the upgrade class initial definition. From this file the script then moves on the any Quartus IP (.qip) or Platform Designer files (.qsys). The script will pars the qip files for any design files using the same tags for the qsf file.

After the qsf file and all qip files are parsed the script will upgrade an IP cores. It does this by finding the associated Qsys, Verilog, or VHDL file and using the quartus shell to auto upgrade the IP.

Once the upgrade is completed the files are parsed again to ensure that no new files from the upgrade are left off the file list. The file list is then written to a text file and the project is archived for the design store.

Last the archived file of the upgraded project is copied into a test directory and extracted. The script then attempts to compile the project in the test directory to verify that it has be archived properly.

## Main Definition

The main definition of the script is only responsible for parsing the arguments passed to the script by the user. Currently there is only one argument that can be passed. However, as functionality is expanded this definition will be responsible for parsing more options and calling more classes.

## Upgrade Class

The upgrade class is the main engine of the whole script. This class is used to parse all the files in addition to upgrading them. Using the class structure is extremely beneficial because of the decreased overhead of passing variables in and out of definitions.

Besides the all of the definitions stored with in the class there two main parts of the class. The initial and the upgradeClassMain.

### Initial

The initial def in the class is used to declare and initialize all of the variables used by the class. The varibles in the initial are broken into six categories: flags, generated lists, user lists, user names, generated names, and commands.

The flags are all boolians that are used to indicate various situations in the script. The flags initial values should not be changed for the most part.

The generated lists are lists that are generated by the script. All of these lists are initialed empty accept for the fileList. The fileList is initialed with the platform\_setup.tcl and filelist.txt files. All projects packaged for the design store must include these files. For the most part, the initial state of these lists should be left as is.

User lists are lists initialed and used by the script to find files. For example the masterImageFileTypes is used find pre-compiled files included in the project but not sourced in any of the Quartus files. If there is a file type missing from one of these lists it should be added here by the user.

User names are used to name and find different files in the project. For example the test testDirName is the name of the directory used to test the archived project produced by the script to verify success.

Generated names are produced by the script and should be initialed as empty strings. If they are initialed the value will just be over written later on in the script.

Last commands are strings used by the script to achieve different operations. These commands contain the syntax used by the Quartus shell to upgrade, extract, and compile projects. If the syntax of the Quartus shell is updated or changed the script can be fixed by adjusting the initial state of these strings.

### Upgrade Class Main

The upgrade class main is where each def in the class is called. This is also where most of the return logic is contained in the case of an error. If a new def is added into the script the def will need to be called in the upgrade class main. Additionally, if the order of the definitions needs to be checked or changed that action can be completed here. However, before changing the order the definitions operate double check that all dependencies will be met for each definition.

## Logging

Throughout the script various steps and variable changes are logged. They are logged using the logging.debugging command. All log entries are saved in the log file declared at before the upgrade class is run. Using the log file can streamline debugging any issues found the script. Additionally, code braking errors are logged using the ERROR syntax to make parsing logs with other scripts to check for completed functionality easy.

# Errata

* The script does not support upgrading any projects that do not come packaged in a par file
* Does not support multiple qsf files
* Does not support nested QIP files
* Does not support custom NIOS instructions.

# Change Log

This log is used to tack major changes in the document and who completed the changes.

|  |  |  |
| --- | --- | --- |
| **Name** | **Date** | **Notes** |
| Unknown | ??? | Initial Creation. |
| D. Henderson | 12-1-17 | Ported documentation to rewrite of the script. |
| D. Henderson | 12-18-17 | Added developer and errata sections to aid in fixing bugs. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |