PLUNIFY

Scripting with TCL in InTime

AN:PIN003

Introduction

This application note describes and provides an example of how to develop and run custom Tcl scripts to automate the InTime software. There are several ways of running InTime; some users like to use the graphical user interface and others prefer command-line scripting. Advanced users can create custom Tcl scripts to automatically try different InTime Recipes and just keep InTime running optimizations in the background.

After following the steps in this application note, you should be able to modify the example script for your needs.

Running Tcl in InTime

InTime provides a Tcl Console for you to enter standard as well as InTime-specific Tcl commands. The Tcl Console is located at the bottom right-hand corner of InTime GUI as shown in Figure 1.

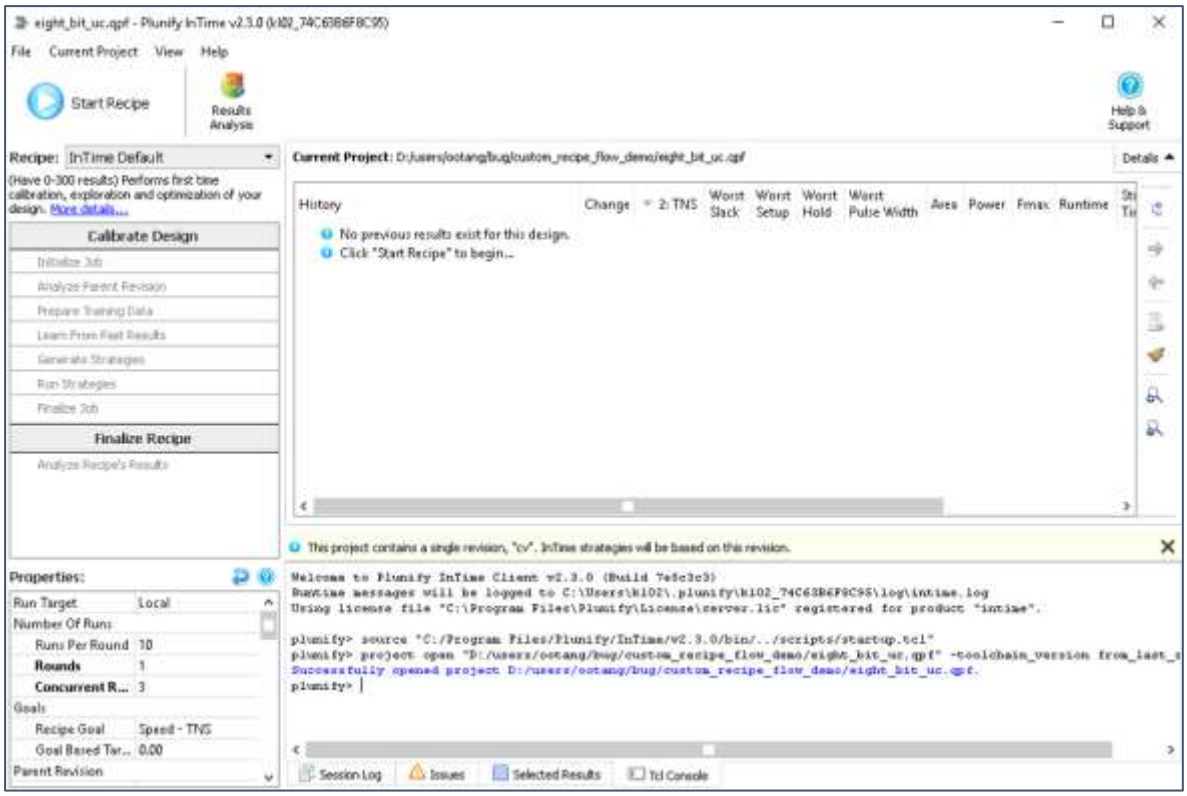


Figure 1. InTime GUI

The Tcl commands provided by InTime are documented in (http://www.plunify.com/docs/intime/flow_properties.html) or you can type `help` in the Tcl Console as shown in Figure 2 to display the available commands.

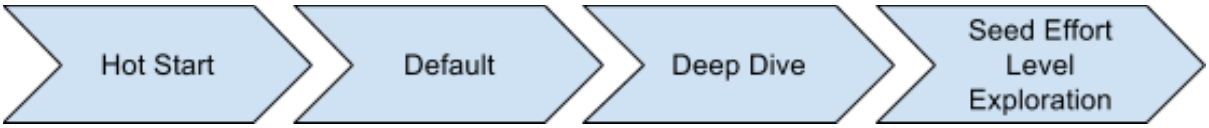
```
Successfully opened project D:/users/ootang/bug/custom_recipe_flow_demo/eight_bit_uc.qpf.
plunify> help
Commands :
==========
exit:        Exit InTime
flow:        Control and execute the InTime flow
flow_steps:  Allows execution of individual steps in the InTime flow
help:        Displays this help message
history:     Shows the commands history
job:         Operations on existing jobs
license:     License management functions
log:         Log messages to the InTime session log
misc:        Miscellaneous helper functions
msgbox:      Show or get feedback from user using a message box
project:     Details and control of the open vendor project
results:     Provides details about the current results set
run_target:  Operations to configure, test and perform actions specific to different run targets
strategy:    Provides details of the currently active strategy/result
vendors:     Allows configuration of vendor tool settings
plunify>
```

Session Log | Issues | Selected Results | Tcl Console

Figure 2. List of InTime-provided Tcl commands

Example: Custom Tcl script to execute multiple recipes

This custom Tcl script automatically executes multiple InTime recipes in the order below (for Quartus).



When each recipe completes, the script sets the revision with the best timing result as the parent revision for the next recipe.

To try the sample Tcl script, download `an_pin003_autorun_multi_recipes.zip` from here: https://support.plunify.com/en/wp-content/uploads/sites/5/2017/06/autorun_multi_recipes.zip and carry out the steps below. The zip file contains:

1. `autorun_multi_recipes.tcl`
2. `eight_bit_uc_quartusii_16p0_std` folder containing a sample Quartus project
3. `eight_bit_uc_vivado_2016p4` folder containing a sample Vivado project

For the purpose of this application note, we will use the Quartus example.

Run from the InTime Tcl Console

To run the Tcl script in an InTime Tcl console,

1. Extract `an_pin003_autorun_multi_recipes.zip`

2. Start InTime and open the `<working_dir>/eight_bit_uc_quartusii_16p0_std/eight_bit_uc.qpf` project

3. Run the `autorun_multi_recipes.tcl` script at the Tcl Console:

```
source ../autorun multi recipes.tcl
```

When it finishes, you should able to see the result in Figure 3.[1] As shown, the script stops the current recipe run once it finds a revision that meets the recipe's goal, Total Negative Slack (TNS), of -2500ns for the Hot Start recipe in this example. Next, it sets that revision as the parent revision for the next recipe. This process repeats until the last recipe is run, or InTