IC Validator Launch User Guide

Version T-2022.03-SP3, September 2022

SYNOPSYS®

Copyright and Proprietary Information Notice

© 2022 Synopsys, Inc. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.
All other product or company names may be trademarks of their respective owners.

Free and Open-Source Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

www.synopsys.com

Feedback

Contents

New in This Release 6
Related Products, Publications, and Trademarks 6
Conventions 7
Customer Support 8
Statement on Inclusivity and Diversity 8

1. Introduction to IC Validator Launch 9
Differences Between VUE Execution and ICV Launch 9
ICV Launch Flow 10

2. Using ICV Launch 12
ICV Launch Command-Line Options 12
Licensing and Resource Requirements 13
Job Submission From the Command Line 13
Running ICV Launch From the Command Line in Batch Mode 14
Running ICV Launch Within a Design Tool 14
Cadence Virtuoso 15
Custom Compiler 15
IC Validator WorkBench 16

3. Running DRC Using ICV Launch 18
Automated Entry Validation 19
Dynamic Runset Inclusion 21
User Customization 26
Using DRC Recipes 27
Creating and Editing IC Validator Recipe Files 29

4. Running LVS Using ICV Launch 33

5. Template, Runset Configurations, and Command-Line File Utilization 39
Template File 39
Keywords for Built-in ICV Launch GUI Entries 42
Importing Versus Loading Template Files 43
Runset Configuration File 43
Command-Line File 44

6. Configuring Distributed Processing Jobs Using IC Validator 46
Local Host 47
Reserved Hosts 47
LSF and SGE 48
Other 49

7. Environment Variable and Preprocessor Directive Control 51
Defining Preprocessor Macros and Environment Variables 52
Special Environment Variables 55

8. ICV Launch Trigger Functionality 56
Export Triggers 56
Setting Custom Compiler Export Options 57
Setting Virtuoso Export Options 58
Setting IC Validator WorkBench Export Options 62
Initialization Trigger 62
Pre or Post-Execution Triggers 63
Trigger Metacharacter Language 64
External Trigger Definition Example 66

9. GUI Customization 68
Customization Language Introduction 70
Types of Customization Widgets 73
Built-In Widget Reference Names 75
Referencing Widget Names 77

A. Example Trigger Scripts 79

B. Example Customization Scripts 90

C. IC Validator Installation 94

Preinstallation Setup 94

Changing Operating System Limits 94

Finding Your Site Identification Number 95

Configuring and Downloading the Synopsys Installer 96

Product Installation 97

D. Environment Variable Setup 102

Runset Caching 102

Configuring Runset Caching Environment Variables 102

Command-Line Option Compatibility 103

Creating or Modifying a .cshrc File 104

Installing IC Validator Integration in Cadence Virtuoso 105

About This User Guide

This user guide provides an introduction to IC Validator Launch usage, including runset requirements, available command-line options, and general ICV Launch setup options. It is designed to improve the user experience by providing interactive execution control that is easy to adopt and configure.

This preface includes the following sections:

- New in This Release
- Related Products, Publications, and Trademarks
- Conventions
- Customer Support
- Statement on Inclusivity and Diversity

New in This Release

Information about new features, enhancements, and changes, known limitations, and resolved Synopsys Technical Action Requests (STARs) is available in the IC Validator Release Notes on the SolvNetPlus site.

Related Products, Publications, and Trademarks

For additional information about the IC Validator tool, see the documentation on the Synopsys SolvNetPlus support site at the following address:

https://solvnetplus.synopsys.com

You might also want to see the documentation for the following related Synopsys products:

- Synopsys® Custom Compiler™
- Synopsys IC Compiler™
- Synopsys IC Compiler™ II
- Synopsys Fusion Compiler™
- Synopsys StarRC™

Feedback

- Synopsys IC Validator WorkBench
- Cadence® Virtuoso® Layout Editor

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates syntax, such as `write_file`.
`Courier italic`	Indicates a user-defined value in syntax, such as `write_file` *`design_list`*
`Courier bold`	Indicates user input—text you type verbatim—in examples, such as `prompt>` **`write_file top`**
`Purple`	• Within an example, indicates information of special interest. • Within a command-syntax section, indicates a default, such as `include_enclosing = true | false`
[ ]	Denotes optional arguments in syntax, such as `write_file [-format` *`fmt`*`]`
...	Indicates that arguments can be repeated as many times as needed, such as *`pin1 pin2 ... pinN`*.
|	Indicates a choice among alternatives, such as `low | medium | high`
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Bold	Indicates a graphical user interface (GUI) element that has an action associated with it.
Edit > Copy	Indicates a path to a menu command, such as opening the **Edit** menu and choosing **Copy**.
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing C.

Feedback

Customer Support

Customer support is available through SolvNetPlus.

Accessing SolvNetPlus

The SolvNetPlus site includes a knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The SolvNetPlus site also gives you access to a wide range of Synopsys online services including software downloads, documentation, and technical support.

To access the SolvNetPlus site, go to the following address:

https://solvnetplus.synopsys.com

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to sign up for an account.

If you need help using the SolvNetPlus site, click REGISTRATION HELP in the top-right menu bar.

Contacting Customer Support

To contact Customer Support, go to https://solvnetplus.synopsys.com.

Statement on Inclusivity and Diversity

Synopsys is committed to creating an inclusive environment where every employee, customer, and partner feels welcomed. We are reviewing and removing exclusionary language from our products and supporting customer-facing collateral. Our effort also includes internal initiatives to remove biased language from our engineering and working environment, including terms that are embedded in our software and IPs. At the same time, we are working to ensure that our web content and software applications are usable to people of varying abilities. You may still find examples of non-inclusive language in our software or documentation as our IPs implement industry-standard specifications that are currently under review to remove exclusionary language.

1

Introduction to IC Validator Launch

This chapter gives an overview of how to run the ICV Launch graphical interface tool for IC Validator physical verification (DRC and LVS) separately from the debugging interface provided by the IC Validator VUE GUI.

The basics of the Launch tool are described in the following sections:

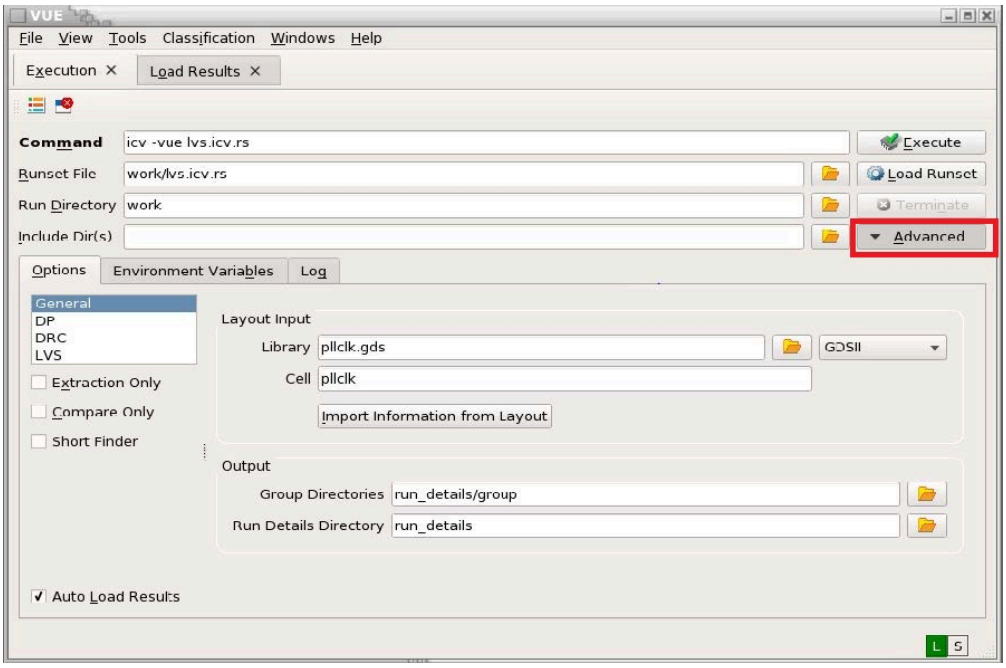
- Differences Between VUE Execution and ICV Launch
- ICV Launch Flow

Differences Between VUE Execution and ICV Launch

The VUE execution capabilities that help you define and run IC Validator jobs are adequate for typical job submission, as shown in Figure 1. The VUE execution tab provides four basic fields: Command-line input, Runset File, Run Directory, and Include Dir.

Click the Advanced button to change basic run settings for each job type in VUE.

Figure 1 VUE Execution Tab

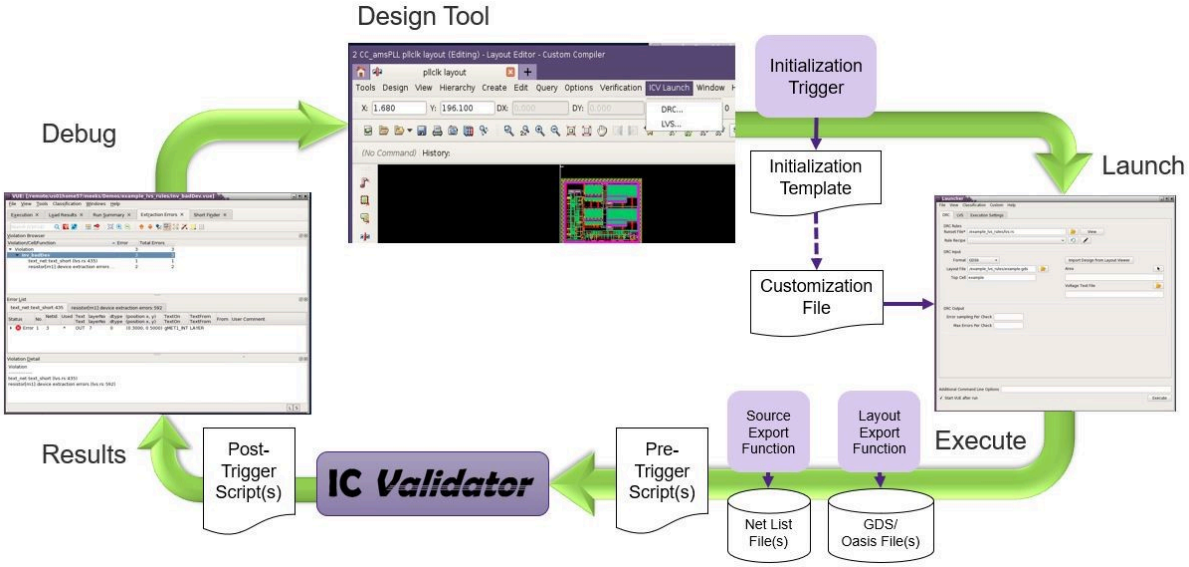


This use model is slightly nonintuitive and delays changing the settings until you click Advanced. There is also no way to customize the VUE interface or insert procedures into the run. These customization features are a regular consideration in a complete physical verification flow. The ICV Launch tool addresses these omissions.

ICV Launch Flow

The ICV Launch tool provides a complete integration flow with additional automation, as shown in Figure 2. You can create custom graphical widgets to configure your run as needed. You can also define scripts or utilities (triggers) inside of the design tool, or externally to automate stages before the IC Validator run.

Figure 2 ICV Launch Process Flow



The ICV Launch tool provides the flexibility and features to automate all IC Validator execution requirements in a common flow, within the design environment, as well as from the (batch) command line.

2

Using ICV Launch

This chapter covers the basics of the IC Validator Launch tool, such as how to configure the optional command-line options and adapt the ICV Launch tool behaviors in the following sections:

- ICV Launch Command-Line Options
- Licensing and Resource Requirements
- Job Submission From the Command Line
- Running ICV Launch From the Command Line in Batch Mode
- Running ICV Launch Within a Design Tool

ICV Launch Command-Line Options

The ICV Launch command-line syntax is

```
icv_launch [-usage | -h | -help | -]
   [-V | -Version] [-batch [file]]
   [-drc] [-lvs] [-load File]
   [-custom file]
```

The command-line options are described in Table 1.

Table 1 ICV Launch Command-Line Options

Option	Description
`-batch file`	Runs in batch mode. If file is set, then output the log to this file.
`-custom file`	Loads the customization file in JSON format
`-drc`	Create a new DRC run.
`-lvs layTool`	Creates a new LVS run
`-load templFile`	Automatically loads the template file upon startup.

Table 1 ICV Launch Command-Line Options (Continued)

Option	Description
`-usage | -h | -help`	Displays usage message and exits.
`-V | -Version`	Prints product version.

Licensing and Resource Requirements

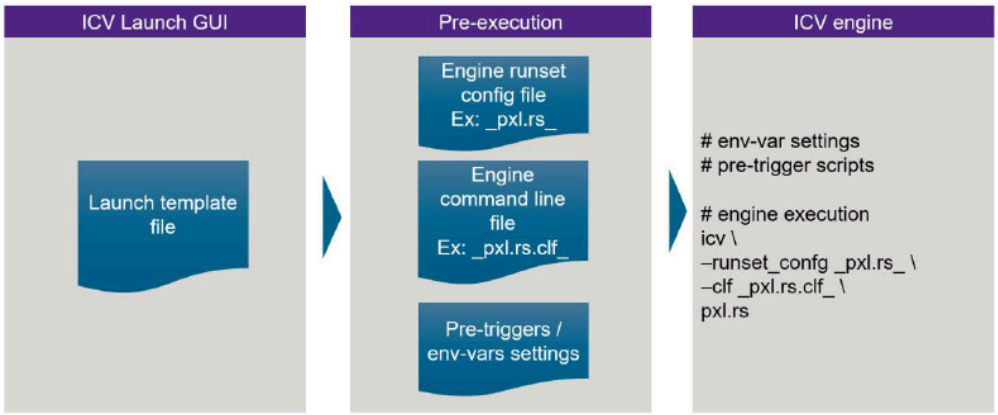
ICV Launch requires an Elite or Base license. ICV Launch does not check out these licenses. Instead, when you run IC Validator, the appropriate license for that run is used. If you have access to these license schemes, you have ICV Launch access.

Job Submission From the Command Line

The following files, which are generated by ICV Launch and defined by the IC Validator PXL runset on the command line start the IC Validator job.

```
icv -runset_config _pxl.rs_ -clf _pxl.rs.clf_ pxl.rs
```

Figure 3 ICV Launch to IC Validator Engine



Before executing the IC Validator run, ICV Launch takes the defined options and separates them into preprocessor directives, runset configuration settings, and any defined environment variables.

The command-line file, `_Runset File Name.clf_`, contains preprocessor directives and additional command-line option settings.

The runset configuration file, `_Runset File Name_` contains PXL settings that supersede the settings in the runset file.

Define these files on the command line to provide IC Validator with the options determined by ICV Launch. See Template, Runset Configurations, and Command-Line File Utilization for more information.

Running ICV Launch From the Command Line in Batch Mode

ICV Launch can be used from the shell command line as well as from within a design environment. From the command line, a preexisting template file must be provided to define the parameters for the IC Validator run. For example:

```
icv_launch -load template filename -batch
```

Where *Template Filename* is the name of your preexisting ICV Launch (JSON template file.

With this feature, you can rerun a previous job, or automate IC Validator jobs in a scripted manner.

Additionally, from within the Virtuoso environment, you can use a Skill function to run IC Validator jobs from a predefined template file in a Skill program or from the CIW window. For example:

```
icvLaunch(tmplFile, batchMode = t, logFile = "")
```

There are ways to use this function:

- Provide the *Template Filename*and let the log go to standard out.

  ```
  icvLaunch("Template Filename.tmpl")
  ```

- Provide the *Template Filename* and a log file name.

  ```
  icvLaunch("Template Filename.tmpl", t, "logFilename.log")
  ```

Running ICV Launch Within a Design Tool

This chapter provides basic information about how to interface with layout and schematic tools.

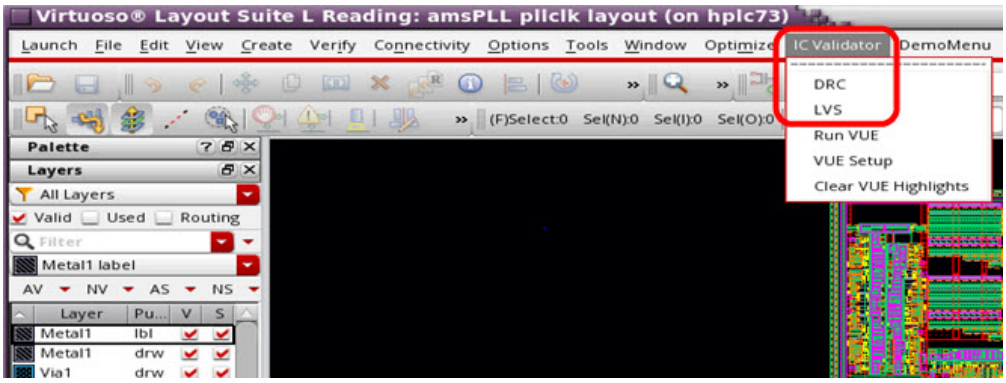
The supported layout interfaces are:

- Cadence Virtuoso
- Custom Compiler
- IC Validator WorkBench

Cadence Virtuoso

To run ICV Launch in the Cadence Virtuoso environment, run Virtuoso with IC Validator already installed. See Installing IC Validator Integration in Cadence Virtuoso for more information on loading the IC Validator integration. The IC Validator menu should be visible. Select this menu to see the DRC and LVS options used to open ICV Launch.

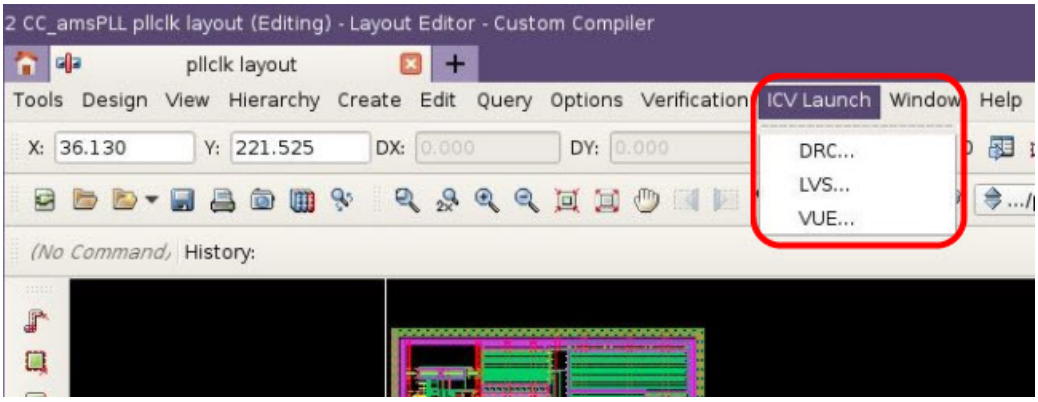
Figure 4 ICV Launch Within Cadence Virtuoso



Custom Compiler

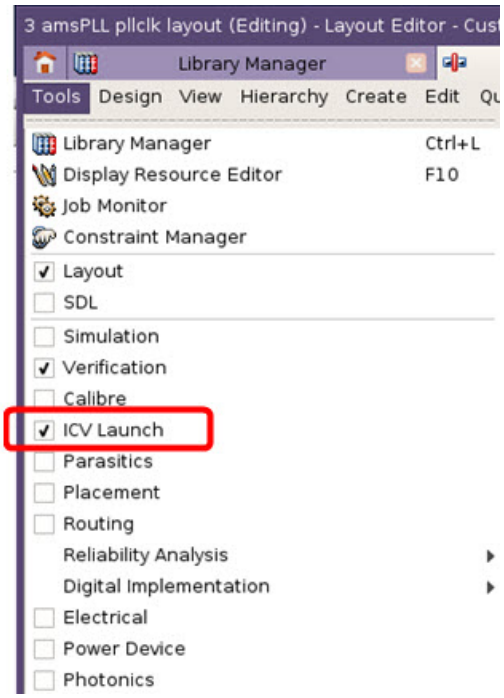
To run ICV Launch in the Custom Compiler environment, set the options for executing ICV Launch for DRC, LVS, or VUE, as shown in Figure 5.

Figure 5 Custom Compiler



If you do not see the ICV Launch menu in Custom Compiler, open the Tools menu and select the ICV Launch option, as shown in Figure 6.

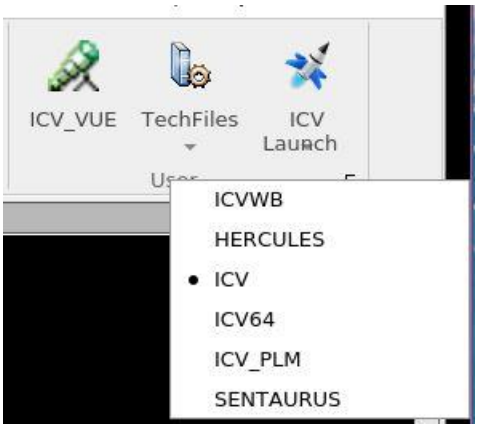
Figure 6 ICV Launch Option From the Tools Menu



IC Validator WorkBench

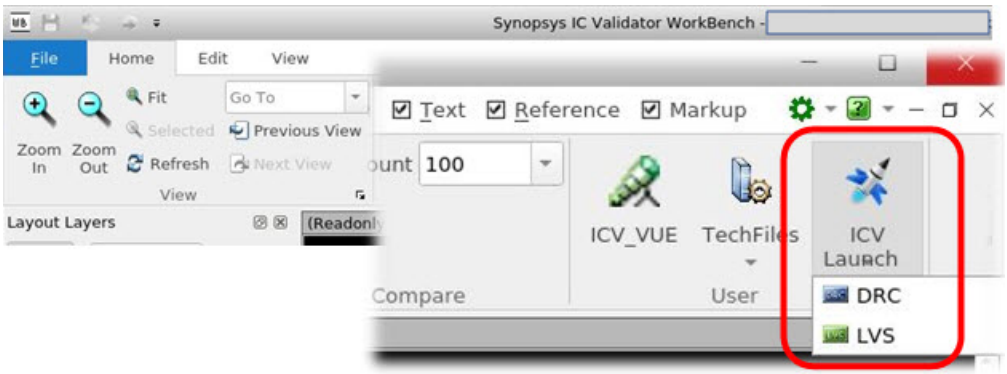
To run ICV Launch in the IC Validator WorkBench (ICVWB) environment, select ICV, as shown in Figure 7.

Figure 7 ICV WorkBench Environment



Now, you can execute ICV Launch for DRC or LVS, as shown in Figure 8.

Figure 8 ICV Launch for DRC or LVS



The ICV Workbench interface provides access to an additional feature in ICV Launch.

Click instead of the text labeling, and ICV Launch starts in default mode. This mode does not run the automation to preload the ICV Launch GUI. Instead, you start at a default landing page that allows you to choose the type of run you want to perform.

After you select the type of run you want, click Go to display the landing page for the selected run. This page does not have any of the entry fields pre-populated by the Initialization Trigger.

3

Running DRC Using ICV Launch

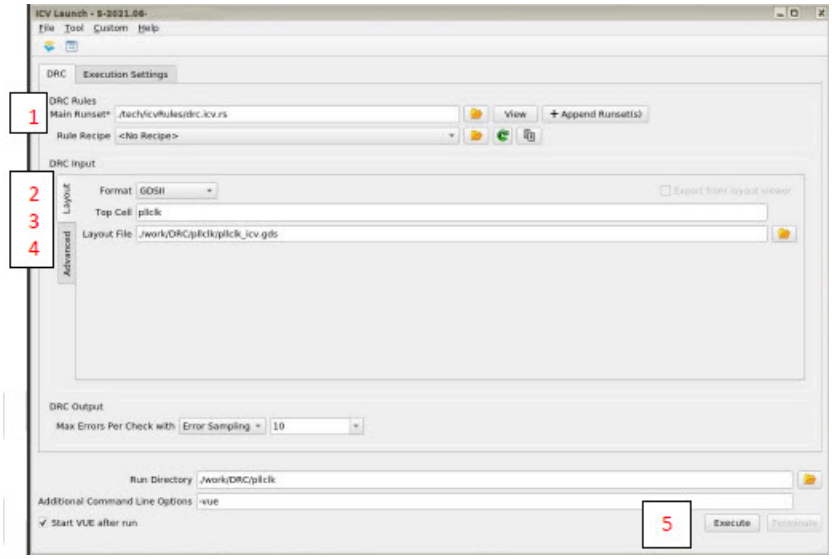
The two most common verification tasks performed on physical IC designs are design rule checking (DRC) and layout-versus-schematic (LVS). This chapter highlights DRC in the following sections:

- Automated Entry Validation
- Dynamic Runset Inclusion
- User Customization
- Using DRC Recipes
- Creating and Editing IC Validator Recipe Files

Running DRC or LVS from ICV Launch should behave similarly between different design tools, or from the command line from which you start ICV Launch. The only variations are due to the available features in each environment.

Enter the required fields for a DRC run, as shown in Figure 9.

Figure 9 ICV Launch DRC Dialog



Typically, there are four fields that must be populated:

1. Runset file
2. Format
3. Top Cell
4. Layout File

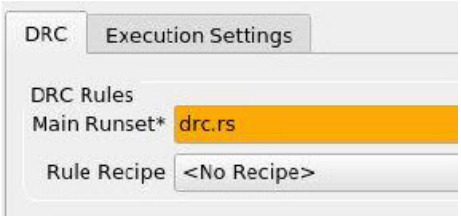
If you run within a design tool, the cell fields are automatically populated by the initialization function, or by the integration interface. Additionally, if you run ICV Launch from the (batch) command line, you can provide a template file to automate populating the required fields. Finally, click (5) Execute to initiate the IC Validator run.

If you manually enter information in the runset file, you can access a graphical file browser by clicking . Additionally, you can view or edit the defined runset file in an external text window (GVIM) by clicking View.

Automated Entry Validation

If you manually enter file information into a field, the entry background color automatically changes to an orange highlight until the file name is validated, as shown in Figure 10.

Figure 10 File Validation



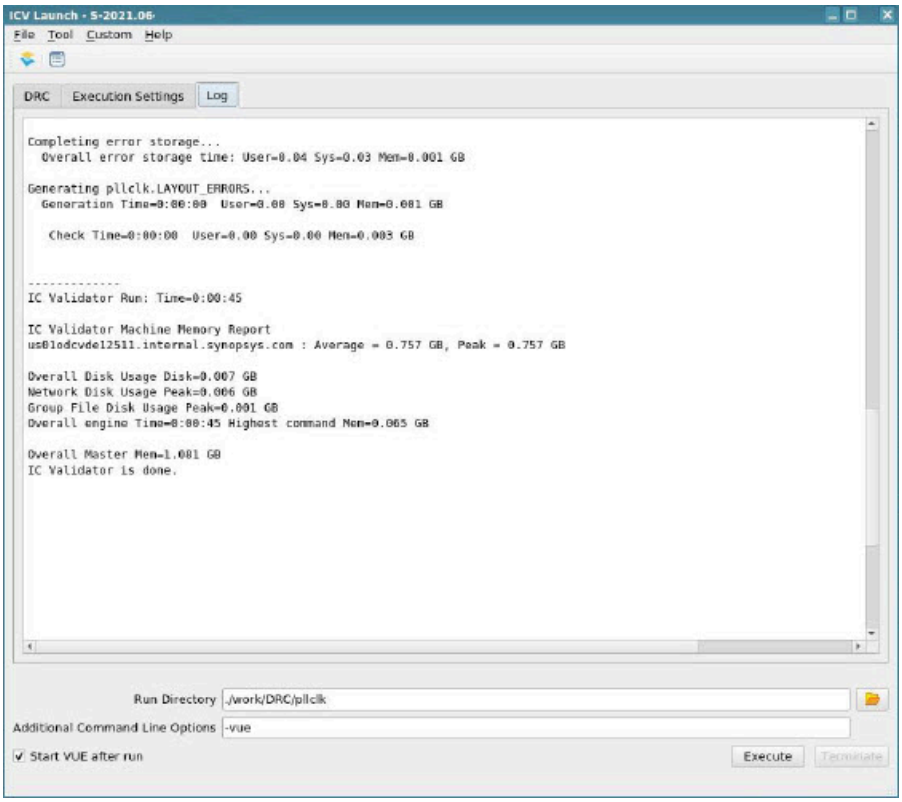
Additionally, in all of the ICV Launch entries that provide a file name, environment variables are allowed as part of that file path. For example:

The `DEMO_TECH_DIR` environment variable must exist to define the prefix path to this directory.

After you enter all of the settings, click Execute to start the process. If selected, the layout data is exported from the design tool, the user-defined pretriggers are executed, and the IC Validator tool starts.

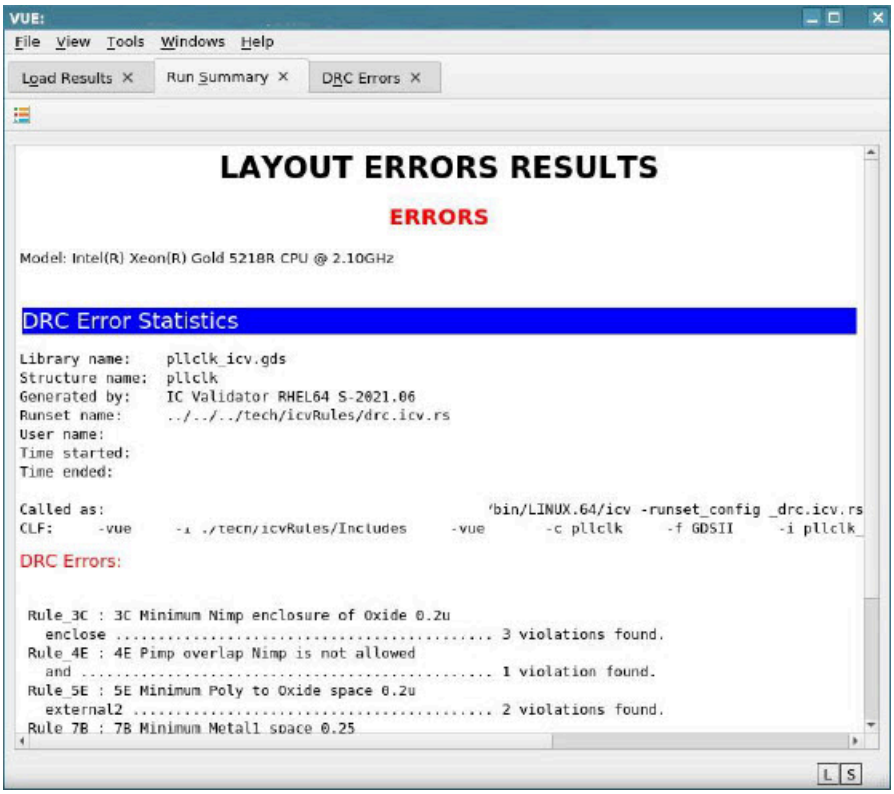
At the start of the ICV run, the Log tab appears in the ICV Launch window. You can interactively monitor the job, as the standard output from the run is dynamically scrolled in the Log window.

Figure 11 ICV Launch Log Window



At the completion of the DRC run, if optionally selected, VUE opens and the results are loaded and ready for you to begin debugging within the design tool.

Figure 12 VUE With DRC Results

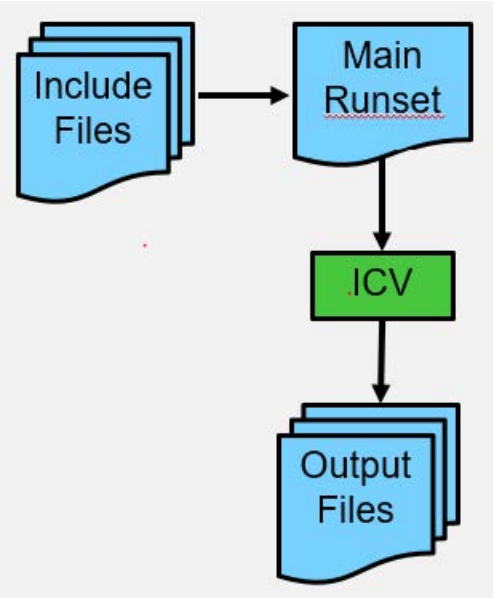


Dynamic Runset Inclusion

ICV Launch provides the ability to supplement your main runset with additional or modified settings and definitions. You can define additional runset files to be included in your main runset by clicking Append Runsets.

Because of the structured nature of the PXL language used in IC Validator, a new set of syntax definitions are required to perform this feature and produce valid final runsets.

Figure 13 Runset Inclusion



By using the preprocessing `#pragma` capabilities, you can define the insertion points within the main runset for the new settings in the included files. In the following DRC runset example, the insertion point for any new ASSIGN statement is at the patch_assigns location (highlighted in yellow) in the runset file. Check selection options are inserted in the patch_unselects location (highlighted in green).

Any new check definitions are inserted at the location of the patch_checks location (highlighted in grey). This allows for the automated combination of files, while also maintaining the proper syntactical structure required by the PXL language.

Figure 14 Defining Insertion Points

```
/ Header

// -- Content --

// Assign Section

L1 = assign({ {1} });

#pragma ICV_patch_insert patch_assigns

#pragma ICV_patch_insert patch_unselects

// Check Definitions

Chk_L1_Width @= { @ "Chk_L1_Width : Minimum L1 width";
  internal( L1, < 0.3, ... );
}

#pragma ICV_patch_insert patch_checks

// End of File
```

Additionally, the include files require some new definitions to establish the PXL statement to be inserted at the points defined in the main input runset file. In the Include file, as shown in Figure 14, the highlighted colors indicate the sections that are inserted. The begin and end definitions enclose the PXL statements included in that section definition.

Figure 15 shows a simple DRC example, but other IC Validator applications such as LVS and PERC might also use this mechanism. You can have as many user-defined section names (identifiers) as required to modularize your IC Validator runsets. You must ensure that the definitions resolve into legal PXL and are able to be compiled. Otherwise, the IC Validator tool exits with an error message.

Figure 15 DRC Example

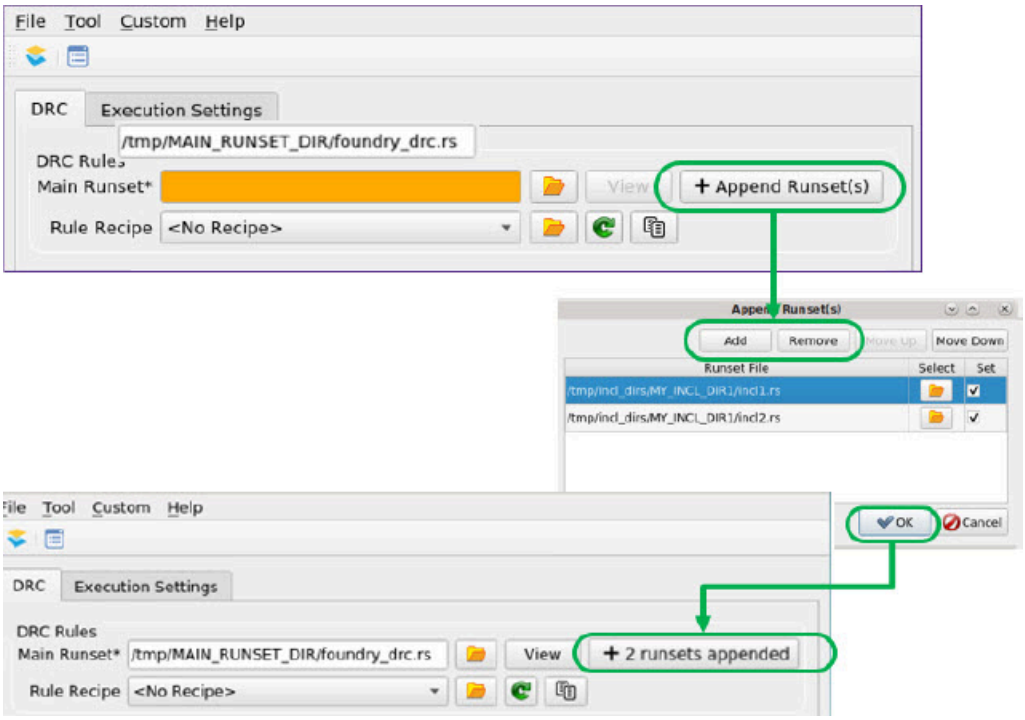
```
// This is an example Include file

#pragma ICV patch_assigns begin

L2   = assign({ { 2 } });

violation_name_select( ( "" ) );

#pragma ICV patch_assigns end

#pragma ICV patch_unselects begin

violation_name_select( ( "Chk_L2_Width" ) );

#pragma ICV patch_unselects end

#pragma ICV patch_checks begin

// L2 Checks

Chk_L2_Width @= { @ "Chk_L2_Width : Minimum L2 width";
  internal1( L2, < 0.4, e...);
}

#pragma ICV patch_checks end

// End of File
```

Example 1

```
// This is an example Include file

#pragma ICV patch_assigns begin

L2= assign({ {2} });
```

Click Append Runsets to launch a dialog box that is used to define the files to be included. You must define the main runset in the original entry field.

Figure 16 Append Runsets



From the Append Runsets dialog box, you can define and select the PXL runset files and their order of inclusion. The check box to the right of this table allows you to predefine the files and choose the ones used in this IC Validator run.

After you set the include files, click OK to set the definitions for this IC Validator run. You see an indicator in the +Append Runset button of how many include files were set to be used. By defining additional include files, ICV Launch automatically generates a new control file for defining these definitions. This control file is passed to IC Validator and the process runs with the new user-defined options in place.

Example 2 Include Files

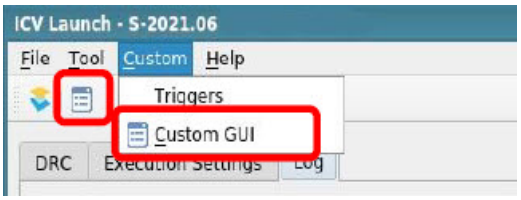
```
// Header

// The Main Runset File

#include "test.drc.rs"

// Supplemental Include Runset Files

#include "incl1.res"

#include "incl2.res"

#include "incl3.res"
```

```
// End of File
```

User Customization

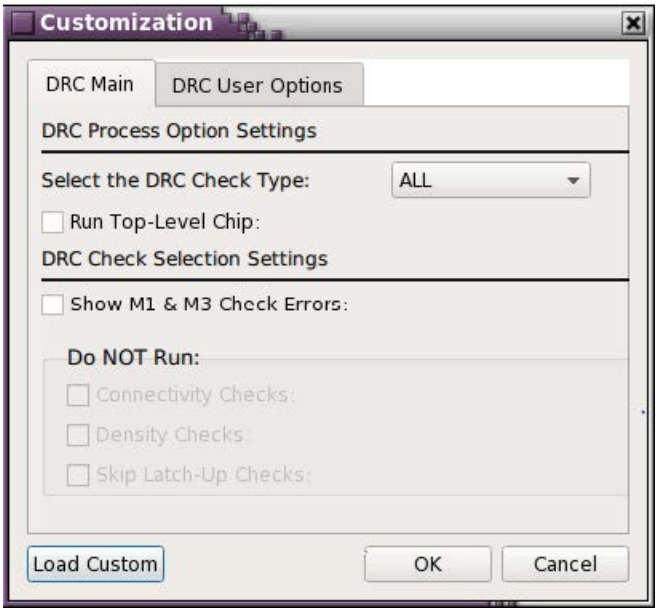
In addition to the entries built into the ICV Launch GUI, additional graphical entries can be defined to allow you the ability to control the IC Validator execution. The additional GUI widgets are defined in a standard external text file in JSON[1] format. See GUI Customization for a complete description of the user-customization definition file.

Figure 17 User Customization



To manually load a customization file, choose ICV Launch > Custom > Custom GUI, or click . From the command line, you can specify the customization file by setting the `-custom filename` command-line option.

Figure 18 Customization Dialog



1. *JSON is an acronym for JavaScript Object Notation format.*

If a customization file is specified, the Customization window automatically appears upon startup, when called by , or from the Custom GUI menu item.

The Customization window supports many types of widgets, such as check boxes, entry fields, and enumerated lists. Another presentation aide is the definition of multiple tabs to better organize the widget features.

Customization widgets might also be linked into groups and controlled by other widgets such that the user experience is dynamic and easy to use.

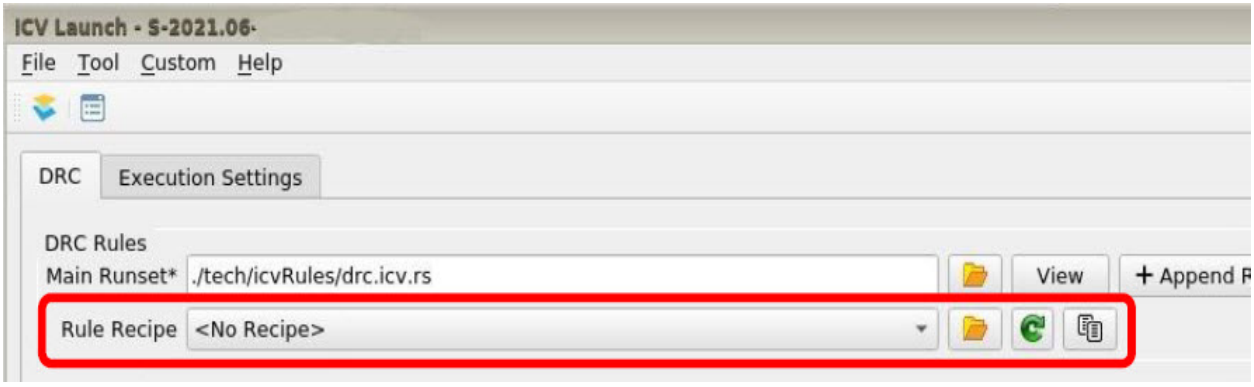
If the number of widgets on a customization tab exceeds the size of the Customization window, scroll bars automatically appear to allow you to scroll to the areas of interest. See GUI Customization for more information.

Figure 19 Customization Dialog (Continued)

Using DRC Recipes

Recipes are a mechanism by which you can selectively choose what checks you want to execute without the need to edit or modify the runset. The Rule Recipe widget is a cyclical entry to allow you to easily select one of the predefined recipes. By not selecting a recipe, or by selecting <No Recipe>, the default is to run all of the checks defined in the runset.

Figure 20 Rule Recipe



Click to load a predefined recipe file. The recipe file is an external text file written in JSON[2] format. ICV Launch uses the same recipe definition files as ICV Live. Recipe files can also be defined in the initialization template file to be automatically loaded upon ICV Launch startup.

Click to reload a Recipe file and the updated recipe definitions.

Click to create or modify IC Validator recipe files and definitions. Recipes are the best solution to making a collection of checks easier to isolate and run.

Example 3 IC Validator DRC Runset

```
{
    "majorVersion": 1,
    "minorVersion": 0,
    "recipeDescription": "",
    "recipes": [
        {
            "recipeName": "Metal_1_Only",
            "selectedRulesByComment": [
            ],
            "selectedRulesByName": [
                {
                    "delim": " : ",
                    "name": "Rule_7A"
                },
                {
                    "delim": " : ",
                    "name": "Rule_7B"
                },
                {
                    "delim": " : ",
                    "name": "Rule_7C"
                }
            ]
```

2. *JSON is an acronym for JavaScript Object Notation format.*

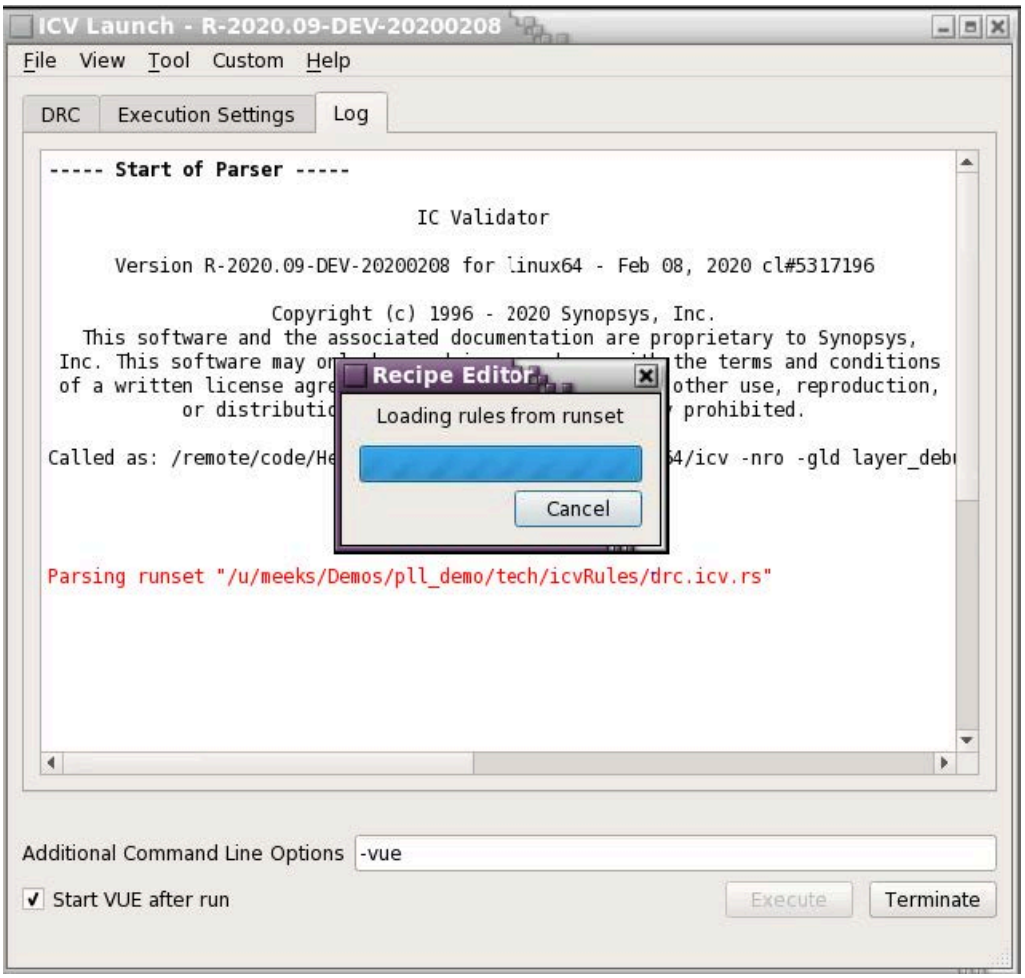
```
                }
            ]
        }
```

Creating and Editing IC Validator Recipe Files

Upon first use, the Edit Recipe feature defines the parsed runset file to determine the defined checks.

This information is helpful when creating a recipe definition. You can cancel this operation while the runset loads, as shown in Figure 21.

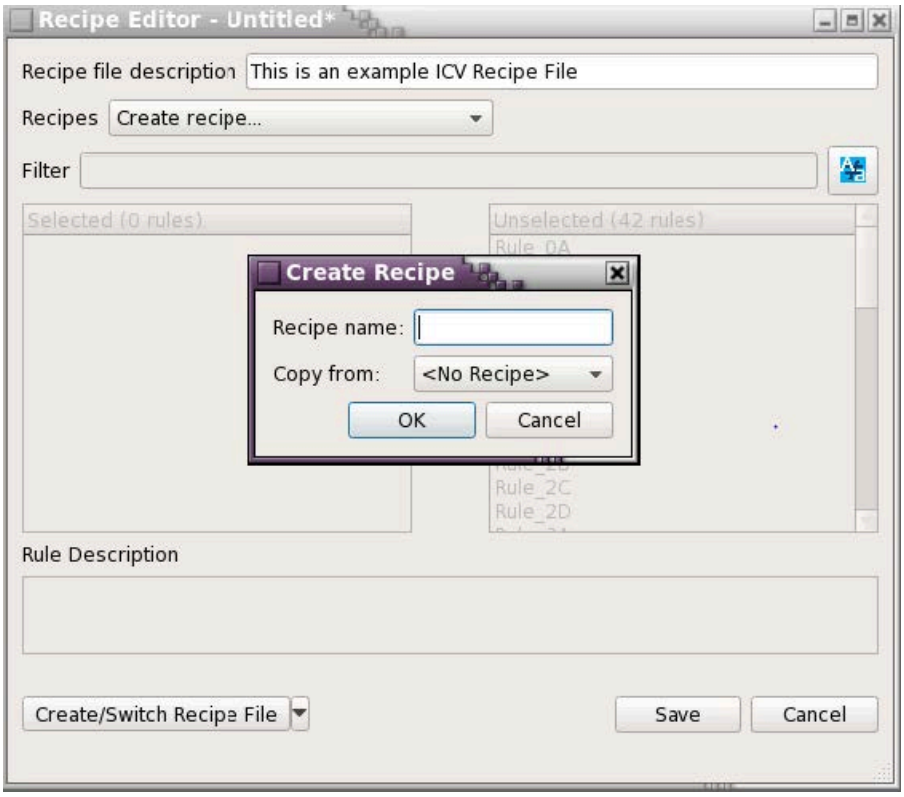
Figure 21 Cancel Recipe Editor



When the runset is read successfully, you see the ICV Recipe Editor. Enter a general description for this recipe file in the Recipe file description field. From the Create Recipe

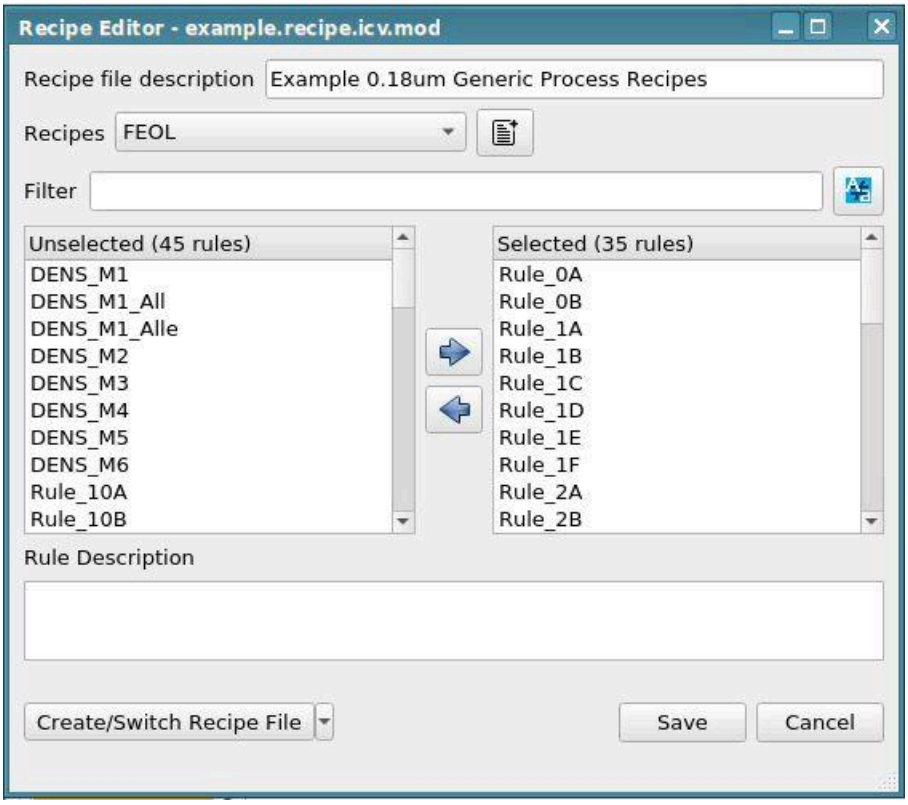
dialog, provide a name for the recipe you want to create. You can also copy from an existing recipe definition. When you complete all of the fields needed for your recipe definition, click OK, as shown in Figure 22.

Figure 22 Create Recipe



Now, you can select the checks from this recipe. A list of all checks in the runset file are listed in the Unselected window. From here, you can choose one or more checks to define in this recipe, as shown in Figure 23.

Figure 23 Select Recipe Checks



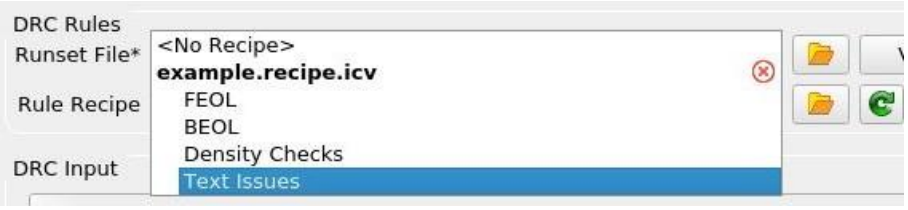
After you choose your checks, click to move these checks to the Selected entry window.

To remove a selected check back to the Unselected window, click .

Use the Filter field to reduce the displayed checks to match your filter pattern. Click to toggle case sensitivity for the filter pattern. After you select the checks for this recipe, click Save to save the recipe definition in the selected IC Validator Recipe (JSON) file.

After you save the recipe definitions, the defined recipe appears in the Rule Recipe entry. When you run IC Validator with this option selected, only those selected checks are run in the IC Validator DRC run. This provides you with more flexibility to run only what you need for this job execution.

Figure 24 Rule Recipe Entry



4

Running LVS Using ICV Launch

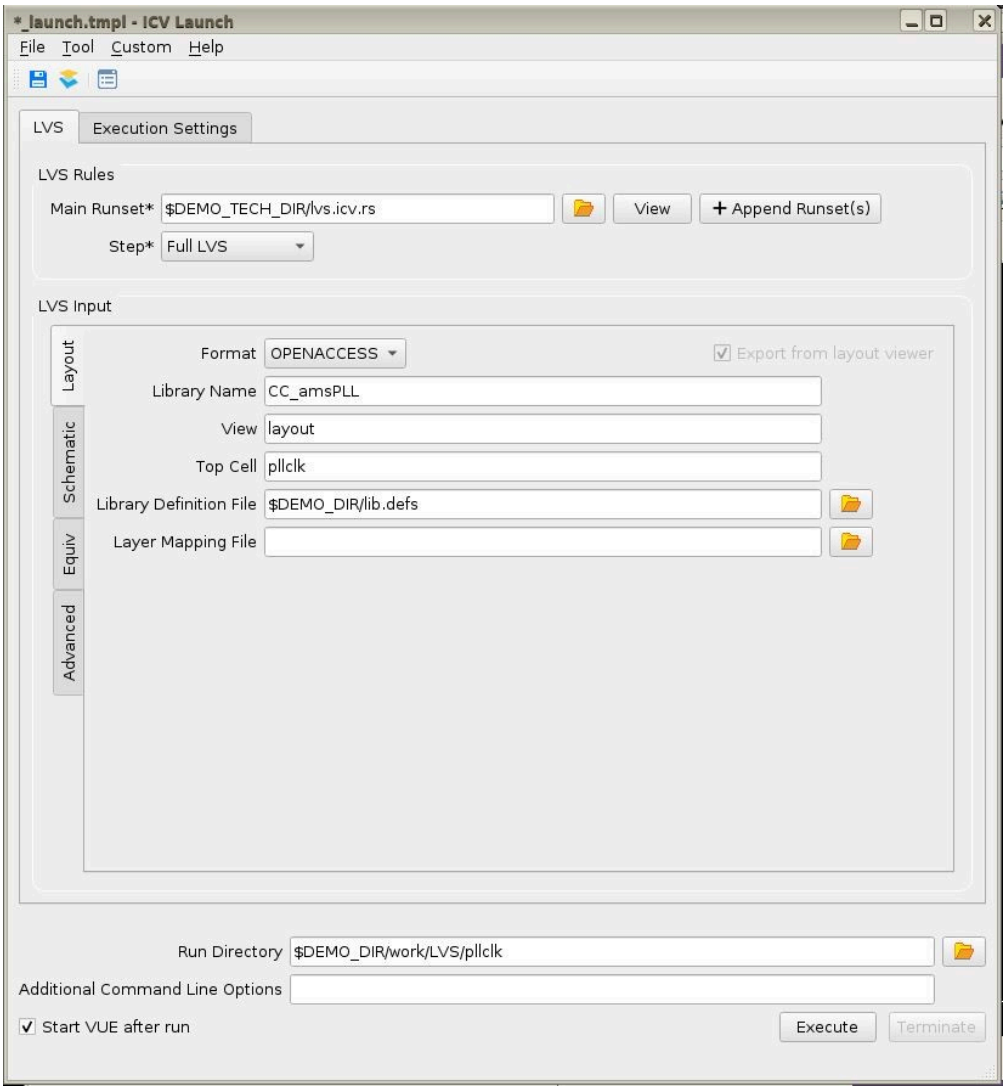
The two most common verification tasks performed on physical IC designs are design rule checking (DRC) and layout-versus-schematic (LVS). This chapter highlights LVS in the following sections:

Running DRC or LVS from ICV Launch should behave similarly between different design tools, or from the command line from which you start ICV Launch. The only variations are due to the available features in each environment.

The Layout Input entries for LVS are very much the same as they are for DRC. The additional tabs (along the left side) for the Schematic and Equivalence Settings add features pertinent to LVS processing, as shown in Figure 25.

Feedback

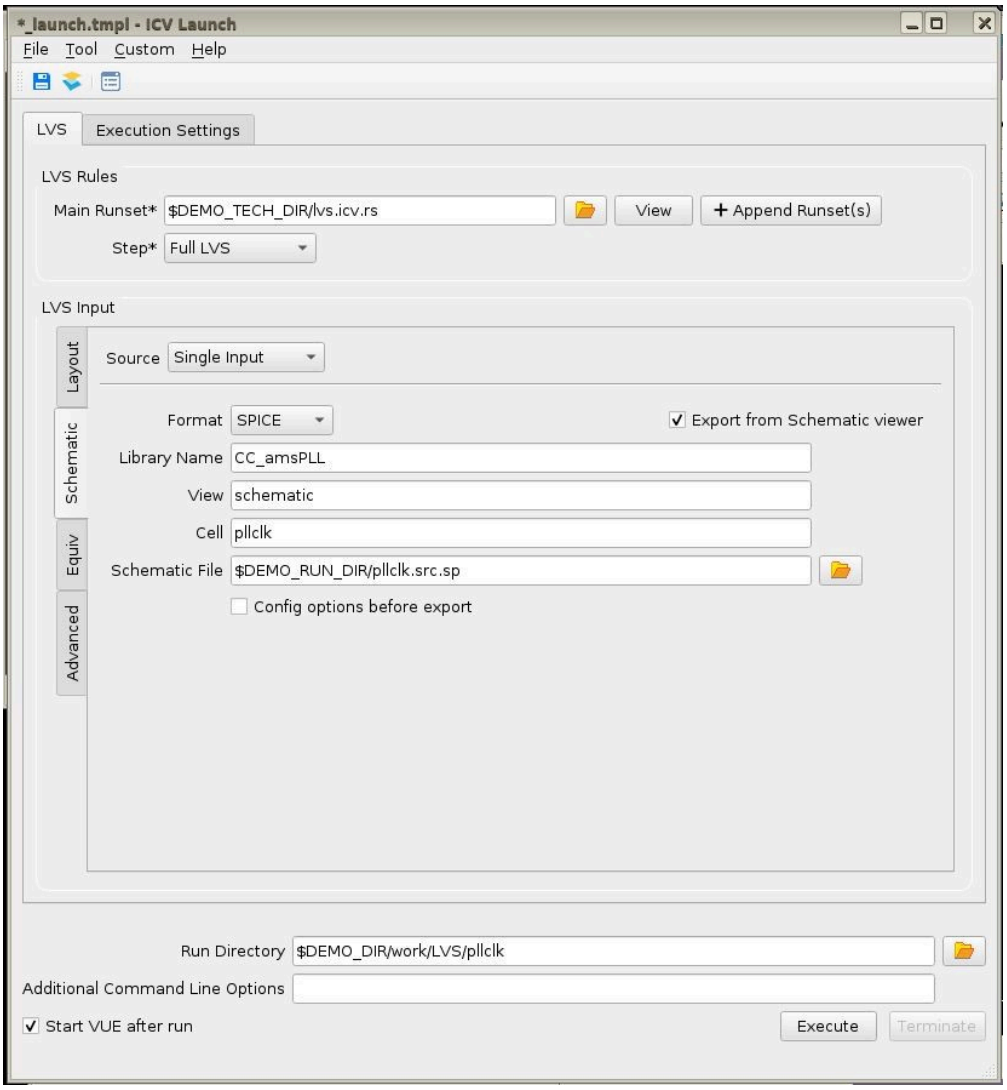
Figure 25 LVS Tab



For the Schematic input, there are two modes to meet your requirements. The most typical Source mode is Single Input, as shown in Figure 26. This mode allows you to point to a preexisting single input netlist file or export the schematic netlist from your design environment. See Export Triggers for more information on setting export options.

Feedback

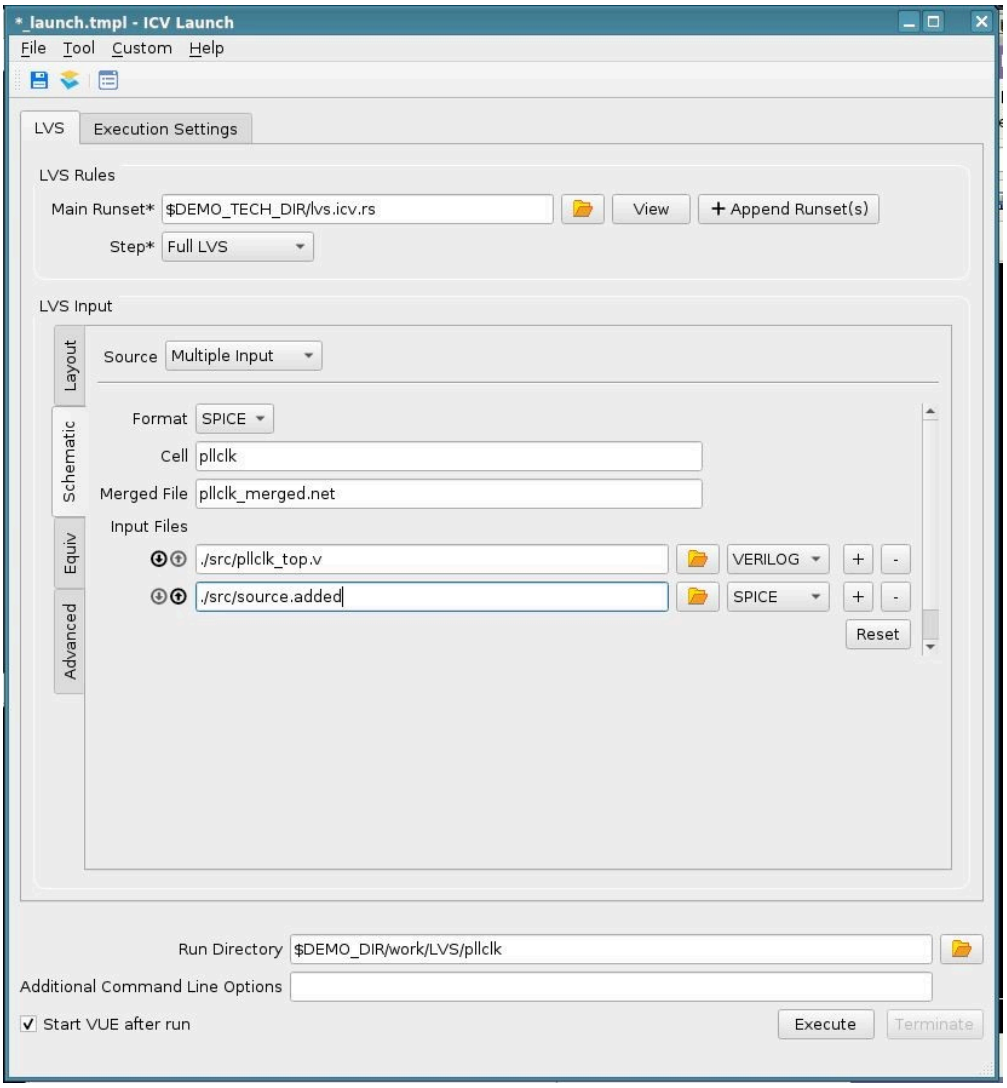
Figure 26 LVS Tab Single Input



To automate the merging of multiple preexisting netlist files in different formats such as Verilog and SPICE, use the Multiple Input mode, as shown in Figure 27.

Feedback

Figure 27 LVS Tab Multiple Input

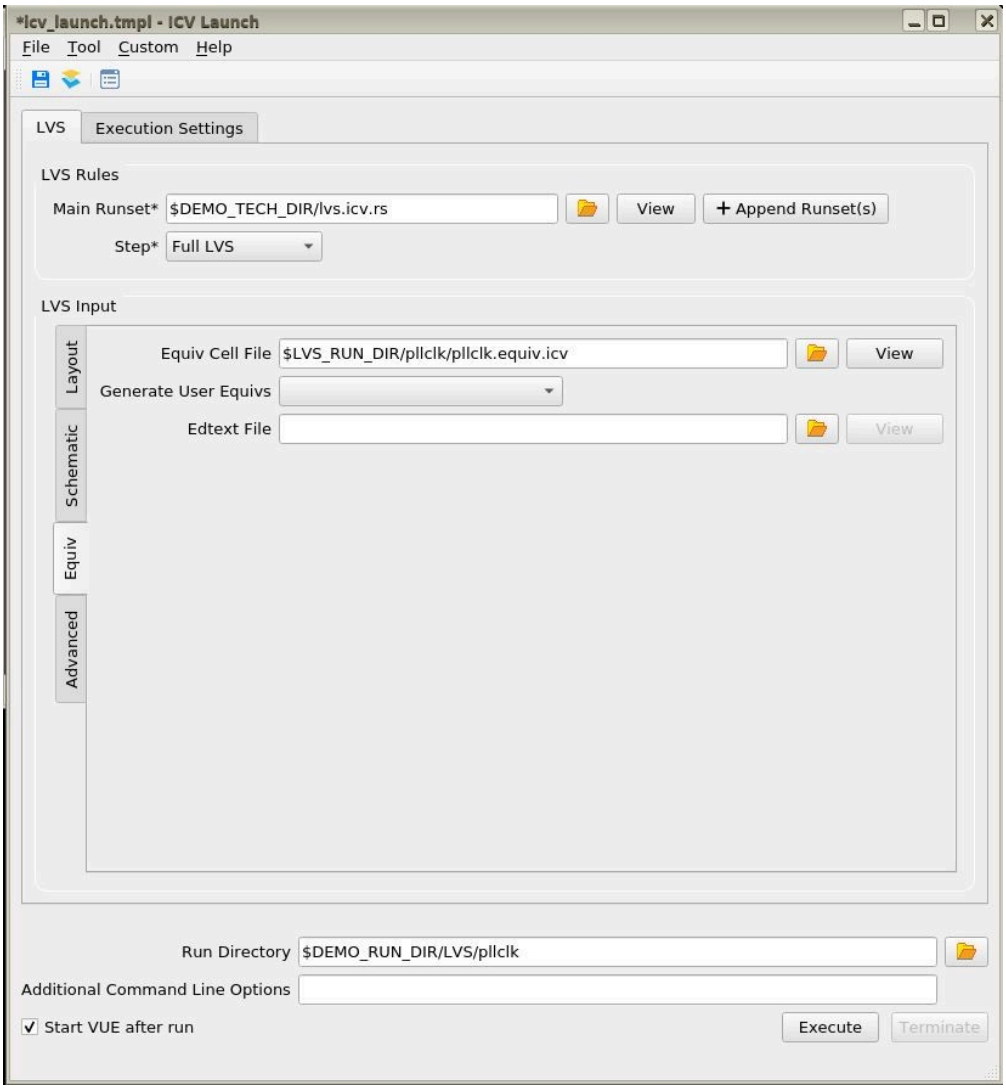


The Multiple Input mode uses the NetTran utility. NetTran can accept multiple netlist formats, such as input and merge, and produce a new combined netlist output.

From the Equiv tab, as shown in Figure 28, you can define an equivalence file, set the options for generating system equivs, and define an Edtext file to specify text elements externally from the layout database.

Feedback

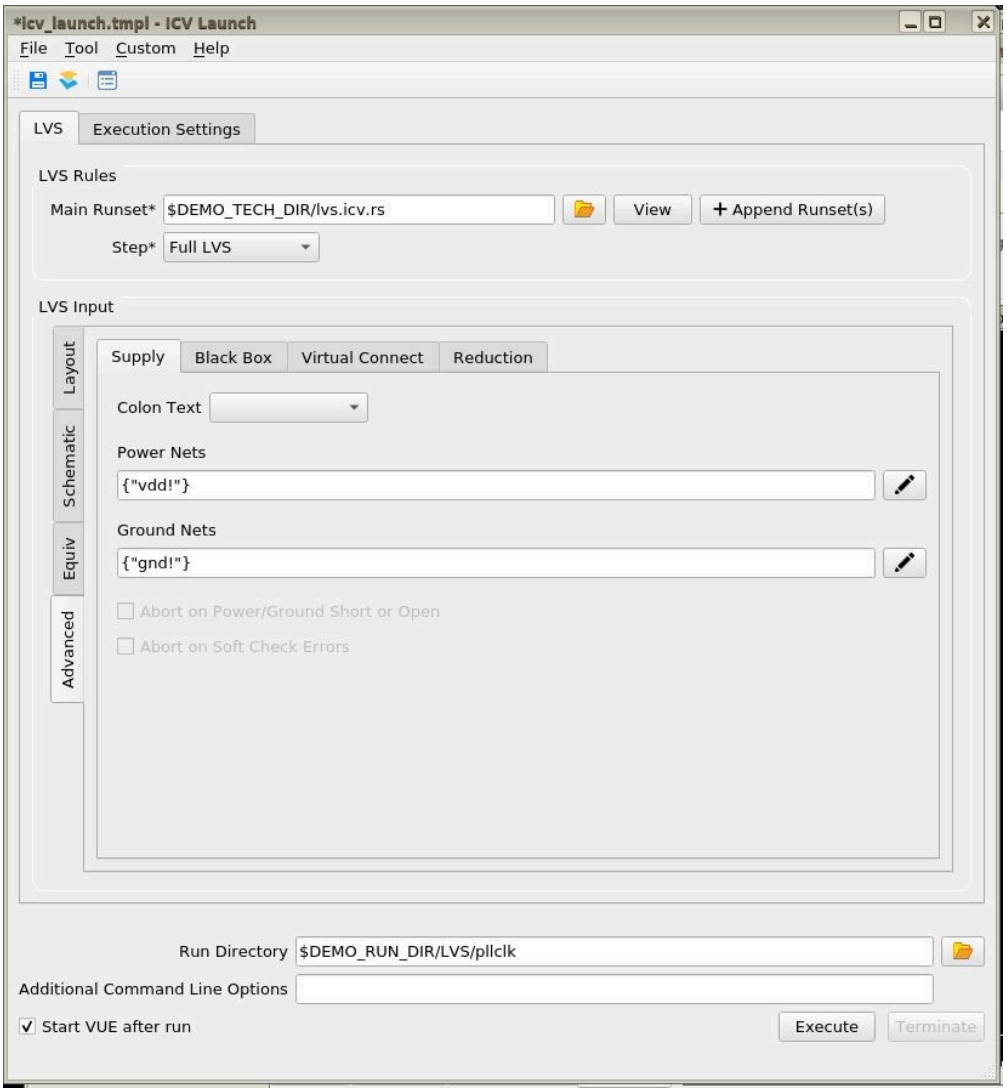
Figure 28 LVS Input Equiv Tab



From the Advanced Tab, there are additional tabs for setting supply names, black-box, virtual connections, and device reduction definitions, as shown in Figure 29.

Feedback

Figure 29 LVS Input Advanced Tab



5

Template, Runset Configurations, and Command-Line File Utilization

This chapter contains the following topics:

- Template File
- Keywords for Built-in ICV Launch GUI Entries
- Runset Configuration File
- Command-Line File

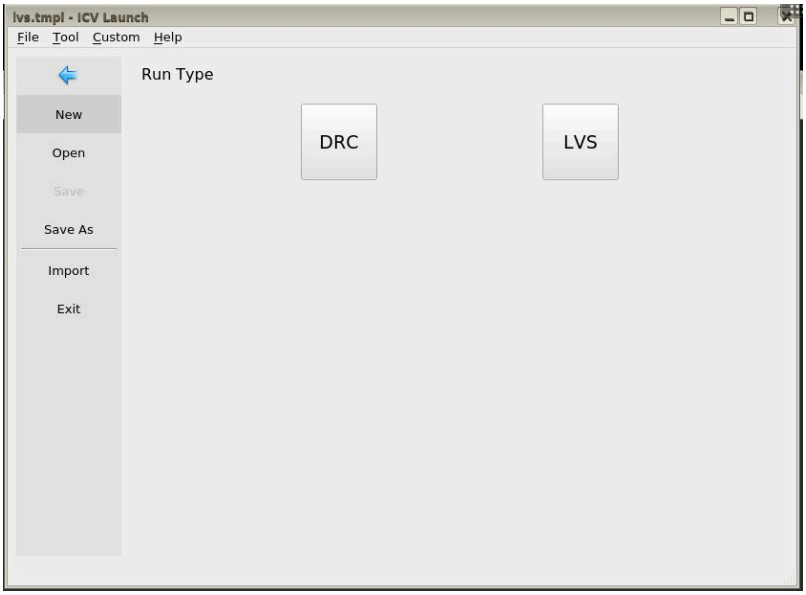
The template file does not directly impact the IC Validator engine, but is meant to facilitate the automated population of the ICV Launch widget values. It also serves as an intermediate step to the creation of the runset configuration and command-line files.

The runset configuration file and command-line files are the mechanism through which ICV Launch passes user settings to the IC Validator engine. Upon execution, these files are generated by ICV Launch before a call to the IC Validator engine. The runset configuration file follows the PXL syntax format for passing certain user settings to the IC Validator engine, while the CLF file contains all of the command-line options to send to IC Validator.

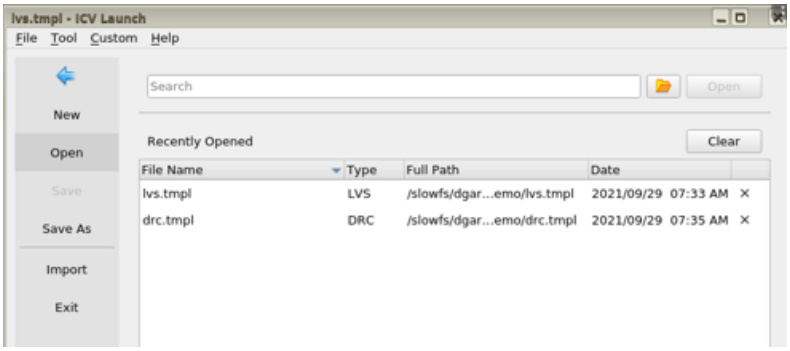
Template File

You can save the values for all ICV Launch fields to a JSON-formatted single file using a template file.

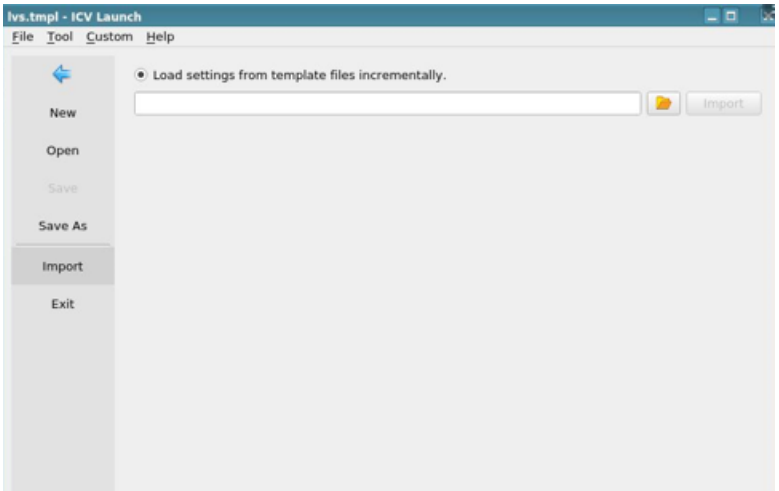
From the File menu, click New to clear all of the entries and start a new definition.



Click Open to load a previously saved Template file after clearing out any current settings. Click Save or Save as to save a JSON-formatted template file for later use.



Select “Load settings from template files incrementally” if you do not want to clear the previous settings. Click the Import button to apply the settings from the selected file to overwrite the defined settings. This enables you to selectively insert settings and overlay multiple files.



Click Exit to close the ICV Launch tool.

You can use previously saved template files for subsequent IC Validator runs choosing File > Load template file.

Additionally, use the `-load` command-line option to directly load a template file, which populates the ICV Launch GUI widgets with all of its settings.

```
% % icv_launch -load mySettings.tmpl
```

The ICV Launch Template file is a JSON-formatted file. Values for all fields are saved to the file in a “fieldname”: “fieldValue” format. The following example of a template file is saved from the ICV Launch GUI.

```
{
    "launch": {
        "run1": {
            "additionalCommandLineOptions": "-vue",
            "drcFormat": "GDSII",
            "drcLayoutFile": "user_errors.gds",
            "drcMaxErrors": "10",
            "drcRunsetFile": " user.rs",
            "drcTopCell": "top",
            "runType": "DRC",
            "shellVariable": [
                {
                    "newValue": "/global/snps_apps/icv_2019.12-BETA",
                    "set": true,
                    "varName": "ICV_HOME_DIR"
                }
            ]
        }
    }
}
```

Keywords for Built-in ICV Launch GUI Entries

General Settings:

#	Field Name	Type	Runset config / CLF	Mapping
1	additionalCommandLineOptions	string		The additional ICV command options
2	startVueAfterRun	bool		If checked, then open VUE after ICV run is successfully completed.
3	runType	string		Indicate the type of ICV run (DRC/LVS)
4	runDirectory	string		-rd
5	triggers	JsonArrary		The settings of pre-trigger

Execution Settings:

#	Field Name	Type	Runset config / CLF	Mapping
1	preprocessorDir	String	JsonArray	-I
2	shellVariable	String	JsonArray	Environment variables
3	dpRunOnType	String		To decide how to run icv
4	dpOnLocalHostCores	integer	CLF	-host_init
5	dpOnLocalHostMemory	integer	CLF	-host_memory
6	dpOnReservedHosts	String	CLF/JsonArray	-host_init
7	dpOnRsvHostsMemory	integer	CLF	-host_memory
8	dpOnRsvHostsLoginMethod	String	CLF	-host_login
9	dpSgeNumberOfHosts	integer		--hosts (for icv_batch)
10	dpSgeCoresPerHost	integer		--cores-per-host (for icv_batch)
11	dpSgeAddHost	Boolean		--add-host (for icv_batch)
12	dpSgeUniqueHosts	Boolean		--unique-hosts (for icv_batch)
13	dpSgeSubmitCmd	String		--submit-cmd (for icv_batch)
14	dpLsfNumberOfHosts	integer		--hosts (for icv_batch)

#	Field Name	Type	Runset config / CLF	Mapping
15	dpLsfCoresPerHost	integer		--cores-per-host (for icv_batch)
16	dpLsfAddHost	Boolean		--add-host (for icv_batch)
17	dpLsfUniqueHosts	Boolean		--unique-hosts (for icv_batch)
18	dpLsfSubmitCmd	String		--submit-cmd (for icv_batch)
19	dpOtherCustomCommand	String		User provided executable
20	dpOtherNumberOfHosts	integer		For meta-character %n
21	dpOtherCoresPerHost	integer		For meta-character %cph
22	dpOtherMemory	integer		For meta-character %m

Importing Versus Loading Template Files

Importing versus loading template files require different steps. When you load a file, all of the option setting values are cleared before inserting the values defined in the template. When you import a file, the defined values are added. This distinction allows for a hierarchy of template files.

Set the `LAUNCH_INIT_TEMPLATE_FILES` environment variable to define a sequence of colon-separated template file names to be imported in the defined order (from left to right). For example,

```
setenv LAUNCH_INIT_TEMPLATE_FILES "file1.tmpl:file2.tmpl:file3.tmpl"
```

In this example, file1.tmpl, then file2.tmpl, and finally, file3.tmpl are imported into the GUI with the last definition setting the value.

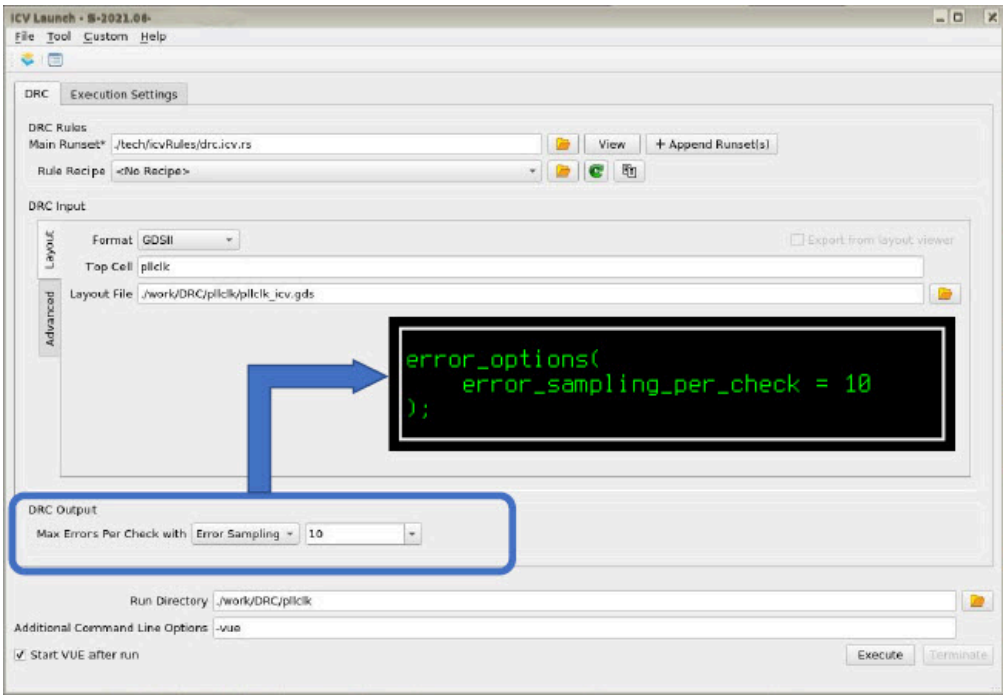
Runset Configuration File

The PXL-formatted runset configuration file overrides functions that exist in the main runset or adds new functions that are not defined in the file. This file supports all IC Validator settings contained within the <Type>_option sections of the PXL runset and LVS compare section.

The name for the runset configuration file is dependent upon the name of the runset. It is derived by appending the underscore character '_' on either end of the name of the runset.

Figure 30 demonstrates how the runset configuration file is used.

Figure 30 Runset Configuration File



The Max Errors Per Check with setting is translated to the `error_options()` function in the runset configuration file. This function takes precedence over any other occurrences of the `error_options()` functions in the runset file, user.rs. Similar PXL functions are generated where applicable and are added to the runset configuration file.

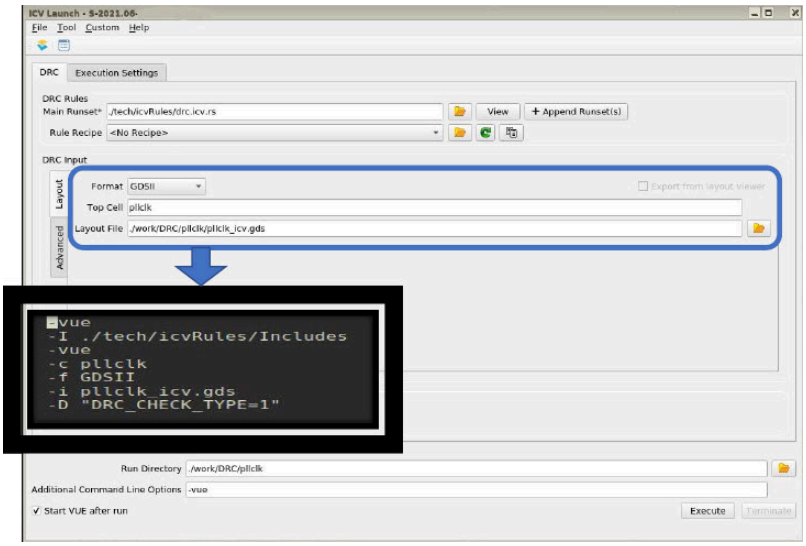
Command-Line File

The CLF file functionality is used by IC Validator to convey command-line options and definitions. ICV Launch leverages this file to provide directives to the PXL preprocessor. These definitions might be provided by the customization widget DEFINE definitions, or from the Additional Command Line Options entry settings.

The name for the CLF file is also dependent on the name of the runset. It is derived by appending the underscore character '_' on either end of the name of the runset, and by changing the existing extension of the runset file to '.clf'.

Figure 31 shows how the CLF file is used.

Figure 31 Command-Line File



Upon execution, the runset configuration and CLF files are both generated and called by IC Validator using the `-runset_config` and `-clf` command-line options, respectively.

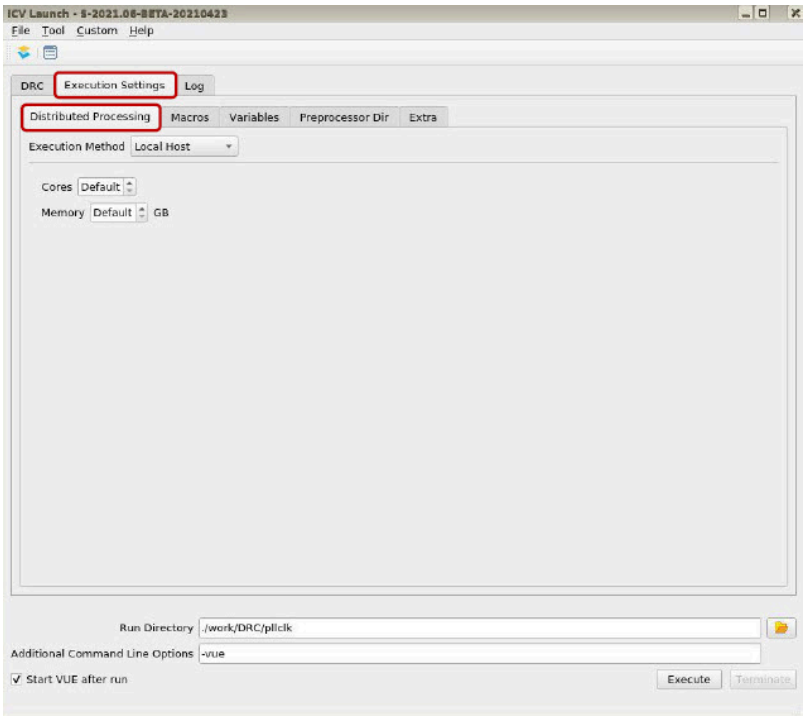
6

Configuring Distributed Processing Jobs Using IC Validator

The ICV Launch GUI provides you with multiple ways of setting up distributed jobs. You can run your IC Validator jobs on a local host (default), reserved hosts, or several cluster environments. Among the cluster environments, ICV Launch provides a GUI interface for commonly used LSF and SGE environments. In addition to these distributed modes, you can also setup a job on other cluster environments.

Choose Execution Settings > Distributed Processing to set up distributed jobs.

Figure 32 Configuring a Distributed Processing Job



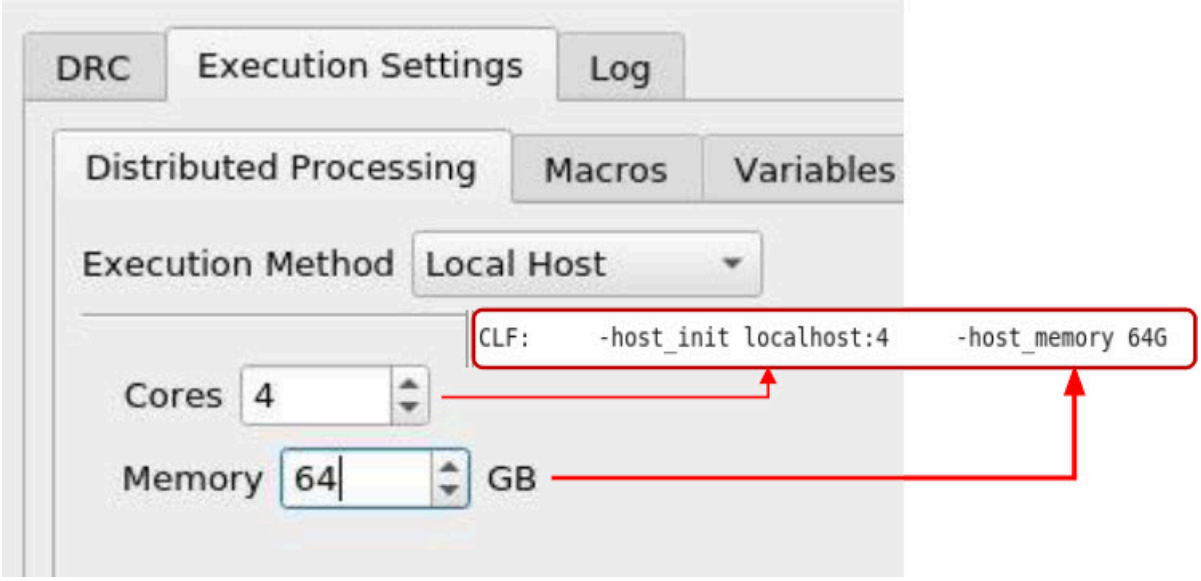
The Distributed Processing tab provides the following execution methods:

- Local Host
- Reserved Hosts

- LSF and SGE
- Other

Local Host

To set up ICV Launch jobs on the same execution host where you are running, use the Local Host execution method. With this method, you can set the number of cores to use (`-host_init`) or set the maximum amount of host memory (`-host_memory`).

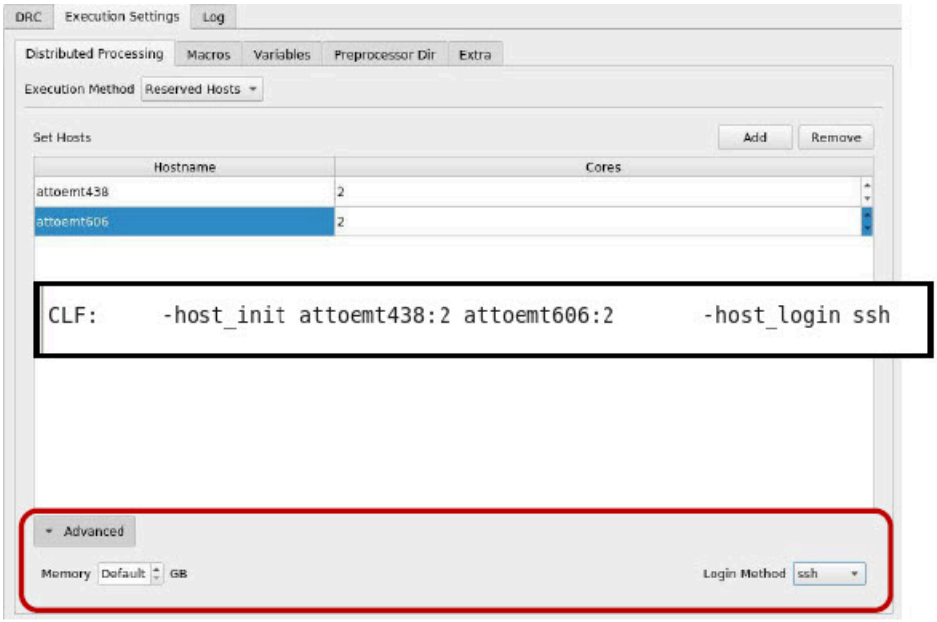


Because these settings can be controlled with command-line options, they are added to the CLF file.

Reserved Hosts

To set up distributed jobs on known hosts, use the Reserved Hosts method. You can specify the name of the host and the number of cores on that host to use for the ICV Launch run.

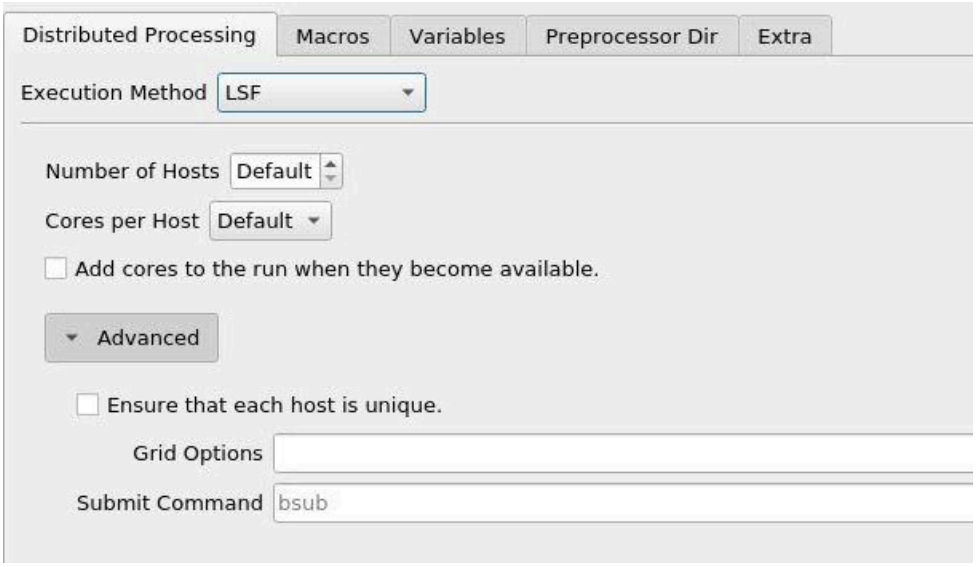
Similar to the Local Host method, you can set host memory from the Advanced tab. You can also choose the host login (`-host_login`) method suitable to your environment to allow IC Validator to communicate across multiple hosts. The default login method for host login is rsh.



LSF and SGE

ICV Launch provides separate execution methods for the LSF and SGE job submission tools. Use either LSF or SGE to specify resource settings when submitting your ICV job.

The LSF and SGE methods use the icv_batch utility as the underlying framework for launching IC Validator jobs across a grid. Based on your settings in the ICV Launch GUI, icv_batch input parameters such as `-hosts` and `-cores_per_host` are generated. You can select any number of hosts and request a certain number of cores per host from this method.

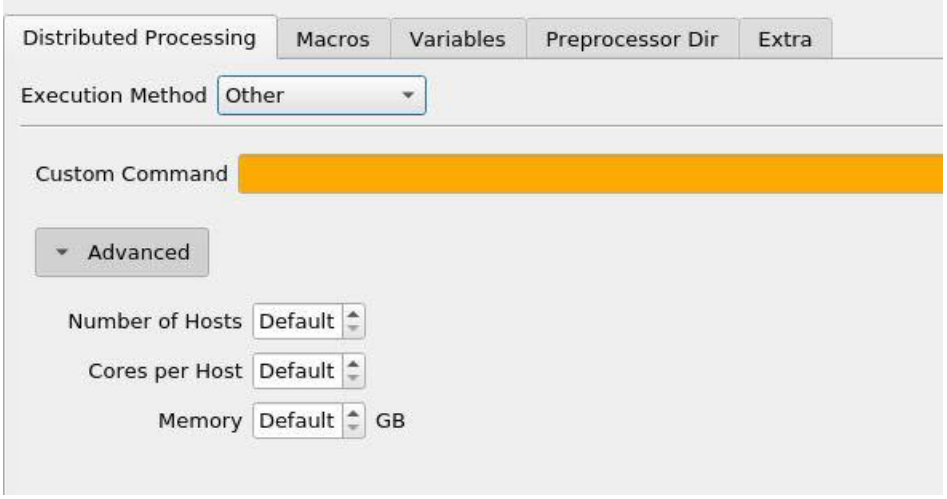


You can request only certain values for the Cores per Host setting. By default, you can select 2, 4, 8, or 16 cores per host. You can change the value list by setting the `LAUNCH_CORES_PER_HOST_LIST` environment variable.

This method also provides other settings such as enabling the ability to add hosts as they become available and ensuring that each acquired host is unique.

Other

Use the Other execution method when you have other job submission tools or your own pre-built scripts to access your execution environment. Contact your application engineer for more information on how to set up this method.



ICV Launch provides ease-of-use metacharacters to simplify coding. See Trigger Metacharacter Language for more information about how to implement your command line. In the Custom Command field, provide the command line that you want ICV Launch to issue when you press Execute. A typical command includes the command for your selected tool and the options to support your system configuration followed by the call to the IC Validator tool and command-line options.

The ICV Launch metacharacters provide easy access to the settings in the built-in option widgets in the Others tab. For example, the %cph metacharacter provides the value for the number of Cores per Host. The %n meta-character provides the value for the number of required hosts. And, the %m provides the value for the memory requirement. These shorthand metacharacters can be used to simplify the Custom Command entry coding. Finally, the %a metacharacter provides all of the command-line options required for submitting the job to the ICV engine. The %a metacharacter provides the command-line options, the call to the CLF file, the call to the runset configuration file, and the defined runset.

Example 4 Custom Command Entry

```
<Call to your submission tool> <System configuration options> <Call to the ICV Tool> <ICV Options>

dpRunScript -numHosts %n -coresPerHost %cph -memory %m $ICV_HOME_DIR/bin/icv %a
```

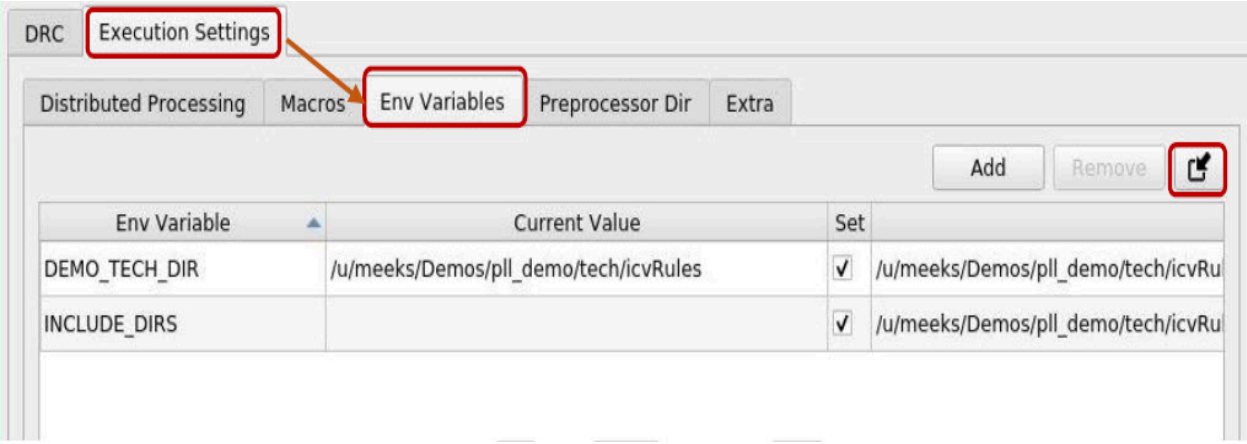
7

Environment Variable and Preprocessor Directive Control

You can set or modify environment variables that might be needed for an IC Validator run using the ICV Launch tool. ICV Launch inherits all environment variables from the shell from which it is invoked. You can also add to these variables from the ICV Launch GUI.

From the Execution Settings tab, choose Env Variables, as shown in Figure 33.

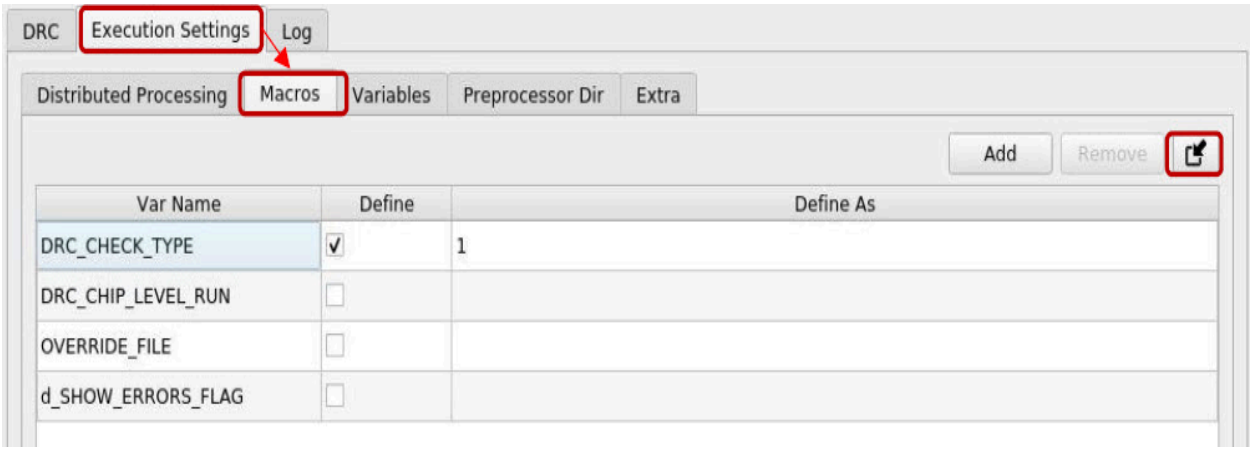
Figure 33 Environment Variables Tab



Type the name of an environment variable in the Var Name field. When you set a variable, the Current Value field is automatically populated with the value. Click the Set check box to set or unset the variable. The Set field allows you to modify the existing value of the variable. You can add or remove environment variables from this window. Additionally, click to read the specified runset and look for referenced environment variables.

From the Execution Settings tab, choose Macros, as shown in Figure 34.

Figure 34 Macros Tab

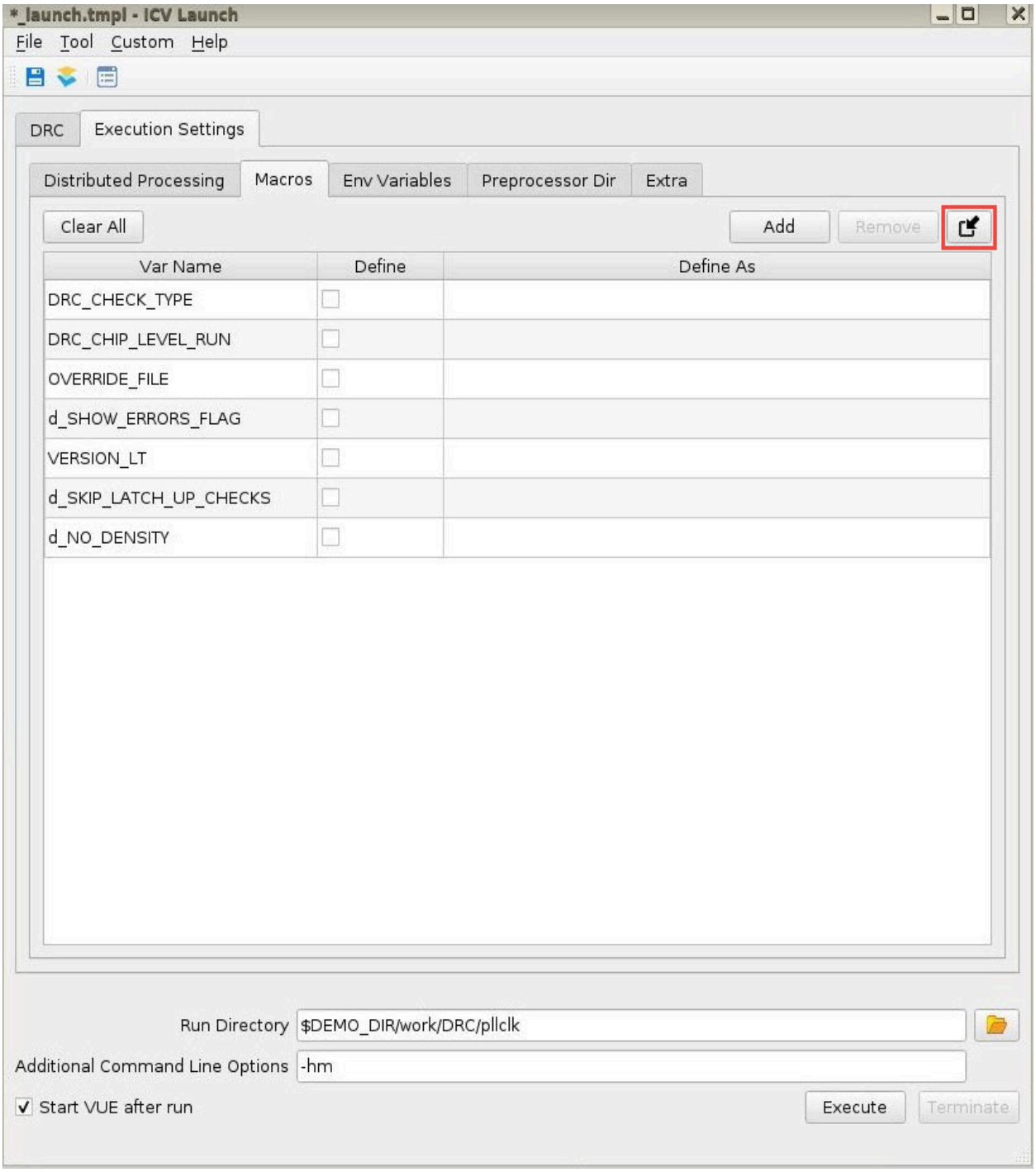


Type the name of the preprocessor directive in the Var Name field. If you check Define, you can populate the Define As field to set that directive. You can add or remove directives from this window. Additionally, click to read the specified runset and look for referenced directives.

Defining Preprocessor Macros and Environment Variables

From the Execution Settings tab, click the Macros or Env Variables tabs to view tables that provide access to establishing either one of these option types. The format for each of these tables and the associated functions are consistent.

Figure 35 Macros Tab



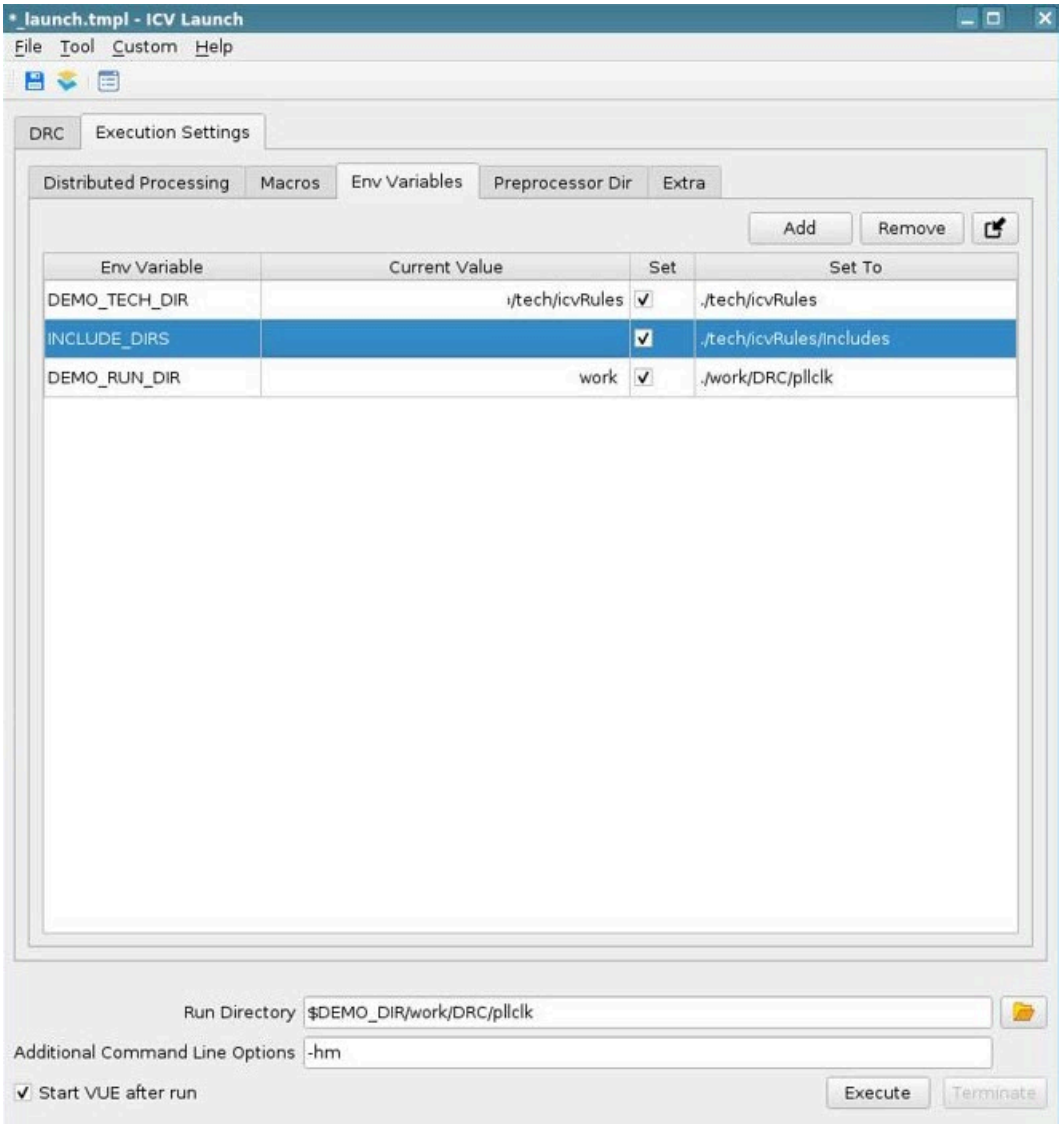
You can add or remove new or existing definitions. Additionally, click the highlighted button, as shown in Figure 36, to parse the defined runset file and automatically add any

of the definitions used in that file. Additional preprocessor definitions are added to the table from the input runset.

From the Env Variables tab, you can see the name and current settings of the definitions, as shown Figure 36.

In the Define or Set columns, you can alter the setting for the IC Validator run to the value entered in the Define As or Set To columns. This feature allows you to determine the initial values for the IC Validator run being configured.

Figure 36 Env Variables Tab



Feedback

Special Environment Variables

To affect tool behavior, you can define environment variables to set options or pass values that set or supersede defaults that a tool might otherwise use. Table 2 lists the IC Launch environment variables that can be used for this purpose.

Table 2 Special Environment Variables

Environment variable	Description
LAUNCH_RUNSET_TEXT_EDITOR	Runset text editor.
LAUNCH_DISABLE_VUE_FILE_GENERATION	Disables auto generation of .VUE file in ICV Launch.
LAUNCH_CORES_PER_HOST_LIST	Customizes the items in Cores per Host options for distributed processing SGE or LSF batch queuing systems.
LAUNCH_OLD_MASTER_SLAVE_BEHAVIOR	Reverts the master or slave to the previous behavior in ICV Launch.
LAUNCH_INIT_TEMPLATE_FILES	Automatically loads template files specified by this environment variable. This is a semicolon-separated list.
LAUNCH_TEMPLATE_FILE_DIR	Sets the default template files search directory.
LAUNCH_EXPORT_OASIS_SETTING	Sets the default export setting for the export OASIS file from the viewer (for Virtuoso).
LAUNCH_EXPORT_GDSII_SETTING	Sets the default export setting for the export GDS file from the viewer (for Virtuoso).
LAUNCH_EXPORT_CDL_SETTING	Sets the default export setting for the export CDL file from the viewer (for Virtuoso).
LAUNCH_EXPORT_FILENAME_SUFFIX	Sets the suffix for autogenerated export file name.
LAUNCH_APP_SETTINGS_PATH	Overrides the path to icv_launch.conf and defaults to ~/.config/Synopsys/icv_launch.conf
VUE_NO_DB_LOCK	Does not lock the PYDB that is preventing the IC Validator run.
VUE_SUPPRESS_LOG_DISPLAY	Specifies a regular regular expression to suppress the IC Validator engine logs.
CDL_HOOK_SCRIPT_NAME	(Legacy from the VUE execution page.)

8

ICV Launch Trigger Functionality

You can customize a flow by defining automation (scripts or executables) to occur at different stages of the process. There are three stages that can be accessed: 1) The Initialization stage; 2) The Data Export stage; and 3) The Pre-Trigger stage. At each of these stages, you can customize the flow and provide additional functionality to perform tasks required for that stage of the flow.

This chapter contains the following topics:

- Export Triggers
- Initialization Trigger
- Pre or Post-Execution Triggers
- Trigger Metacharacter Language
- External Trigger Definition Example

Export Triggers

Optional settings are often necessary for the exported (Layout or Schematic) design data to be correct for IC Validator processing. By default, ICV Launch uses the standard export utilities supported by the integrated design environment. For this usage, ICV Launch also provides access to the optional features for these utilities. The Export Trigger functions allow you to modify the export utilities used and provide any additional automation that might be required.

The Export Trigger functions are written in the native languages of the integrated design environments. For IC Validator Workbench and Custom Compiler, the native language is Tcl. For Cadence Virtuoso, the native language is Skill. Within these procedures, the ICV Launch flow can be tailored to automatically output the design data for IC Validator to process.

In the Custom Compiler and Virtuoso environments, built-in options in ICV Launch allow you to set the export options for those tools. In the Custom Compiler environment, there is an optional setting to call the export utility Options dialog box prior to the export execution. In Virtuoso, ICV Launch provides a pointer to a template file containing the export options.

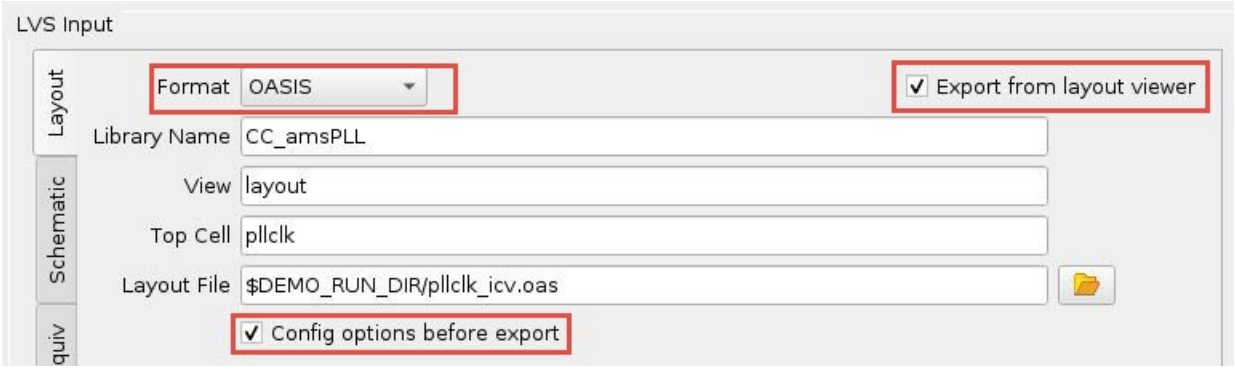
Feedback

Setting Custom Compiler Export Options

From the Layout tab, as shown in Figure 37, there are three highlighted settings:

- The Export from layout viewer option must be selected.
- The Format option must NOT be either OPEN ACCESS or NDM, as these options read the databases directly.
- The Config options before export option must be selected.

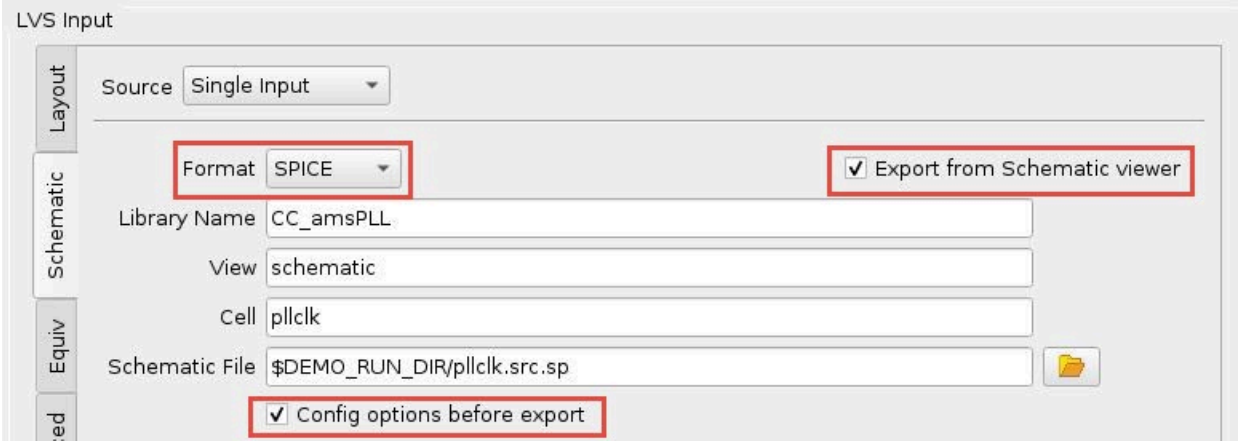
Figure 37 Layout Tab



From the Schematic tab, as shown in , there are three highlighted settings:

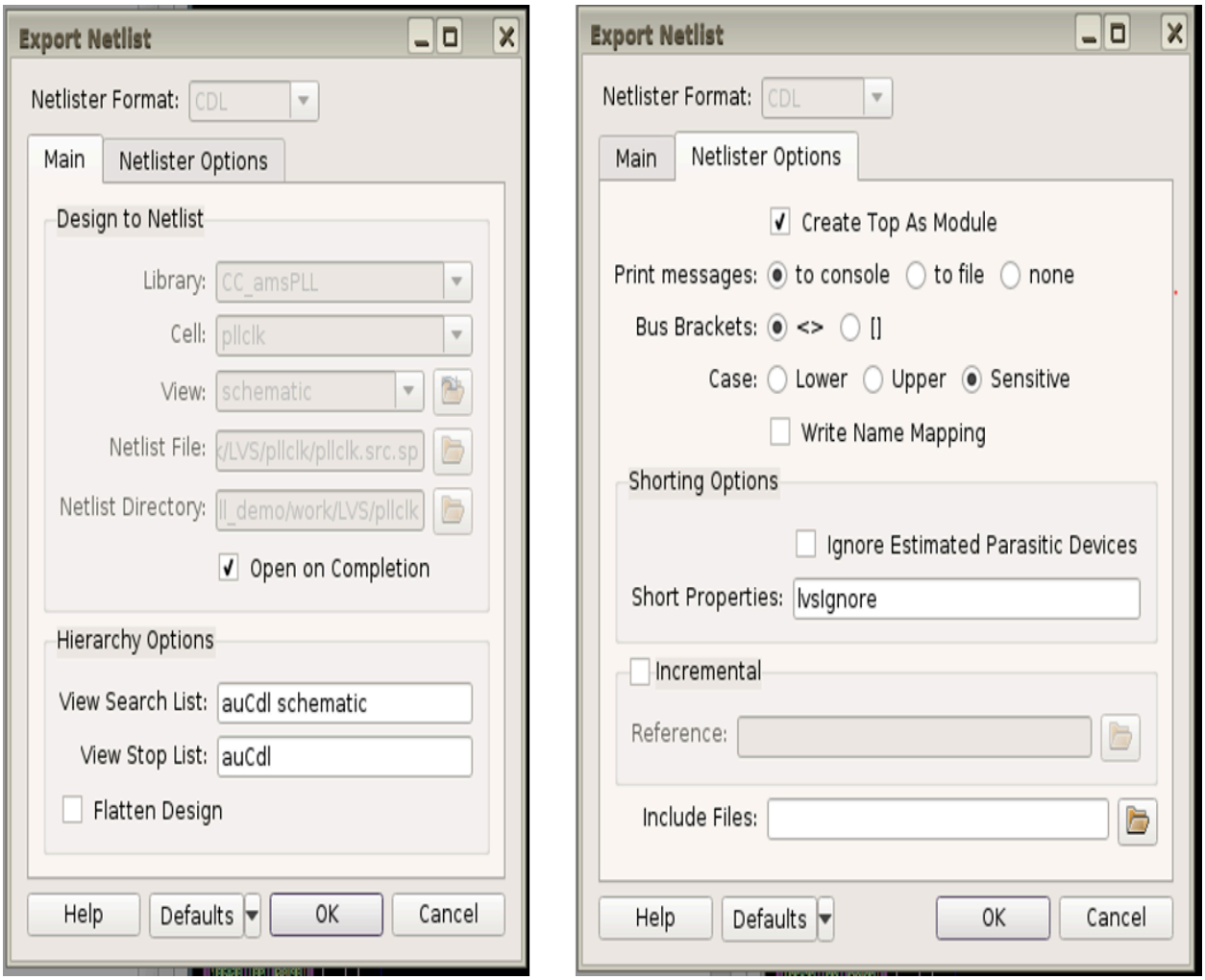
- The Export from Schematic viewer option must be selected.
- The Format option must be set to SPICE, as this option allows you to export a netlist.
- The Config options before export option must be selected.

Figure 38 Schematic Tab



After executing these options, and before the export function is called, a dialog window opens that provides access to all the option settings for that export. Options are available such as View Search List and View Stop List, as well as the Include Files list, which facilitates adding in cell libraries that might be externally provided.

Figure 39 Export Netlist Options

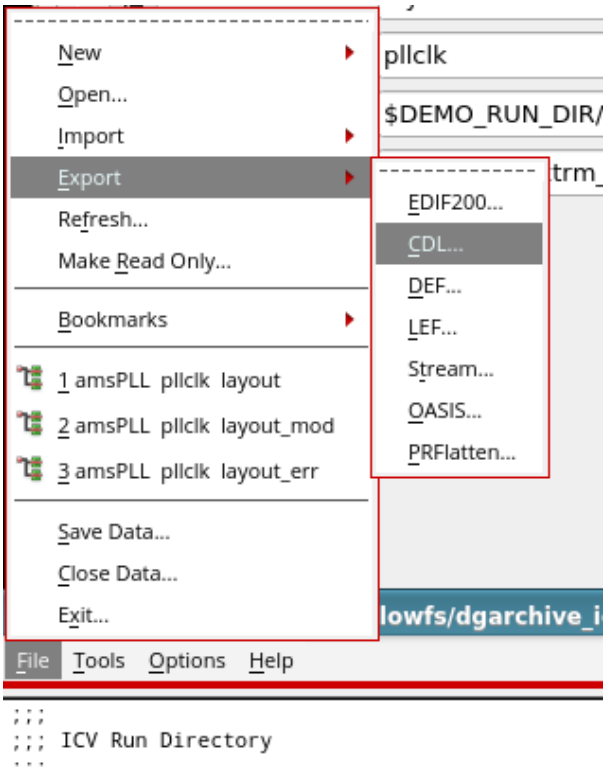


The options set here are automatically saved in your home directory and retained until changed again.

Setting Virtuoso Export Options

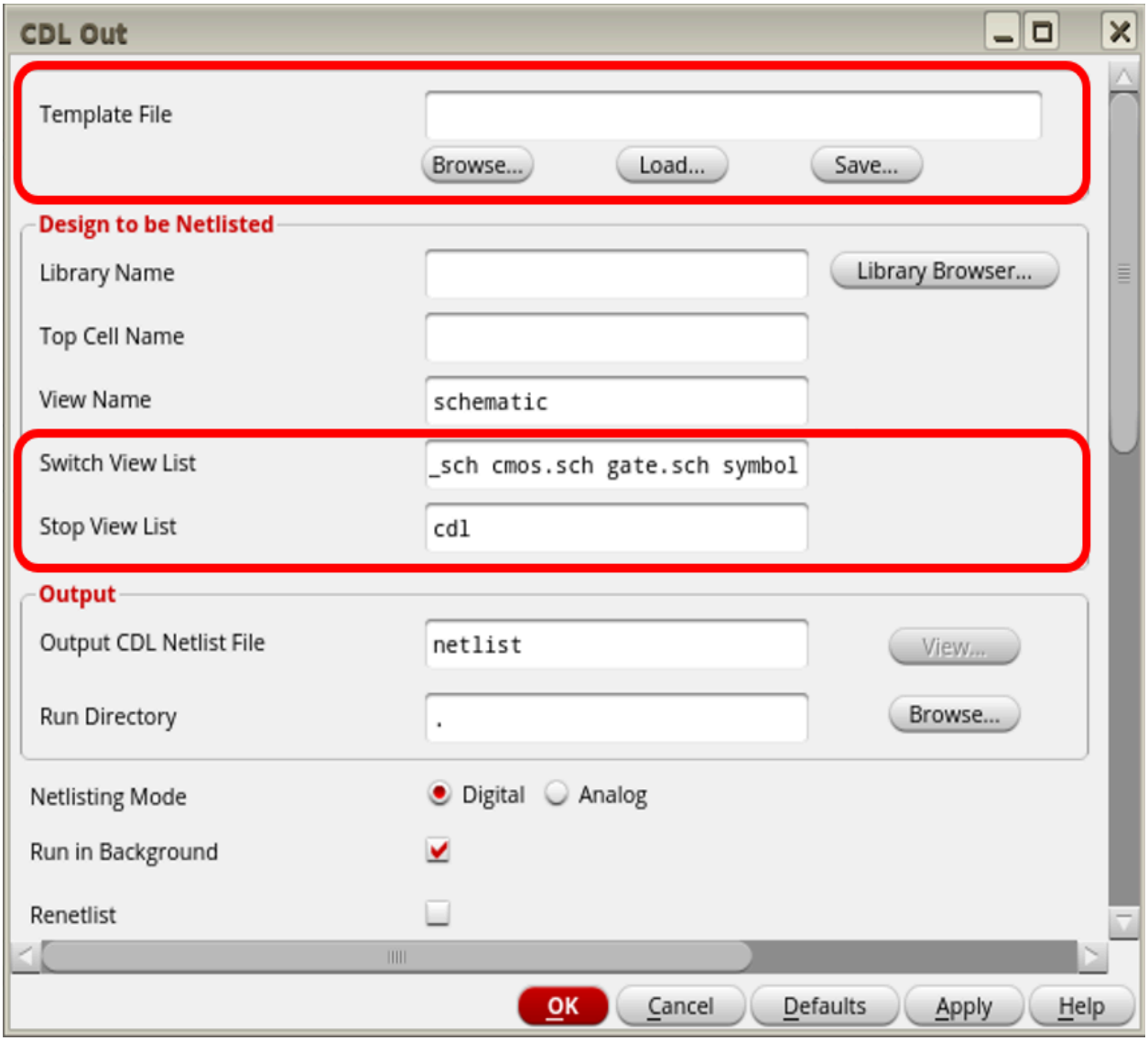
The ICV Launch integration into Cadence Virtuoso utilizes a standard feature in this design tools' export utilities.

From the Cadence Virtuoso CIW pull-down File, click the Export menu, to select the type of export you need. For Layout data, choose GDSII Stream or Oasis formats. For Schematic data, choose CDL (SPICE).



Following this selection, a new dialog box appears with all of the available option settings for that type of export. The Template File entry stores the option values for later use.

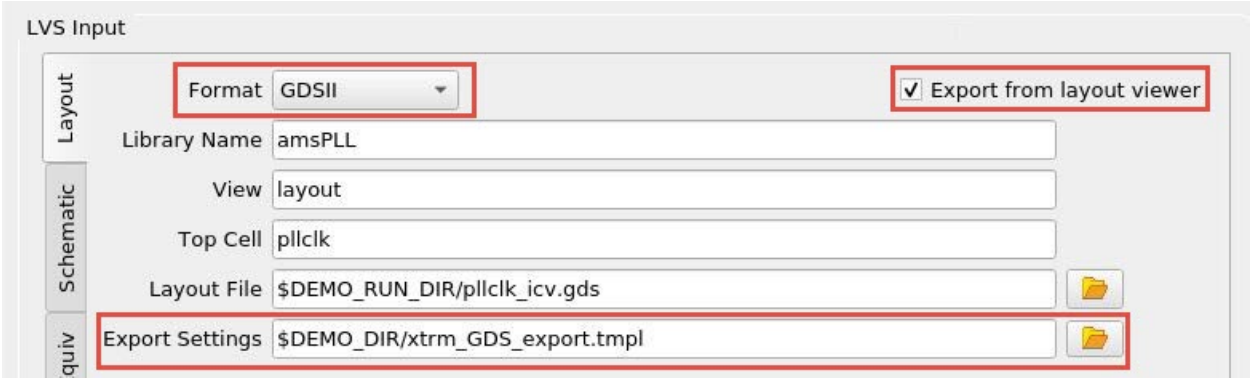
Feedback



From the ICV Launch GUI, there are three highlighted settings, as shown in Figure 40:

- The Export from layout viewer option must be selected.
- The Format option must NOT be either OPEN ACCESS or NDM, as these options read the databases directly.
- The Export Settings entry allows you to provide a template file, which is directly created from the design tool export function.

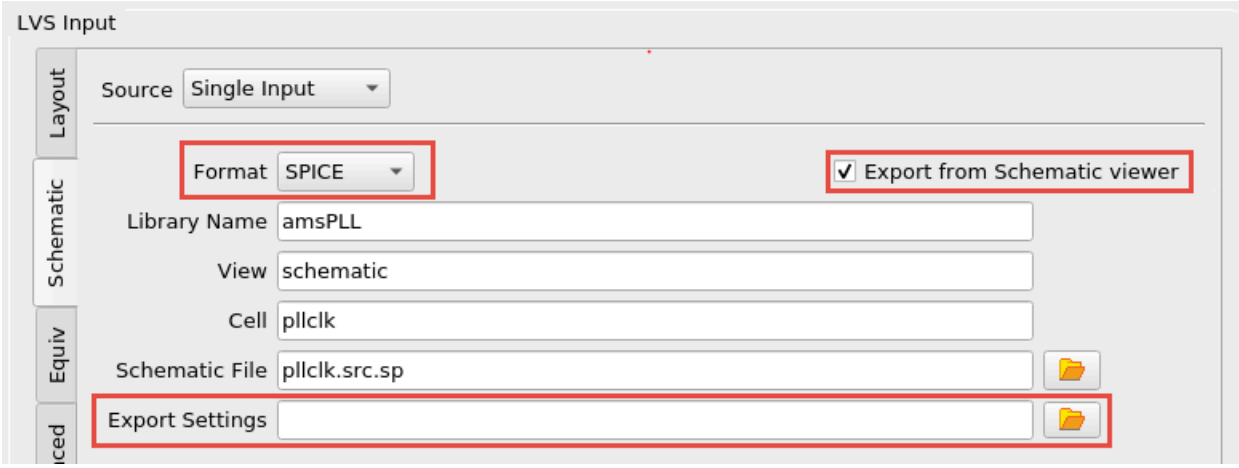
Figure 40 Layout Tab



From the Schematic tab, as shown in , there are three highlighted settings:

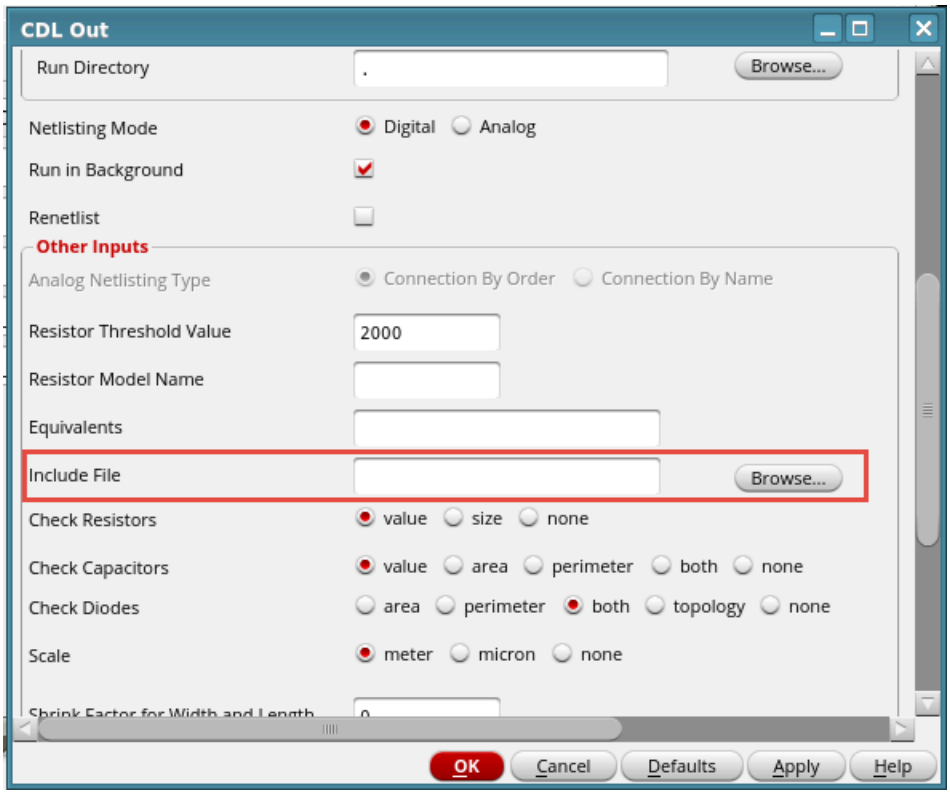
- The Export from Schematic viewer option must be selected.
- The Format option must be set to SPICE, as this option allows you to export a netlist.
- The Export Settings entry allows you to provide a template file, which is directly created from the design tool export function.

Figure 41 Schematic Tab



Additional option settings in the CDL export define the View Search and Stop lists, as well as a list of files to include in the newly exported net list, which facilitates adding in cell libraries that might be externally provided

Figure 42 CDL Out



The options set here are automatically saved in your home directory and retained until changed again.

Setting IC Validator WorkBench Export Options

ICV WorkBench supports only Layout data. Use the layout save command to export to the GDSII Stream or OASIS formats. The format and supported options are:

```
layout save [layout] -rename filename [-format format] [-compression
 value]
[-compaction boolean] [-modals boolean][-save_fig_properties on | off]
 [-libname libname]
[-options option_values] [-omit_text on | off]
```

Initialization Trigger

The Initialization Trigger function is available when ICV Launch is called from within a design environment (Custom Compiler, Cadence Virtuoso, or ICV Workbench). The Initialization Trigger is not part of an ICV Launch invocation from the (batch) command

line. From the command line, loading an existing template automates preloading of the entry settings.

When you select DRC or LVS from the menu to execute ICV Launch, the first operation performed is the Initialization Trigger. This is a predefined function or procedure within the design tool, and it is written in the native language for that tool. For Cadence Virtuoso, the Initialization Trigger is a Skill function. For Custom Compiler or ICV WorkBench, the Initialization Trigger is a Tcl function. This call defaults to the function definition provided in the ICV Launch installation tree. You can also modify this function and load (overload) this new definition in the design environment at any point before ICV Launch is called.

You can use the Initialization Trigger to query the design tool environment for the cell that is currently open and pass that information to the ICV Launch GUI through a JSON format template file. This provides a means by which ICV Launch can be fully populated automatically with all of the required settings. Additional uses for the Initialization Trigger are to establish environment variables, run directories, or any other utility that must exist before running an IC Validator job. See Example Trigger Scripts for an example of the Skill Initialization Trigger.

Previously, the trigger procedure required a setting of True to signal that the trigger completed. Now, the return value must be the file and path name for the template file generated by the trigger procedure. You can define your template file name instead of using a fixed file name.

Example 5 Initialization Trigger Procedure

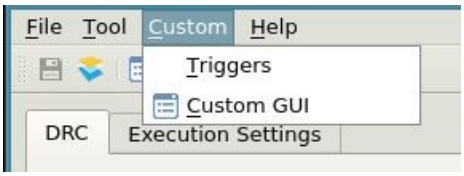
```
    close(templFile)
    ;;; return(t)                    ;;; Commented as of ICV 2021.06-SP3
    return(templFilePath)            ;;; Return Template Filename as of ICV
 2021.06-SP3
  ) ; end prog
) ; end procedure
```

Pre or Post-Execution Triggers

You can use pre-execution triggers to add automation following the export of the design data, but before the execution of the IC Validator job. You can use post-execution triggers to add automation following the execution of the IC Validator job.

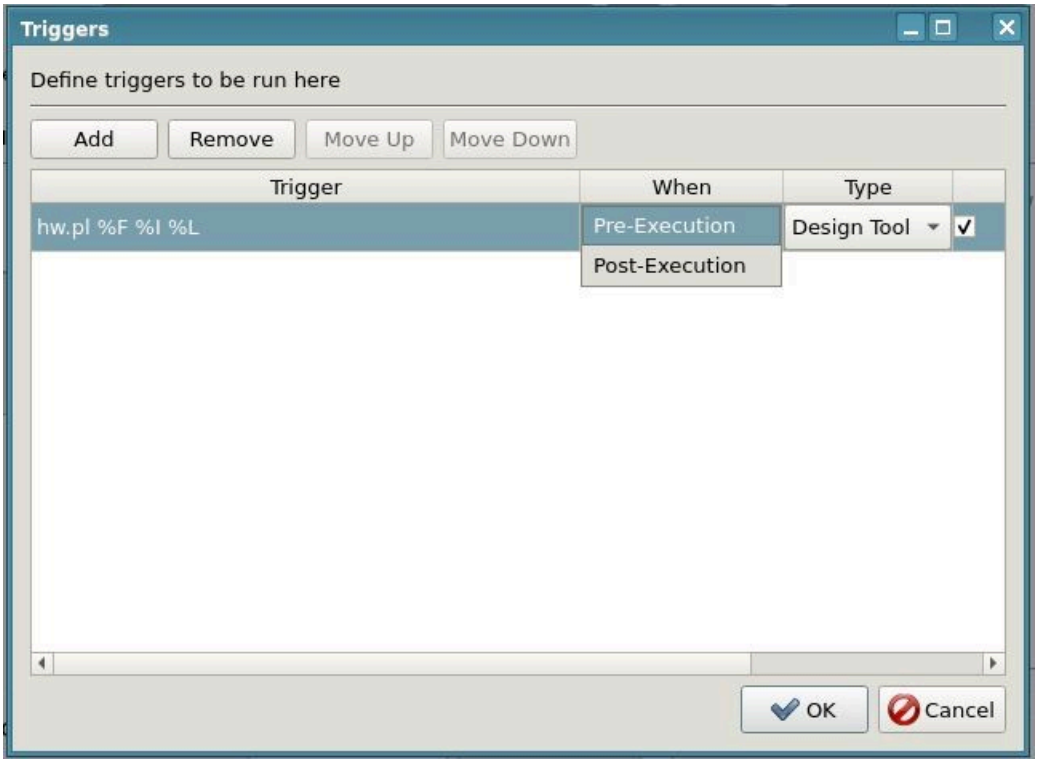
You can define executable scripts within or external to the design environment by selecting Custom >Triggers. Additionally, when running ICV Launch from the command line (batch mode), you can still use scripts external to a design environment.

Figure 43 Custom > Triggers



From the Triggers window you can add, remove, or adjust the order of execution for the defined trigger procedures. The Trigger field defines the command line to call the executable using parameters.

Figure 44 Triggers Functionality



The When field defines the stage where the trigger is executed. The Type field defines whether the trigger is internal or external to the design environment. Check the Run box if you want to execute this procedure.

Trigger Metacharacter Language

The ICV Launch Trigger definition field allows you to pass the values contained within ICV Launch GUI widgets to your trigger functions. The ICV Launch metacharacter language for DRC in Figure 45 and LVS in Figure 46 is a percent sign character “%” followed by an

alpha string. This metacharacter is replaced by the current value of the widget entry when the trigger function is called.

Figure 45 Trigger Parameters (DRC)

%F	Layout format
%l	Layout path
%L	Layout primary cell
%y	Layout library
%v	Layout view
%r	ICV runset file
%d	Run directory
%t	ICV template file

Figure 46 Trigger Parameters (LVS)

%c	Layout extracted netlist
%e	Equiv file
%F	Layout format
%l	Layout path
%L	Layout primary cell
%y	Layout library
%v	Layout view
%m	LVS report file
%r	ICV runset file
%d	Run directory
%t	ICV template file
%s	Schematic netlist
%S	Schematic primary cell

Figure 47 ICV Launch Metacharacters

#	meta-characters	Related Field Name	Description
1	%a	NA	ICV Options for following ICV run
2	%c	lvsLayoutTopCell + ".net"	Layout extracted netlist
3	%clf	NA	Command Line File
4	%cph	dpOtherCoresPerHost	Value of "Number of Cores per Host" field in DP->Other
5	%d	runDirectory	Run directory
6	%e	lvsEquivCellFile	Equiv file
7	%F	drcFormat, lvsLayoutFormat	Layout format
8	%l	drcLibraryPath, lvsLayoutLibraryPath	Layout design path
9	%L	drcTopCell, lvsLayoutTopCell	Layout primary cell
10	%m	dpOtherMemory	Value of Memory field in DP->Other
11	%n	dpOtherNumberOfHosts	Value "Number per Cores" field in DP->Other
12	%p	lvsLayoutTopCell + ".LVS_ERRORS"	LVS report file
13	%r	drcRunsetFile, lvsRunsetFile, Appended runset	ICV runset file path
14	%rc	NA	Runset Config
15	%rf	drcRunsetFile, lvsRunsetFile, Appended runset	ICV runset filename
16	%s	lvsSchematicFile	Schematic netlist
17	%S	lvsSchematicCell	Schematic primary cell
18	%t	NA	The loaded/Saved template file
19	%v	drcView, lvsLayoutView	Layout view
20	%y	drcLibraryName, lvsLayoutLibraryName	Layout library

Figure 48 ICV Launch Environment Variables (LAUNCH_)

#9	ENV name	Description	Expiration
1	LAUNCH_RUNSET_TEXT_EDITOR	Runset text editor	
2	LAUNCH_DISABLE_VUE_FILE_GENERATION	Disable auto generation of .VUE file in icv_launch	2020.09
3	LAUNCH_CORES_PER_HOST_LIST	Customize the items of Cores per Host options for DP SGE/LSF	
4	LAUNCH_OLD_MASTER_SLAVE_BEHAVIOR	Revert the master/slave to old behavior in icv_launch	2020.09
5	LAUNCH_INIT_TEMPLATE_FILES	Automatically load template files specified by this envvar	
6	LAUNCH_TEMPLATE_FILE_DIR	Set the default template files search directory	
7	LAUNCH_EXPORT_OASIS_SETTING	Set the default export setting for export OASIS file from viewer (for Virtuoso)	
8	LAUNCH_EXPORT_GDSII_SETTING	Set the default export setting for export GDS file from viewer (for Virtuoso)	
9	LAUNCH_EXPORT_CDL_SETTING	Set the default export setting for export CDL file from viewer (for Virtuoso)	
10	LAUNCH_EXPORT_FILENAME_SUFFIX	Set the suffix for auto-generated export filename	
11	LAUNCH_APP_SETTINGS_PATH	Override the path to icv_launch.conf. Default to ~/.config/Synopsys/icv_launch.conf	

Figure 49 ICV Launch Environment Variables (Other)

#9	ENV name	Description	Expiration
1	VUE_NO_DB_LOCK	Do not lock the PYDB that preventing ICV run	
2	VUE_SUPPRESS_LOG_DISPLAY	A regular expression to suppress ICV engine logs	
3	CDL_HOOK_SCRIPT_NAME	(It is a legacy from VUE execution page)	

External Trigger Definition Example

The following example shows a Perl script that prints to the total number and the values, of the parameters passed into this script, followed by the traditional "Hello World" message

to the shell. This utility is useful when testing the use of Triggers, as you can pass in metacharacters and see in the output IC Validator log when and what is produced.

```
#! /usr/bin/perl -w

if ($#ARGV >= 0) {
  $numArgs = $#ARGV + 1;
  print "$numArgs : @ARGV\n"
} # endif

print "Hello World!\n";
```

9

GUI Customization

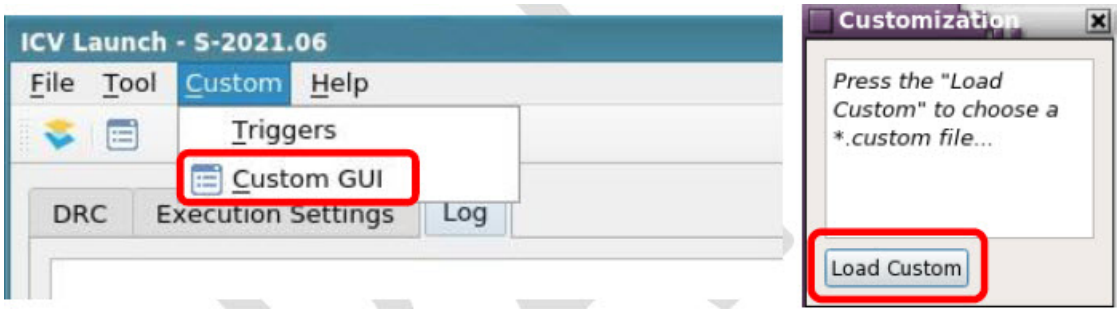
You can create your own custom GUI and add custom widgets that can impact an IC Validator run using ICV Launch. This provides the ability to control several settings such as preprocessor definitions, and override runset PXL option settings.

This feature requires you to write a JSON format definition file that can be used by ICV Launch to create custom widgets.

There are two methods for using the user customization file:

- Interactive:

  This can be done through the Custom menu which opens the customization dialog box. The JSON customization script can be loaded from this dialog box.



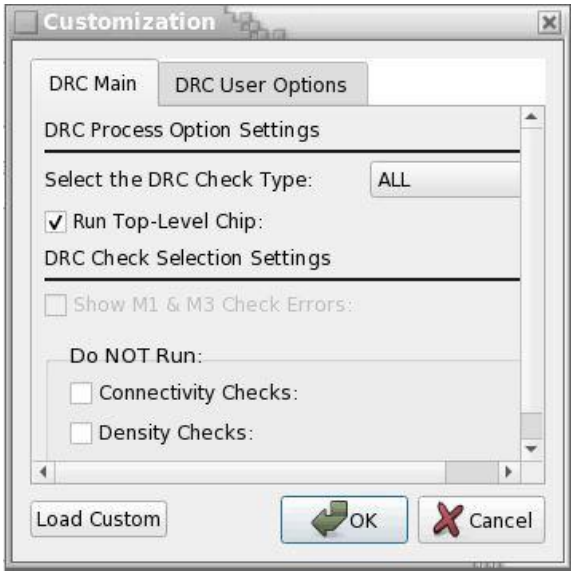
- Command line:

  ICV Launch can directly point to the JSON customization script from the command line using the `-custom` command-line option.

  ```
  icv_launch -custom userMenu.custom
  ```

Feedback

Both methods render GUI widgets that you define in the customization file. An example of a customized GUI is shown here:



Based on selections in the custom GUI, ICV Launch updates either the CLF File or the runset configuration file to pass the custom settings to the IC Validator engine.

Using the JSON customization script, you can also set up primary-secondary relationships between widgets, as well as a grouping of widgets, which can aid in the progressive disclosure of settings.

Here is an example of a customization script used for rendering the preceding custom GUI:

```
[
    {
        "type": "TAB",
        "prompt": "DRC Main"
    },
        {
            "type": "LABEL",
            "prompt": "DRC Process Option Settings"
        },
        {
                "type": "SEPARATOR"
        },
        {
            "type": "DEFINE",
            "name": "DRC_CHECK_TYPE",
            "choices": [
                {
                    "value": "0",
                    "prompt": "FEOL"
                },
                {
                    "value": "1",
                    "prompt": "ALL"
                },
                {
                    "value": "3",
                    "prompt": "BEOL"
                }
            ],
            "initval": "1",
            "boolean": 0,
            "select": 1,
            "listbox": 1,
            "prompt": "Select the DRC Check Type: "
        },
        {
            "type": "DEFINE",
            "name": "DRC_CHIP_LEVEL_RUN",
            "boolean": 1,
            "select": 0,
            "prompt": "Run Top-Level Chip: "
        },
```

Customization Language Introduction

The ICV Launch user-customization definition file is an external text file in JSON format. This file structure provides you with the ability to generate a working set of widgets for use with ICV Launch.

This file contains the definitions for the GUI widgets in the Customization window. The widgets are listed in sequential order. However, there are some options that allow you to make the widgets visibly dynamic in the definition file by defining a primary-secondary relationship and defining the conditions for the secondary widgets to be presented.

Types of Customization Widgets:

- Window Definitions
  - Window Geometry (Type “PROPERTY”)
    - Must be the first definition in the JSON customization file.
    - Defines the size of the customization window.

      Note:

      The default customization window geometry definitions are contained in a subdirectory in your home directory in a file called

      ~/.config/Synopsys/icv_launch.conf

      In this file under the main definitions section, you see settings as follows:

      ```
      [main]
      pos=@Point(12 56)
      size=@Size(769 786)
      ```

    - Parameters:
      - Height: Number of pixels tall.
      - Width. Number of pixels wide.

```
[
  {
        "type": "PROPERTY",
        "width": 500,
        "height": 300
        },
        …
        < Rest of JSON Definitions>
        ---
]
                ]
```

- Organization Definitions

- Tab: Allows the presentation of the graphical widget on multiple panes (tabs), thereby allowing for better organization and easier access to categories of widgets.
  - Parameters
    - Prompt: Title of the defined tab.
- Group: Collection of graphical widgets that are displayed or hidden as a group.
  - Parameters
    - Prompt: Displayed title of the group
    - Bold: 1 to make the group title bold.
    - Master: Name of another macro to depend on. This group is visible based on the setting of the primary widget.
    - Children: Array of widget definitions to be included in this group.

- Presentation Definitions
  - Label
    - Parameters
      - Prompt: Text to show in the GUI.
      - Bold: 1 to make the text bold.
  - Separator: Graphical horizontal line to indicate a separation.
- Functional Definitions
  - Define: Provides a *#define* definition for the PXL runset preprocessor in the command-line file (.clf).
  - Variable: Provides access to a built-in ICV Launch widget in the customization window, for example, Top Cell Name. This allows for easier organization of functions for user presentation. Additionally, variable widgets can be used to create fields (meta-characters) that can be passed to triggers or distributed processing command-line syntax.
  - Runset: Provides access to the runset configuration file to alter option settings in the runset.
    - Parameters for each of the functional widget types:
      - Name: Defines the name of the macro.

        Prompt: Text to show in the GUI.

        - Boolean: 1 to display a check box in the GUI.
        - Select: 1 to set a check box as checked.
        - Choices: A JSON array of strings or objects with value and prompt attributes, to generate UI for mutually exclusive choice.
        - Columns: An integer number of columns to layout buttons of choices.
        - Listbox: 1 to make the choices be represented as a combination box instead of buttons.
        - Initval: Selected text string from one element in the list defined in choices; the GUI shows this one as the default selected, or the value to yield if Boolean is set to 1.
        - Master: Name of another macro to depend on. This is visible if the master is checked.
        - columns: organize multiple-choice items in number of columns. Default is 1.
    - Added Parameter for the runset widget
        - PXL: Provides the ability to specify the additional text to be placed in the runset configuration file to address option settings.

Types of Customization Widgets

The following types of customization widgets provide you with a working set of widgets to use in ICV Launch.

- Organization Definitions
    - TAB. Tab definitions allow the presentation of the graphical widget on multiple pages (tabs) thereby allowing for better organization and easier access to categories of widgets.
        - Parameters
            - prompt. Title of the defined TAB.
    - GROUP. A collection of graphical widgets that are displayed or hidden as a group.
        - Parameters
            - prompt. The displayed title of the group.
            - bold. 1 to make the group title bold.

        - master. Name of another macro to depend on, this group will be visible based on the setting of the master widget.
        - children. Array of widget definitions to be included in this group.
- Presentation Definitions
    - LABEL
        - Parameters
            - prompt. Text to show on the GUI.
            - bold. 1 to make the text bold.
    - SEPARATOR. A graphical horizontal linked to indicate a separation.
- Functional Definitions
    - DEFINE. Provides a `#define` definition for the PXL runset preprocessor in the command-line file (.clf).
    - VARIABLE. Provides access to a built-in ICV Launch widget in the customization window, for example, Top Cell Name. This allows easier organization of functions for user presentation. Additionally, variable widgets can be used to create fields (metacharacters) that can be passed to triggers or distributed processing command-line syntax.
    - RUNSET. Provides access to the runset configuration file to alter OPTION settings in the PXL runset.
        - Parameters
            - name. Defines the name of the macro.
            - prompt. Text to show in the GUI.
            - Boolean. 1 to display a check box in the GUI.
            - select. 1 to set the check box as checked.
            - choices. JSON array of strings or objects with value and prompt attributes used to generate UI for mutually exclusive choices.
            - columns. Integer number of columns to layout buttons of choices.
            - listbox. 1 to make the choices be represented as combination box instead of buttons.

- initval. Selected text string from one element in the list defined in choices. The GUI shows this one as the selected default, or the value to yield if Boolean is set to 1.
- master. Name of another macro to depend on; this is visible if the master is checked.
- columns. Organize multiple-choice items in number of columns. Default is 1.

Built-In Widget Reference Names

For the main ICV Launch GUI widgets, you can reference keyword names based on their assigned entry widget. The following definitions for the widget names can be used in the User-Customization VARIABLE widgets, as shown in Figure 50.

Figure 50 DRC GUI Built-In Widget Keywords

DRC Page

#	Field Name	Type	Runset Config CLF	Mapping
1	drcRunsetFile	String	CLF / JSONObject	runset file
2	drcRuleRecipeSetting	String	CLF	-svc
3	drcFormat	String	CLF	-f
4	drcLayoutFile	String	CLF	-I (drcFormat = GDSII, OASIS)
5	drcLibraryPath	String	CLF	-P (drcFormat = NDM, Milkyway)
6	drcLibraryName	String	CLF	I (drcFormat = NDM, Milkyway, OpenAccess)
7	drcView	String	CLF	-oa_view (drcFormat = OpenAccess)
8	drcTopCell	String	CLF	-c
9	drcLibraryDefinitionFile	String	CLF	-oa_lib_defs
10	drcLayerMappingFile	String	CLF	-oa_layer_map
11	drcArea	String	Runset Config	-D INCREMENTAL_OPTIONS_SELECT_WINDOW
12	drcVoltageTextFile	String	Runset Config	text_options() - edtext
13	drcMaxErrorsPerCheck	String	Runset Config	error_options() - error_sampling_per_check
14	drcMaxErrors	integer	Runset Config	error_options() - error_limit_per_check

The LVS pages contain more options and settings. Therefore, there are more widgets that can be accessed from the keywords, as shown in Figure 51.

Figure 51 LVS GUI Built-In Widget Keywords

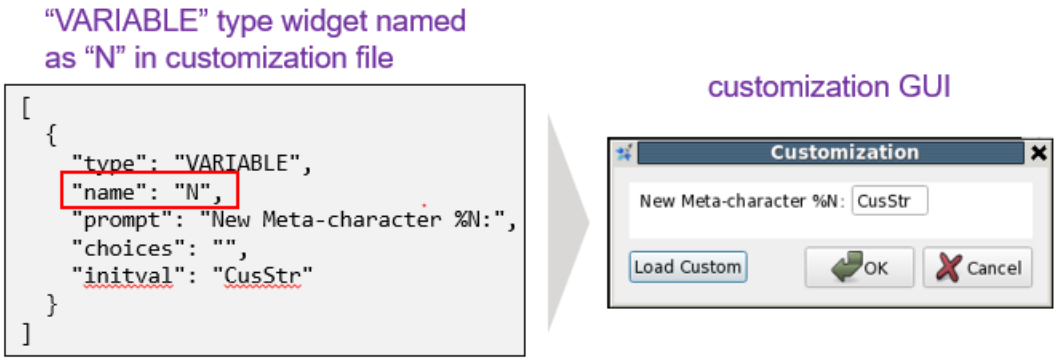
LVS Page

#	Field Name	Type	Runset Config CLF	Mapping
1	lvsRunsetFile	String		runset file
2	lvsStep	String	CLF	"-ex / -C"
3	lvsLayoutFormat	String	CLF	"-f"
4	lvsLayoutFile	String	CLF	"-l (lvsLayoutFormat = GDSII, OASIS)"
5	lvsLayoutLibraryPath	String	CLF	"-P (lvsLayoutFormat = NDM, Milkyway)"
6	lvsLayoutLibraryName	String	CLF	"-l (lvsLayoutFormat = NDM, Milkyway, OpenAccess)"
7	lvsLayoutView	String	CLF	"-oa_view (lvsLayoutFormat = OpenAccess)"
8	lvsLayoutTopCell	String	CLF	"-c"
9	lvsLayoutLibraryDefinitionFile	String	CLF	"-oa_lib_defs"
10	lvsLayoutLayerMappingFile	String	CLF	"-oa_layer_map"
11	lvsSchematicSource	String	CLF	To decide ***single*** or ***multiple*** input
12	lvsSchematicFormat	String	CLF	"-sf (lvsSchematicSource = Single Input)"
13	lvsSchematicFile	String	CLF	"-s (lvsSchematicFormat!= Export CDL)
14	lvsSchematicLibrary	String		For export CDL process
15	lvsSchematicCell	String	CLF	"-stc (lvsSchematicSource = Single Input)"
16	lvsSchematicView	String		For export CDL process
17	lvsSchematicMIFormat	String	CLF	"-sf (lvsSchematicSource = Multiple Input)
18	lvsSchematicMICell	String	CLF	"-s (lvsSchematicSource = Multiple Input)"
19	lvsSchematicMIMergedFile	String	CLF	"-stc (lvsSchematicSource = Multiple Input)"
20	lvsSchematicMIInputFile	String		Input files for Nettran
21	lvsSchematicMiOptionFile	String		Option file for Nettran
22	lvsEquivCellFile	String	CLF	"-e"
23	lvsGenUserEquiv	String	Runset Config	lvs_options() -
24	lvsEdtextFile	String	Runset Config	text_options() - edtext
25	lvsAdvPowerNets	String	Runset Config	text_options() - layout_power
26	lvsAdvGroundNets	String	Runset Config	text_options() - layout_ground
27	lvsAdvEquivCells	String	Runset Config	lvs_black_box_options() - equiv_cells
28	lvsAdvEquivPorts	String	Runset Config	lvs_black_box_options() - equate_ports
29	lvsAdvRemoveSchProts	String	Runset Config	lvs_black_box_options() - remove_schematic_ports
30	lvsAdvRemoveLayPorts	String	Runset Config	lvs_black_box_options() - remove_layout_ports
31	lvsAdvVirtualConnectNetNames	Boolean	Runset Config	text_net() - opens = "MERGE_CONNECTED_AND_TOP"
32	lvsAdvVirtualConnectNet	String	Runset Config	text_net() - merge_open_net_names
33	lvsAdvMergeSeriesTable	String	Runset Config / JSON Array	merge_series() - device_type, device_names
34	lvsAdvParallelTable	String	Runset Config / JSON Array	merge_parallel() - device_type, device_names
35	lvsAdvShortEquivNodesTable	String	Runset Config / JSON Array	short_equivalent_nodes() - device_type, device_names

Referencing Widget Names

Built-in widgets in the ICV Launch GUI and user-defined variable widgets in the Customization window often interact to provide access to other external features. These widgets might need to be referenced as the primary for enabling or disabling that widget. Therefore, a consistent method of addressing these user-defined widgets is necessary.

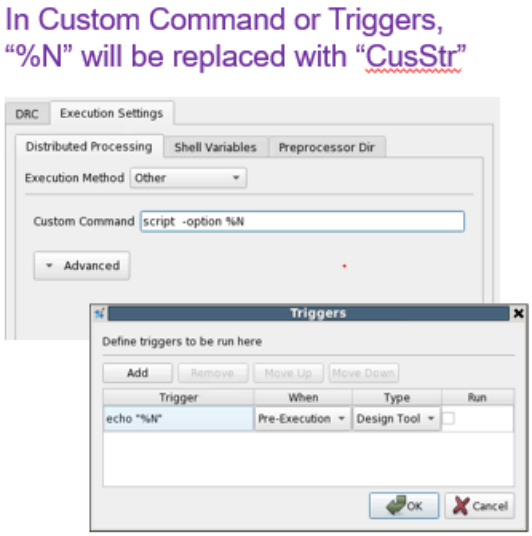
The values of and references to these user-defined customization variable widgets or built-in widgets in the ICV Launch GUI can be referenced using meta language. The built-in meta characters are shown in the following figures. The built-in and user-defined widgets can also be referenced by a name prefixed with a percent sign (%) followed by the name for that widget enclosed in curly-braces “{“ & “}”.



If a customization VARIABLE widget, “N” is defined, pass the value of this widget to the Custom Command entry or to a Trigger function by setting the following, as shown in Figure 52.

Feedback

Figure 52 Triggers in Execution Settings



Examples:

Curly braces allow for concatenation of suffix strings to complete the required results.

A widget with a value of "CusStr" is defined.

```
%N    => CusStr
```

This example attempts to concatenate the "otherStrings" suffix to the value contained in %N. But, the "NotherStrings" widget name is not recognized.

```
%NotherStrings  => Not recognized
```

This example resolves the value "CurStr" contained in the widget name %{N}.

```
%{N}  => CusStr
```

This example correctly concatenates the variable value with the suffix string.

```
%{N}otherStrings   => CusStrotherStrings
```

A

Example Trigger Scripts

Initialization Triggers are called when ICV Launch is invoked from within a design environment, and before the tool is started. The purpose for this procedure is to determine the currently active design attributes and create an Initialization Template (JSON) file to pre-load the entries in the ICV Launch GUI. Additionally, this function might create and populate run directories, or any other setup operation required to successfully complete your IC Validator job.

Example 6 shows a Tcl procedure, “VUE::ICVLaunchInitializationFile”, which determines the currently active cell and creates an Initialization Template (JSON) file named "$HOME/.synopsys/icvwb/icv_launch.tmpl” by default. The default file is in the IC Validator WorkBench installation directory in the “./etc/fw/contrib/ICVLaunchInitialization.tcl” file.

Example 6 IC Validator WorkBench Initialization Trigger

```
#
# IC Validator Workbench / ICV Launch Initialization Trigger
#
proc VUE::ICVLaunchInitializationFile {type {filename "icv_launch.tmpl"}}
 {
# Local Variables
    variable application
    set templFilePath
 $::env(HOME)/.synopsys/$user_mode::application/$filename
    set templFile [open $templFilePath w+]
# Get Cell Data
    set layoutLibraryName ""
    if { [catch {layout filename [layout active]}] == 0 } {
        set layoutLibraryName [layout filename [layout active]]
    }
    set layoutCellName ""
    if {[catch {cell active}] == 0} {
        set layoutCellName [cell active]
    }
    set layoutFormat ""
    if {[catch {layout format [layout active]}] == 0} {
        set activeFormat [layout format [layout active]]
        if {$activeFormat == "oasis"} {
            set layoutFormat "OASIS"
        } elseif {$activeFormat == "gds"} {
            set layoutFormat "GDSII"
        }
    }
```

Feedback

```
    set layoutViewName "Layout"
# Generate ICV Launch Initialization Template File
    puts $templFile "\{"
    puts $templFile "    \"launch\": \{"
    puts $templFile "        \"run1\": \{"
    switch $type {
      "DRC" -
      "drc" {
            puts $templFile "            \"drcFormat\":
 \"$layoutFormat\","
            puts $templFile "            \"drcLayoutFile\":
 \"$layoutLibraryName\","
            puts $templFile "            \"drcTopCell\":
 \"$layoutCellName\","
      }
      "lvs" {
          puts $templFile "            \"lvsLayoutFormat\":
 \"$layoutFormat\","
          puts $templFile "            \"lvsLayoutFile\":
 \"$layoutLibraryName\","
          puts $templFile "            \"lvsLayoutTopCell\":
 \"$layoutCellName\","
          puts $templFile "            \"lvsSchematicCell\": \"\","
      }
      default {
          puts "*** Error *** Unknown Run Type: ($type) Cannot generate
 Template file."
      }
    }; # end case
# Output Environmental Variable Definitions
#     puts ";;; Generating ICV Launch Template file Shell Variables"
    puts $templFile "            \"startVueAfterRun\": true,"
    puts $templFile "            \"runType\": \"$type\""
    puts $templFile "        \}"
    puts $templFile "    \}"
    puts $templFile "\}"
    close $templFile
    return $templFilePath
} ;# end prog
# end procedure
```

Example 7 shows a Tcl procedure, initializationTriggers, which determines the currently active cell and creates an Initialization Template (JSON) file name by default, _launch.tmpl in the defined run directory. The default file in the Custom Compiler installation directory is in the ./auxx/physical_verification/icv/launch/setup_dialog.tcl file.

Example 7 Custom Compiler Initialization Trigger

```
#
# Custom Compiler / ICV Launch Initialization Trigger
#
proc initializationTriggers { type xtPveJob } {
```

Feedback

```
   variable tmplFile
   variable name
   set procName "::initializationTrigger"
   if {  "" != [info procs $procName]  } {
       xt::_pve::warning "execute customize procedure '$procName'"
       $procName
       return
   }
   set libName [db::getAttr editFile.container.libName -of
[de::getActiveContext]]
   set cellName [db::getAttr editFile.container.cellName -of
[de::getActiveContext]]
   set viewName [db::getAttr editFile.container.name -of
[de::getActiveContext]]
   set dmCellView [::xt::physicalVerification::_private::getDmCellView
[de::getActiveContext]]
   set viewType [db::getAttr viewType -of $dmCellView]
   set libDefs [dm::getTopLibDefPath]
   set rundir [::xt::getRunDirName "$libName/$cellName/$viewName" -tool
"icv" \
       -job xtPveJob]
   set rundir [file normalize $rundir]
   if { [catch { file mkdir $rundir } err] } {
       error "Unable to create directory '$rundir' because: $err"
   }
   if {[catch {file mkdir [file dirname $tmplFile]} fd] || [catch {open
$tmplFile w} fd]} {
       ::xt::_pve::warning "Unable to create initialization triggers
file because: $fd"
       return
   }
   set expFileNameSuffix "_exported"
   if { [info exists ::env(LAUNCH_EXPORT_FILENAME_SUFFIX)] } {
       set expFileNameSuffix $::env(LAUNCH_EXPORT_FILENAME_SUFFIX)
   }
   set templFilePath $tmplFile
   set templFile [open $templFilePath w+]
   puts $templFile "{\n"
   puts $templFile "    \"launch\": {\n"
   puts $templFile "        \"run1\": {\n"
   if { $type == "DRC" } {
       if { $viewType == "maskLayout" } {
           puts $templFile "            \"drcFormat\":
\"OPENACCESS\",\n"
           puts $templFile [format "            \"drcLibraryName\":
\"%s\",\n" $libName]
           puts $templFile [format "            \"drcTopCell\":
\"%s\",\n" $cellName]
           puts $templFile [format "            \"drcView\": \"%s\",\n"
$viewName]
           puts $templFile [format "            \"drcLayoutFile\":
\"%s%s.gds\",\n" \
               $cellName $expFileNameSuffix]
```

Feedback

```
            puts $templFile "            \"drcExpFromLayViewer\":
true,\n"
            puts $templFile "            \"startVueAfterRun\": true,\n"
            puts $templFile [format "    \"drcLibraryDefinitionFile\":
\"%s\",\n" $libDefs]
            puts $templFile [format "    \"runDirectory\": \"%s\",\n"
$rundir]
        }
    } else {
        if { $viewType == "maskLayout" } {
            puts $templFile "            \"lvsLayoutFormat\":
\"OPENACCESS\",\n"
            puts $templFile [format "
\"lvsLayoutLibraryName\": \"%s\",\n" $libName]
            puts $templFile [format "            \"lvsLayoutTopCell\":
\"%s\",\n" $cellName]
            puts $templFile [format "            \"lvsLayoutView\":
\"%s\",\n" $viewName]
            puts $templFile [format "            \"lvsLayoutFile\":
\"%s%s.gds\",\n" \
                $cellName $expFileNameSuffix]
            puts $templFile "            \"lvsExpFromLayViewer\":
true,\n"
            puts $templFile "            \"startVueAfterRun\": true,\n"
            puts $templFile [format "
\"lvsLayoutLibraryDefinitionFile\": \"%s\",\n" $libDefs]
            puts $templFile [format "    \"runDirectory\": \"%s\",\n"
$rundir]
            set xrefCellViewName [::xt::icv::_private::pickXrefCellView
$cellName "sch"]
            if { $xrefCellViewName != "" } {
                puts $templFile "            \"lvsSchematicFormat\":
\"SPICE\",\n"
                puts $templFile [format "
\"lvsSchematicLibraryName\": \"%s\",\n" $libName]
                puts $templFile [format "
\"lvsSchematicCell\": \"%s\",\n" $cellName]
                puts $templFile [format "
\"lvsSchematicView\": \"%s\",\n" $xrefCellViewName]
                puts $templFile [format "
\"lvsSchematicFile\": \"%s%s.cdl\",\n" \
                    $cellName $expFileNameSuffix]
                puts $templFile "            \"lvsExpFromSchViewer\":
true,\n"
            }
        }
        if { $viewType == "schematic" } {
            puts $templFile "            \"lvsSchematicFormat\":
\"SPICE\",\n"
            puts $templFile [format "
\"lvsSchematicLibraryName\": \"%s\",\n" $libName]
            puts $templFile [format "            \"lvsSchematicCell\":
\"%s\",\n" $cellName]
```

Feedback

```
            puts $templFile [format "            \"lvsSchematicView\":
 \"%s\",\n" $viewName]
            puts $templFile [format "            \"lvsSchematicFile\":
 \"%s%s.cdl\",\n" \
                $cellName $expFileNameSuffix]
            puts $templFile "            \"lvsExpFromSchViewer\":
 true,\n"
            set xrefCellViewName [::xt::icv::_private::pickXrefCellView
 $cellName "lay"]
            if { $xrefCellViewName != "" } {
                puts $templFile "            \"lvsLayoutFormat\":
 \"OPENACCESS\",\n"
                puts $templFile [format "
 \"lvsLayoutLibraryName\": \"%s\",\n" $libName]
                puts $templFile [format "
 \"lvsLayoutTopCell\": \"%s\",\n" $cellName]
                puts $templFile [format "            \"lvsLayoutView\":
 \"%s\",\n" $xrefCellViewName]
                puts $templFile [format "            \"lvsLayoutFile\":
 \"%s%s.gds\",\n" \
                    $cellName $expFileNameSuffix]
                puts $templFile "            \"lvsExpFromLayViewer\":
 true,\n"
                puts $templFile "            \"startVueAfterRun\":
 true,\n"
                puts $templFile [format "
 \"lvsLayoutLibraryDefinitionFile\": \"%s\",\n" $libDefs]
                puts $templFile [format "    \"runDirectory\": \"%s\",\n"
 $rundir]
            }
        }
    }
    puts $templFile [format "            \"runType\": \"%s\"\n" $type]
    puts $templFile "        }\n"
    puts $templFile "    }\n"
    puts $templFile "}\n"
    close $templFile
    return 1
}
# end procedure
```

Example 8 shows a Tcl procedure, “” , which determine the currently active cell and create an Initialization Template (JSON) file named by default, _launch.tmpl in the defined run directory. The default file is in the IC Validator WorkBench installation directory in the ./etc/fw/contrib/ICVLaunchInitialization.tcl file.

Example 8 Cadence Virtuoso Initialization Trigger

```
;
; Cadence Virtuoso / ICV Launch Initialization Trigger
;
procedure(initializationTriggers(type)
```

Feedback

```
prog((viewType templFilePath templFile xrefCellViewName)
    viewType = hiGetCurrentWindow()~>cellView~>cellViewType
    templFilePath = "./_launch.tmpl"
    templFile = outfile(templFilePath)
    if(templFile == nil then
        printf("[ICV Launch] Cannot write to the file ./_launch.tmpl,
 probably due to permission, please check.\n")
        return(nil)
    )
    fprintf(templFile "{\n")
    fprintf(templFile "    \"launch\": {\n")
    fprintf(templFile "        \"run1\": {\n")

    fileNameSuf = "_exported"
    if(getShellEnvVar("LAUNCH_EXPORT_FILENAME_SUFFIX") != nil then
        fileNameSuf = getShellEnvVar("LAUNCH_EXPORT_FILENAME_SUFFIX")
    )
    if(type == "DRC" then
        if(viewType == "maskLayout" then
            fprintf(templFile "            \"drcFormat\": \"GDSII\",\n")
            fprintf(templFile "            \"drcLibraryName\": \"%s\",\n"
 hiGetCurrentWindow()~>cellView~>libName)
            fprintf(templFile "            \"drcTopCell\": \"%s\",\n"
 hiGetCurrentWindow()~>cellView~>cellName)
            fprintf(templFile "            \"drcView\": \"%s\",\n"
 hiGetCurrentWindow()~>cellView~>viewName)
            fprintf(templFile "            \"drcLayoutFile\":
 \"%s%s.gds\",\n" hiGetCurrentWindow()~>cellView~>cellName, fileNameSuf)
            fprintf(templFile "            \"drcExpFromLayViewer\":
 true,\n")
        )
    else
        if(viewType == "maskLayout" then
            fprintf(templFile "            \"lvsLayoutFormat\":
 \"GDSII\",\n")
            fprintf(templFile "            \"lvsLayoutLibraryName\":
 \"%s\",\n" hiGetCurrentWindow()~>cellView~>libName)
            fprintf(templFile "            \"lvsLayoutTopCell\":
 \"%s\",\n" hiGetCurrentWindow()~>cellView~>cellName)
            fprintf(templFile "            \"lvsLayoutView\": \"%s\",\n"
 hiGetCurrentWindow()~>cellView~>viewName)
            fprintf(templFile "            \"lvsLayoutFile\":
 \"%s%s.gds\",\n" hiGetCurrentWindow()~>cellView~>cellName, fileNameSuf)
            fprintf(templFile "            \"lvsExpFromLayViewer\":
 true,\n")
            xrefCellViewName =
 pickXrefCellView(hiGetCurrentWindow()~>cellView~>cellName "sch")
            if(xrefCellViewName != "" then
                fprintf(templFile "            \"lvsSchematicFormat\":
 \"SPICE\",\n")
                fprintf(templFile "
    \"lvsSchematicLibraryName\": \"%s\",\n"
 hiGetCurrentWindow()~>cellView~>libName)
```

Feedback

```
                fprintf(templFile "            \"lvsSchematicCell\":
\"%s\",\n" hiGetCurrentWindow()~>cellView~>cellName)
                fprintf(templFile "            \"lvsSchematicView\":
\"%s\",\n" xrefCellViewName)
                fprintf(templFile "            \"lvsSchematicFile\":
\"%s%s.cdl\",\n" hiGetCurrentWindow()~>cellView~>cellName, fileNameSuf)
                fprintf(templFile "            \"lvsExpFromSchViewer\":
true,\n")
            )
        )
        if(viewType == "schematic" then
            fprintf(templFile "            \"lvsSchematicFormat\":
\"SPICE\",\n")
            fprintf(templFile "            \"lvsSchematicLibraryName\":
\"%s\",\n" hiGetCurrentWindow()~>cellView~>libName)
            fprintf(templFile "            \"lvsSchematicCell\":
\"%s\",\n" hiGetCurrentWindow()~>cellView~>cellName)
            fprintf(templFile "            \"lvsSchematicView\":
\"%s\",\n" hiGetCurrentWindow()~>cellView~>viewName)
            fprintf(templFile "            \"lvsSchematicFile\":
\"%s%s.cdl\",\n" hiGetCurrentWindow()~>cellView~>cellName, fileNameSuf)
            fprintf(templFile "            \"lvsExpFromSchViewer\":
true,\n")
            xrefCellViewName =
pickXrefCellView(hiGetCurrentWindow()~>cellView~>cellName "lay")
            if(xrefCellViewName != "" then
                fprintf(templFile "            \"lvsLayoutFormat\":
\"GDSII\",\n")
                fprintf(templFile "            \"lvsLayoutLibraryName\":
\"%s\",\n" hiGetCurrentWindow()~>cellView~>libName)
                fprintf(templFile "            \"lvsLayoutTopCell\":
\"%s\",\n" hiGetCurrentWindow()~>cellView~>cellName)
                fprintf(templFile "            \"lvsLayoutView\":
\"%s\",\n" xrefCellViewName)
                fprintf(templFile "            \"lvsLayoutFile\":
\"%s%s.gds\",\n" hiGetCurrentWindow()~>cellView~>cellName, fileNameSuf)
                fprintf(templFile "            \"lvsExpFromLayViewer\":
true,\n")
            )
        )
    )
    fprintf(templFile "            \"startVueAfterRun\": true,\n")
    fprintf(templFile "            \"runType\": \"%s\"\n" type)
    fprintf(templFile "        }\n")
    fprintf(templFile "    }\n")
    fprintf(templFile "}\n")
    close(templFile)
    ; return(t) ;;; Commented as
          ;;; of ICV 2021.06-SP3
    return(templFilePath)       ;;; Return Template Filename
                                ;;; as of ICV 2021.06-SP3
))
; end procedure
```

Feedback

Export Triggers are called when ICV Launch is set to export Layout or Schematic data from within a design environment. By default, each of the integrated design environments use the standard utilities to perform the data export.

Example 9 IC Validator WorkBench Export Trigger

```
#
# IC Validator Workbench / ICV Launch Export Layout Trigger
#
proc exportLayout {format path log lib view cell templ} {
# error-out if there is no active layout
    set lId [layout active]
    if {$lId == ""} {
        echo -error "No active layout"
        return 1
    }
# if the export-filename == current filename, then just save the layout
    set filename [layout filename $lId]
    if {[file normalize $filename] == [file normalize $path]} {
        set res [catch {layout save $path -rename $path -libname $lib}
 err]
        if { $res != 0 } {
            echo -error $err
        }
        return $res
    }
# otherwise - extract the provided cell
# log lib cell view templ parameters are not used and added for
 compatibility
    set format [string tolower $format]
# supported formats are oasis and gds only
    if {$format != "oasis"} {set format "gds"}
    set res [catch {layout extract $path -format $format -cell $cell}
 err]
    if { $res != 0 } {
        echo -error $err
    }
    return $res
}
# end procedure
```

Example 10 Custom Compiler Export Trigger

```
#
# Custom Compiler / ICV Launch Export Layout Trigger
#
proc exportLayout {t_format t_outputFile t_logFile t_lib t_view t_cell
 t_tmplFile { t_advConfig false }} {
        set retVal [::xt::icv::launch::exportLayout $t_format
 $t_outputFile $t_logFile \
                                                $t_lib $t_view $t_cell
 $t_tmplFile $t_advConfig]
        return $retVal
```

Feedback

```
}
# end procedure

#
# Custom Compiler / ICV Launch Export Schematic Trigger
#
proc exportSchematic {t_format t_outputFile t_logFile t_lib t_view t_cell
 t_tmplFile { t_advConfig false }} {
        set retVal [::xt::icv::launch::exportSchematic $t_format
 $t_outputFile $t_logFile \
                                                $t_lib $t_view $t_cell
 $t_tmplFile $t_advConfig]
        return $retVal
}
# end procedure
```

Example 11 Cadence Virtuoso Export Trigger

```
;
; Cadence Virtuoso / ICV Launch Export Layout Trigger
;
procedure(exportLayout(t_format t_outputFile t_logFile t_lib t_view
 t_cell t_tmplFile)
prog((expResult exeTool l_winID winID tCellView tmplFileCmd errorMsg)
    _vueDebugMsg("Export Layout to file: %s " t_outputFile)
    _vueDebugMsg("The export flow log file: %s " t_logFile)
    _vueDebugMsg("The template file: %s " t_tmplFile)
    if(isFile(t_logFile) then
        sh(strcat("rm " t_logFile ))  ; remove previous log file if it
 exist.
    )
    exeTool = "";
    outFileOption = ""
    if(t_format == "OASIS" then
        exeTool = "oasisout"
        outFileOption = "-oasisFile"
    )
    if(t_format == "GDSII" then
        exeTool = "strmout"
        outFileOption = "-strmFile"
    )

    if(exeTool == "" then
        _vueErrorMsg("The input format is invalid: %s" t_format)
        sprintf(errorMsg "echo \"Error: The input format is invalid:
 %s.\" > %s" t_format t_logFile)
        return(nil)
    )
    if((t_lib == "" || t_view == "" || t_cell == "") then
        ; Get the first layout window from all open windows
        l_winID = hiGetWindowList()
        foreach(winID l_winID
            _vueDebugMsg("Access window ID: %s " winID)
```

Feedback

```
            if(hiGetAppType(winID) == "Layout" then
                tCellView = geGetWindowCellView(winID)
                ; Get lib cell view info
                t_lib = tCellView~>libName
                t_cell = tCellView~>cellName
                t_view = tCellView~>viewName
            )
        )
        if((t_lib == "" || t_view == "" || t_cell ==="") then
            ; If there is no layout window, then do not export and return
 error.
            _vueErrorMsg("There is no layout application window.")
            sprintf(errorMsg "echo \"Error: there is no layout
 application window.\" > %s" t_logFile)
            sh(errorMsg)
            return(nil)
        )
    )
    _vueDebugMsg("Export layout from Library: %s, Cell: %s, View: %s in
 Format: %s\n" t_lib t_cell t_view t_format)
    tmplFileCmd = ""
    if(t_tmplFile != "" then
        ; Template file exists
        _vueDebugMsg("The template file for export flow: %s" t_tmplFile)
        sprintf(tmplFileCmd "-templateFile %s", t_tmplFile)
    )
    sprintf(cmd "%s -library %s -topCell %s -view %s %s %s %s > %s",
 exeTool t_lib t_cell t_view outFileOption t_outputFile tmplFileCmd
 t_logFile)
    _vueDebugMsg("Export layout command: %s" cmd)
    expResult = sh(cmd)
    ; Return with exec result
    return(expResult)
))
; end procedure

;
; Cadence Virtuoso / ICV Launch Export Layout Trigger
;
procedure(exportSchematic(t_format t_outputFile t_logFile t_lib t_view
 t_cell t_tmplFile)
prog((expResult l_winID winID tCellView  lib cell view cdlFile runDir cmd
 errorMsg siEnvFile)
    _vueDebugMsg("Export schematic to file: %s " t_outputFile)
    _vueDebugMsg("The export flow log file: %s " t_logFile)
    if(isFile(t_logFile) then
        sh(strcat("rm " t_logFile ))  ; remove previous log file if it
 exist.
    )
    if((t_lib == "" || t_view == "" || t_cell ==="") then
        ; Get the first layout window from all open windows
        l_winID = hiGetWindowList()
        foreach(winID l_winID
```

Feedback

```
            _vueDebugMsg("Access window ID: %s " winID)
            if(hiGetAppType(winID) == "Schematics" then
                tCellView = geGetWindowCellView(winID)
                ; Get lib cell view info.
                t_lib = tCellView~>libName
                t_cell = tCellView~>cellName
                t_view = tCellView~>viewName
            )
        )
        if((t_lib == "" || t_view == "" || t_cell == "") then
            ; If there is no schematic window, then do not export and
return error.
            _vueErrorMsg("There is no schematic application window.")
            sprintf(errorMsg "echo \"Error: There is no schematic
application window.\" > %s" t_logFile)
            sh(errorMsg)
            return(nil)
        )
    )
    _vueDebugMsg("Export schematic from Library: %s, Cell: %s, View: %s
in Format: %s\n" t_lib t_cell t_view t_format)
    ; Get output file name.
    cdlFile =  car(reverse(parseString(t_outputFile "/")))
    _vueDebugMsg("Export schematic file name: %s" cdlFile)
    ; Get rundir from output file path
    runDir = substring(t_outputFile 1 (strlen(t_outputFile) -
strlen(cdlFile)))
    ; Generate si.env file in current dir for si tool
    siEnvFile = strcat(runDir "/si.env")

    if(genSiFileFromTemplate(t_lib t_cell t_view cdlFile siEnvFile
t_tmplFile) then
        sprintf(cmd "si %s -batch -command netlist > %s" runDir
t_logFile)
        _vueDebugMsg("Export schematic command: %s" cmd)
        expResult = sh(cmd)
        ; Return with exec result (we only export data from first
schematic window)
        return(expResult)
    else
        ; Something wrong while generating si.env file.
        _vueErrorMsg("Something wrong while generating si.env file.")
        sprintf(errorMsg "echo \"Error: Something wrong while generating
si.env file.")
        sh(errorMsg)
        return(nil)
    )
))
; end procedure
```

B

Example Customization Scripts

This appendix contains examples of DRC and LVS customization scripts.

Example 12 DRC Customization Example

```
[
    {
        "type": "TAB",
        "prompt": "DRC Main"
    },
        {
            "type": "LABEL",
            "prompt": "DRC Process Option Settings"
        },
        {
                "type": "SEPARATOR"
        },
        {
            "type": "DEFINE",
            "name": "DRC_CHECK_TYPE",
            "choices": [
                {
                    "value": "0",
                    "prompt": "FEOL"
                },
                {
                    "value": "1",
                    "prompt": "ALL"
                },
                {
                    "value": "3",
                    "prompt": "BEOL"
                }
            ],
            "initval": "1",
            "boolean": 0,
            "select": 1,
            "listbox": 1,
            "prompt": "Select the DRC Check Type: "
        },
        {
            "type": "DEFINE",
            "name": "DRC_CHIP_LEVEL_RUN",
            "boolean": 1,
```

Feedback

```
            "select": 0,
            "prompt": "Run Top-Level Chip: "
        },
        {
            "type": "LABEL",
            "prompt": "DRC Check Selection Settings"
        },
        {
                "type": "SEPARATOR"
        },
        {
            "type": "DEFINE",
            "name": "d_SHOW_ERRORS_FLAG",
            "boolean": 1,
            "select": 0,
            "master": "!DRC_CHIP_LEVEL_RUN",
            "prompt": "Show M1 & M3 Check Errors: "
        },
        {
            "type": "GROUP",
            "prompt": "Do NOT Run: ",
            "master": "DRC_CHIP_LEVEL_RUN",
            "children": [
              {
                "type": "DEFINE",
                "name": "d_NO_CONNECTIVITY",
                "boolean": 1,
                "select": 0,
                "prompt": "Connectivity Checks: "
              },
              {
                "type": "DEFINE",
                "name": "d_NO_DENSITY",
                "boolean": 1,
                "select": 0,
                "prompt": "Density Checks: "
              },
              {
                "type": "DEFINE",
                "name": "d_SKIP_LATCH_UP_CHECKS",
                "boolean": 1,
                "select": 0,
                "prompt": "Skip Latch-Up Checks: "
              }
            ]
        },
    {
        "type": "TAB",
        "prompt": "DRC User Options"
    },
        {
            "type": "LABEL",
            "prompt": "DRC Customer Option Settings"
```

Feedback

```
            },
            {
                    "type": "SEPARATOR"
            },
            {
                "type": "DEFINE",
                "name": "OVERRIDE_FILE",
                "boolean": 1,
                "select": 0,
                "prompt": "Use the Override File?"
            },
            {
                "type": "VARIABLE",
                "name": "drcMaxErrorsPerCheck",
                "boolean": 0,
                "select": 0,
                "choices": ["Error Sampling", "Error Limit"],
                "initval": "Error Sampling",
                "listbox": 1,
                "prompt": "DRC Max Error Filter Type : "
            },
            {
                "type": "RUNSET",
                "name": "opt_drcErrorReportTop",
                "boolean": 1,
                "select": 0,
                "pxl": "error_options(\n    report_layout_errors = TOP\n);",
                "prompt": "Output TOP_LAYOUT_ERRORS file?: "
            }
    ]
```

Example 13 LVS Customization Example

```
    [
        {
            "type": "TAB",
            "prompt": "LVS Main"
        },
            {
                "type": "LABEL",
                "prompt": "LVS Process Option Settings"
            },
            {
                    "type": "SEPARATOR"
            },
            {
                "type": "DEFINE",
                "name": "RUN_SOFT_CONNECT_CHECKS",
                "boolean": 1,
                "select": 1,
                "prompt": "Run Soft-Connect Checks: "
            },
            {
```

Feedback

```
                "type": "DEFINE",
                "name": "RUN_SOFT_CHECKS",
                "boolean": 1,
                "select": 1,
                "prompt": "Run Soft Checks: "
            },
            {
                "type": "DEFINE",
                "name": "RUN_ERC_CHECKS",
                "boolean": 1,
                "select": 0,
                "prompt": "Run ERC Checks: "
            },
        {
            "type": "TAB",
            "prompt": "LVS User Options"
        },
            {
                "type": "LABEL",
                "prompt": "LVS Customer Option Settings"
            },
            {
                    "type": "SEPARATOR"
            },
            {
                "type": "DEFINE",
                "name": "EXTRACT_AD_AS",
                "boolean": 1,
                "select": 0,
                "prompt": "Extract PEX Properties?: "
            },
            {
                "type": "DEFINE",
                "name": "PERC_RUN",
                "boolean": 1,
                "select": 0,
                "prompt": "Run for PERC?: "
            }
    ]
```

C

IC Validator Installation

This appendix provides steps for installing IC Validator on a linux64-bit operating system. The commands used are in a TCSH terminal-line environment. See the following sections:

- Preinstallation Setup
- Changing Operating System Limits
- Finding Your Site Identification Number
- Configuring and Downloading the Synopsys Installer
- Product Installation

Preinstallation Setup

To install IC Validator, create a new root directory for installation, as described in the following steps.

1. Create a new directory.

   ```
   % mkdir -p /root_dir
   ```

2. Set the permissions on the new directory.

   ```
   % chmod 755 /root_dir
   ```

3. Set the SYNOPSYS environment variable to the path of this newly created directory.

   ```
   % setenv SYNOSPYS /path_to_root_dir
   ```

This directory can now be accessed or referenced by using “$SYNOPSYS”. All other instructions use this environment variable when referring to the root directory.

Changing Operating System Limits

The Linux kernel defines hard preprocessing limits on the maximum amount of memory that can be used. These limits can cause installation processes to fail even when

the resources are available. To ensure the correct limits are set, enter the following commands:

```
% unlimit filesize
% unlimit datasize
% limit stacksize 65536
```

The command limit now shows the updated settings.

```
viado@us01lts100:/global/gtsnae/gts_ibm/viado % limit
cputime      unlimited
filesize     unlimited
datasize     unlimited
stacksize    65536 kbytes
coredumpsize 0 kbytes
memoryuse    unlimited
vmemoryuse   unlimited
descriptors  16384
memorylocked 64 kbytes
maxproc      1000
```

Finding Your Site Identification Number

Before installing any Synopsys products, you must locate and make a note of your site identification number (ID).

You can locate your site ID in

- Your Synopsys Order Notification email
- The header information in your Synopsys license file
- The package with your Synopsys tools if you receive physical media

The site ID information is in the following format:

SYNOPSYS INC. KEY CERTIFICATE

Site ID: xxx

Provided that you entered a valid site identification number during installation, you can check it at any time by viewing your site_info file.

$SYNOPSYS/admin/license/site_info

If you are still unable to locate your site ID, contact your Synopsys sales representative. Although you need your site ID to open a support call or to register at https://solvnetplus.synopsys.com, you can install your product by using the default site identification number, which is 000. When you are prompted to enter your site ID during the installation process, you can leave the default of 000 and press Enter to continue the installation.

Configuring and Downloading the Synopsys Installer

The Synopsys Installer is used to install Synopsys product files in the `.spf` file format. The Synopsys Installer is not used to install Windows-based products, or products in other file formats. In this guide, the Synopsys Installer is obtained using the `sftp` command.

1. Create a new directory in the root directory for the Synopsys Installer.

```
% mkdir $SYNOPSYS/installer_directory
% cd $SYNOPSYS/installer_directory
```

2. In the new directory, enter

```
% sftp solvnetplus_username@eft.synopsys.com
```

3. At the end of the output text, enter the password for the SolvNetPlus account.

```
EXPORT CONTROL NOTICE: Unless notified otherwise technical data
received by Synopsys on this server shall be deemed and treated
by Synopsys as being classified under the Export Control
Classification Number EAR99 on the Commerce Control List of
the U.S. Export Administration Regulations for purposes of export
from the United States.  If the technical data is otherwise classified you
must provide alternative export control information before submitting
the data on this server.ndviado1@eft.synopsys.com's password:
```

4. The shell environment should have “sftp >” followed by any commands that are typed.

```
sftp> cd MyProducts/rev
sftp> Is installer*
```

```
sftp> ls installer*
installer_v5.0.1/SynopsysInstaller_v5.0.1.run          installer_v5.0.1/checksum_info.txt
installer_v5.0.1/installer_INSTALL_README.txt          installer_v5.0.1/snps_container_v1.1.bin.README
installer_v5.0.1/snps_container_v1.1.bin.csh           installer_v5.0.1/snps_container_v1.1.bin.part00
installer_v5.0.1/snps_container_v1.1.bin.part01
```

5. Locate the installer version and change into its directory.

```
sftp> cd installer_version
```

6. Download the run file for Synopsys Installer.

```
sftp> get SynopsysInstaller_version.run
```

```
sftp> get SynopsysInstaller_v5.0.1.run
Fetching /MyProducts/rev/installer_v5.0.1/SynopsysInstaller_v5.0.1.run to SynopsysInstaller_v5.0.1.run
/MyProducts/rev/installer_v5.0.1/SynopsysInstaller_v5.0.1.run                100%   69MB  34.4MB/s   00:02
```

7. Exit the sftp environment.

```
Sftp> quit
```

8. Give the run file executive rights.

Feedback

% chmod 755 SynopsysInstaller_*version.run*

9. Run the Synopsys Installer File.

```
% ./SynopsysInstaller_version.run
```

```
viado@us01lts100:/global/gtsnae/gts_ibm/viado/SNPS_ROOT/SNPS_INSTALL % ./SynopsysInstaller_v5.0.1.run

 #*****************************************************************
 #
 #    Synopsys Installer Self-Extracting Executable
 #
 #    This script extracts and installs Synopsys Installer 5.0.1 into
 #    the given directory
 #
 #    For help type:
 #
 #       SynopsysInstaller_v5.0.1.run -help
 #
 #*****************************************************************

Please specify installation directory [.]: .
Installing Synopsys Installer 5.0.1 into the directory '/global/gtsnae/gts_ibm/viado/SNPS_ROOT/SNPS_INSTALL'...

  Unpacking: SynopsysInstaller.tgz ...
  Unpacking: singularity.tar.gz ...

Installation complete.
```

10. Add the Synopsys Installer directory to system path.

```
% set path=($SYNOPSYS/installer_dir$path)
```

Product Installation

Now that the Synopsys Installer is set up, you can download the IC Validator installation files from SolvNetPlus.

To install IC Validator:

1. Create a temporary directory for the IC Validator installation files.

```
% mkdir $SYNOPSYS/tmpfiles_dir
% mkdir $SYNOPSYS/tmpfiles_dir
```

2. Use the `sftp` command to access the SolvNetPlus directory.

```
% sftp solvnetplus_username@eft.solvnet.com
```

Enter the password.

3. The shell environment should have “sftp >” followed by any commands typed in

```
sftp> cd MyProducts/rev
sftp> ls icvalidator*
```

4. Find the version number and change into the IC Validator directory.

```
sftp> Cd icvalidator_vP-version
```

5. Download the product files.

```
sftp> Get*
```

```
sftp> get *
Fetching /MyProducts/rev/icvalidator_vP-2019.06/checksum_info.txt to checksum_info.txt
/MyProducts/rev/icvalidator_vP-2019.06/checksum_info.txt                    100%  502     0.5KB/s   00:00
Fetching /MyProducts/rev/icvalidator_vP-2019.06/icvalidator_INSTALL_README.txt to icvalidator_INSTALL_README.t
t
/MyProducts/rev/icvalidator_vP-2019.06/icvalidator_INSTALL_README.txt        100%   13KB  12.8KB/s   00:00
Fetching /MyProducts/rev/icvalidator_vP-2019.06/icvalidator_vP-2019.06_LINUX.64.spf.README to icvalidator_vP-20
19.06_LINUX.64.spf.README
/MyProducts/rev/icvalidator_vP-2019.06/icvalidator_vP-2019.06_LINUX.64.spf.R 100%  905     0.9KB/s   00:00
Fetching /MyProducts/rev/icvalidator_vP-2019.06/icvalidator_vP-2019.06_LINUX.64.spf.csh to icvalidator_vP-2019.
06_LINUX.64.spf.csh
/MyProducts/rev/icvalidator_vP-2019.06/icvalidator_vP-2019.06_LINUX.64.spf.c 100% 2970     2.9KB/s   00:00
Fetching /MyProducts/rev/icvalidator_vP-2019.06/icvalidator_vP-2019.06_LINUX.64.spf.part00 to icvalidator_vP-20
19.06_LINUX.64.spf.part00
/MyProducts/rev/icvalidator_vP-2019.06/icvalidator_vP-2019.06_LINUX.64.spf.p 100% 1900MB  23.5MB/s   01:21
Fetching /MyProducts/rev/icvalidator_vP-2019.06/icvalidator_vP-2019.06_LINUX.64.spf.part01 to icvalidator_vP-20
19.06_LINUX.64.spf.part01
/MyProducts/rev/icvalidator_vP-2019.06/icvalidator_vP-2019.06_LINUX.64.spf.p 100% 1021MB  26.2MB/s   00:39
Fetching /MyProducts/rev/icvalidator_vP-2019.06/icvalidator_vP-2019.06_LINUXPPC.64.spf to icvalidator_vP-2019.0
6_LINUXPPC.64.spf
/MyProducts/rev/icvalidator_vP-2019.06/icvalidator_vP-2019.06_LINUXPPC.64.sp 100% 1852MB  23.2MB/s   01:20
Fetching /MyProducts/rev/icvalidator_vP-2019.06/icvalidator_vP-2019.06_common.spf to icvalidator_vP-2019.06_com
mon.spf
/MyProducts/rev/icvalidator_vP-2019.06/icvalidator_vP-2019.06_common.spf     100%   80MB  39.8MB/s   00:02
```

6. Exit the sftp environment.

```
sftp> quit
```

7. Give the downloaded files more permissions.

```
% chmod 755*
```

```
total 4988236
-rwxr-xr-x 1 viado synopsys        502 Jul 29 13:55 checksum_info.txt
-rwxr-xr-x 1 viado synopsys      13078 Jul 29 13:55 icvalidator_INSTALL_README.txt
-rwxr-xr-x 1 viado synopsys        905 Jul 29 13:55 icvalidator_vP-2019.06_LINUX.64.spf.README
-rwxr-xr-x 1 viado synopsys       2970 Jul 29 13:55 icvalidator_vP-2019.06_LINUX.64.spf.csh
-rwxr-xr-x 1 viado synopsys 1992294400 Jul 29 13:56 icvalidator_vP-2019.06_LINUX.64.spf.part00
-rwxr-xr-x 1 viado synopsys 1070202880 Jul 29 13:57 icvalidator_vP-2019.06_LINUX.64.spf.part01
-rwxr-xr-x 1 viado synopsys 1941882880 Jul 29 13:58 icvalidator_vP-2019.06_LINUXPPC.64.spf
drwxr-xr-x 2 viado synopsys       4096 Jul 29 13:58 .
-rwxr-xr-x 1 viado synopsys   83476480 Jul 29 13:58 icvalidator_vP-2019.06_common.spf
drwxr-xr-x 4 viado synopsys       4096 Jul 29 14:06 ..
```

8. Navigate to the Synopsys root directory.

```
% cd $SYNOPSYS
```

9. To start the installation process.

```
% installer
```

10. Provide the path to the temporary files directory when prompted.

```
./tmpfiles_dir
```

```
viado@us01lts100:/global/gtsnae/gts_ibm/viado/SNPS_ROOT % installer

                Synopsys (R) Installer
                    Version 5.0.1

Copyright (c) 2005 - 2019 Synopsys, Inc.
This software and the associated documentation are proprietary to Synopsys, Inc.
This software may only be used in accordance with the terms and conditions of a
written license agreement with Synopsys, Inc. All other use, reproduction, or
distribution of this software is strictly prohibited.

Enter the path to the source directory containing the downloaded EFT file(s) [/global/gtsnae/gts_ibm/viado/SNPS
_ROOT]:./tmp_files
```

11. Provide the path for the IC Validator install location. If the directory does not exist, it is created.

```
./ICV_Install_dir
```

```
Enter the full path to the directory where you want to
install Synopsys products. If the directory does not exist,
it will be created. [/usr/synopsys]: ./ICV
```

12. Enter in the Site ID number when prompted.

```
Site_ID_#
```

```
Creating EST file icvalidator_vP-2019.06_LINUX.64.spf....
Creating EST file icvalidator_vP-2019.06_LINUX.64.spf successfully
Extracting release information from icvalidator_vP-2019.06_LINUXPPC.64.spf
Extracting release information from icvalidator_vP-2019.06_LINUXPPC.64.spf successfully
Extracting release information from icvalidator_vP-2019.06_common.spf
Extracting release information from icvalidator_vP-2019.06_common.spf successfully
Extracting release information from icvalidator_vP-2019.06_LINUX.64.spf
Extracting release information from icvalidator_vP-2019.06_LINUX.64.spf successfully

Your site ID number is in the upper-right corner of your Synopsys license
key certificate. If you have trouble locating it, contact your Synopsys
representative.

Site ID number [000]: 000
```

13. Accept the installation.

```
Yes
```

```
#####################################################
############### Installation Summary ################
Product:    ICVALIDATOR (icvalidator)
Release:    P-2019.06
Platforms:   LINUXPPC.64 common LINUX.64
Disk Space: 7497 MB
Target Dir: /global/gtsnae/gts_ibm/viado/SNPS_ROOT/ICV/icvalidator/P-2019.06

Licensed Products communicate with Synopsys servers for the purpose of providing software updates,
detecting software piracy and verifying that customers are using Licensed Products in conformity with
the applicable License Key for such Licensed Products. Synopsys will use information gathered in
connection with this process to deliver software updates and pursue software pirates and infringers.
#####################################################
If the information is correct, continue with the installation.
Accept, Install? [yes]: yes
```

14. Wait for installation to complete.

```
INFO: Prepare for installing P-2019.06 release of product icvalidator ...
INFO: Extracting LINUXPPC.64 package from icvalidator_vP-2019.06_LINUXPPC.64.spf ...
INFO: Extracting common package from icvalidator_vP-2019.06_common.spf ...
INFO: Extracting LINUX.64 package from icvalidator_vP-2019.06_LINUX.64.spf ...
INFO: Verify SHA256 sum for icvalidator.taz ....
INFO: Verify SHA256 sum for linuxppc.64/icvalidator.taz ....
INFO: Verify SHA256 sum for linux.64/icvalidator.taz ....
INFO: Installing P-2019.06 release of product icvalidator to /global/gtsnae/gts_ibm/viado/SNPS_ROOT/ICV/icvali
ator/P-2019.06 ....
INFO: Installing common package for P-2019.06 release of product icvalidator ...
INFO: Installing LINUXPPC.64 package for P-2019.06 release of product icvalidator ...
INFO: Installing LINUX.64 package for P-2019.06 release of product icvalidator ...
INFO: Installation for P-2019.06 release of product icvalidator has finished successfully.


Synopsys tools require that a supported version of Synopsys Common
Licensing (SCL) be installed and serving the necessary licenses.
For information on how to obtain SCL, or your license key file,
see http://www.synopsys.com/licensing

For any postinstallation setup requirements, see the product-specific
chapters in the Installation Guide at http://www.synopsys.com/install

INFO: See installer log file /global/gtsnae/gts_ibm/viado/SNPS_ROOT/installer.log
```

15. Delete the temporary files used to install IC Validator.

```
% rm -rf $SYNOPSYS/tmpfiles_dir
```

16. Set the ICV_HOME_DIR environment variable.

```
% setenv ICV_HOME_DIR $SYNOPSYS/ICV_Install_Dir/icvalidator/version
```

```
% setenv ICV_HOME_DIR $SYNOPSYS/ICV/icvalidator/P-2019.06
```

17. Add the IC Validator bin to the system's path.

Feedback

```
% set path=($ICV_HOME_DIR/bin $path)
```

```
% set path=($ICV_HOME_DIR/bin $path)
```

18. Set the license environment variable.

```
% setenv SNPSLMD_LICENSE_FILE port@hostname
```

19. To ensure IC Validator was configured correctly.

```
% icv
```

IC Validator should output its version number, operating system, and optional command-line options. However, since no other files or options were specified in the IC Validator command, it will not complete.

```
/SNPS_ROOT % icv

                    IC Validator

     Version P-2019.06 for linux64 - May 17, 2019 cl#4828830

            Copyright (c) 1996 - 2019 Synopsys, Inc.
   This software and the associated documentation are proprietary to Synopsys,
 Inc. This software may only be used in accordance with the terms and conditions
 of a written license agreement with Synopsys, Inc. All other use, reproduction,
     or distribution of this software is strictly prohibited.

Called as: icv

      -zn                       Compress the generated schematic and layout
                                netlists
      -inter_die                Enable inter-die DRC/LVS checking.
      -64                       Use 64 bit coordinates.
      -hlic                     Force shared licensing scheme.
      -ilic                     Force native licensing scheme.
      -ulic                     Force unified licensing scheme.
      -filecache                Stores temporary files in memory to reduce
                                disk usage
      -time_limit <hours>       User specified time limit in hours (minimum 1
                                hour)
      <runsetfile>              IC Validator runset.
IC Validator did not complete.
```

To install ICV WorkBench, follow the steps you used for installing IC Validator. The same Synopsys Installer can be used to install the ICV Workbench `.spf` files. You can retrieve these files using the `sftp` command.

Instead of changing to the MyProducts > rev > revicvalidator_vP-version directory, navigate to the MyProducts > rev_o > icwb_ev_plus_version directory.

There are multiple ways to install Synopsys products that are not mentioned in this User Guide. For more information regarding these methods, see Downloads in https://solvnetplus.synopsys.com

D

Environment Variable Setup

This appendix provides information on setting up runset caching, linking VUE to ICV WorkBench, and creating a .cshrc file for startup scripts in the following sections:

- Runset Caching
- Configuring Runset Caching Environment Variables
- Command-Line Option Compatibility
- Creating or Modifying a .cshrc File
- Installing IC Validator Integration in Cadence Virtuoso

Runset Caching

Runset caching significantly decreases the time of runset parsing. When the IC Validator tool writes a runset cache file, the following messages are displayed indicating the path of the saved cache file.

```
Saved runset to cache:                    /.icvrscache/
e3b32d5de337d888106672bbba370041-rod-3908.rscache
Parsing Finished
Runset Compile Time=0:00:03 User=3.37 Sys=0.16 Mem=0.132 GB
```

When the IC Validator tool loads a runset cache file, the following message shows that the path of the cache file was read.

```
Loaded runset from cache:
                    /.icvrscache/
e3b32d5de337d888106672bbba370041-rod-3908.rscache
Cached Runset Compile Time=0:00:00 User=0.25 Sys=0.04 Mem=0.108 GB
```

Configuring Runset Caching Environment Variables

The four main environment variables associated with runset caching are `ICV_RUNSET_CACHE_DIR`, `ICV_RUNSET_CACHE_DIR_SHARED`, `ICV_RUNSET_CACHE_MAXFILES`, and `ICV_RUNSET_CACHE_MAXSIZE`.

- To set the local runset cache directory, enter

  ```
  %setenv ICV_RUNSET_CACHE_DIR directory
  ```

  This environment variable sets the local directory used by the IC Validator tool. IC Validator uses the cache file if it matches the specified runset. If there is no match, the IC Validator creates a new cache file in the local directory. The default local directory is ~/.icvrscache.

- To set the shared runset cache directory, enter

  ```
  % setenv ICV_RUNSET_CACHE_DIR_SHARED directory
  ```

  The shared directory is not user specific and allows multiple users to point to the same shared cache directory. If there is no match in the shared directory, IC Validator searches the local directory.

- To limit the number of runset cache files, enter

  ```
  % setenv ICV_RUNSET_CACHE_MAXFILES num_files_to_keep
  ```

  The IC Validator tool does not delete any preexisting runset cache files until either the`ICV_RUNSET_CACHE_MAXFILES` or the `ICV_RUNSET_CACHE_MAXSIZE` limit is reached. When either limit is reached, the IC Validator tool deletes files in the least-recently used order; that is, cache files that have been loaded recently are retained while the older files are deleted.

- To limit the file size of the cache file, enter

  ```
  % setenv ICV_RUNSET_CACHE_MAXSIZE max_size
  ```

  The *max_size* value is a numeric value followed by an optional letter indicating the unit type. You can use a unit type of K for kilobytes, M for megabytes, G for gigabytes, or T for terabytes. If no unit type is used, the *max_size* value is assumed to be in megabytes. The default size for runset cache file is 200 MB.

Command-Line Option Compatibility

When using the same runset file, certain command-line options cause the cache file to miss or match. Other command-line options disable runset caching completely. See Table 3.

Table 3 Command-Line Option Compatibility

Matched	Missed	Disabled
`-c`	`-64`	`-checkpoint`

Table 3 Command-Line Option Compatibility (Continued)

Matched	Missed	Disabled
-g	-D	-debug
-i	-U	-debug_source
-p	-il	-ndg
-f	-iln	
-rd	-sfn	
-stc	-svc	
	-svn	
	-ufn	
	-uvc	
	-uvn	
	-verbose	

Creating or Modifying a .cshrc File

The IC Validator environment variables are reset every time a terminal is open or closed. To avoid typing in values for each of these variables every time there is an open terminal, create a .cshrc file. Upon startup, the terminal automatically executes each command written in the .cshrc file.

An example of a .cshrc file might have the following lines of code:

```
#IC Validator Variables
setenv SYNOPSYS /home_dir
setenv ICV_HOME_DIR /install_dir
set path=($SYNOPSYS/installer_dir $path)
setenv SNPSLMD_LICENSE_FILE port@hostname
#Runset Caching Variables
setenv ICV_RUNSET_CACHE_MAXSIZE filesize
setenv ICV_RUNSET_CACHE_MAXFILES num_files
setenv ICV_RUNSET_CACHE_DIR /local_dir
setenv ICV_RUNSET_CACHE_DIR_SHARED /shared_dir
```

With the following .cshrc file, the terminal is automatically set up to run the IC Validator tool.

The .cshrc file is typically found in the your home directory. If the file already exists, simply add the required commands at the end of its existing text. If the .cshrc file does not exist

1. Open any text editor (VIM, gedit, emacs, and so on).
2. Type in the required commands to be executed at startup.
   a. Set IC Validator licensing and home directory.
   b. Configure runset caching variables.
   c. Link ICV Workbench and VUE.
3. Save the file.
   a. If a .cshrc file exists but is read-only, save the file as .cshrc.system_username.
   b. If a .cshrc file does not exist, save the file as .cshrc.
4. Place the .cshrc files in the your home directory.

Installing IC Validator Integration in Cadence Virtuoso

The following example shows a .cdsinit file used to test for ICV installation and to load the integration.

```
1 ; .cdsinit file
2 ; 02/09/2020
3
4 ;
5 ;   This is a .cdsinit example for Synopsys ICV customers.
6 ;
7
8 ; Check for Synopsys Installation Variables
9 if( getShellEnvVar("ICV_HOME_DIR")==nil then
10   print("*** ERROR *** Synopsys ICV Installation NOT found.")
11
12   hiDisplayAppDBox(
13     ?name           'ICVHOMEErrorDlg
14     ?dboxBanner     "ICV Error"
15     ?dboxText       "Synopsys ICV Interface not loaded."
16     ?dialogType     hicErrorDialog
17     ?dialogStyle    'modal
18     ?buttonLayout   'Close
19   )
20 else
21 ; Check for Synopsys ICV Instalation Directory
22   if( isDir(getShellEnvVar("ICV_HOME_DIR"))==nil then
23     printf("*** DBG ERROR *** ICV Instalation Directory NOT Found : %s\n" getShellEnvVar("ICV_HOME_DIR"))
24   else
25     printf("--- Found ICV Instalation Directory: %s\n" getShellEnvVar("ICV_HOME_DIR"))
26
27   ; Set ICV Launch Envirinment Variable
28     setShellEnvVar("ICV_LAUNCH=1")
29
30   ; Load Synopsys ICV Environment
31     printf("--- Loading the Synopsys ICV Interface\n")
32     load(strcat(getShellEnvVar("ICV_HOME_DIR") "/etc/VUE/SkillVueMenu.il"))
33
34   ; Enter other startup procedures...
35
36   ) ; endif
37 ) ; endif
```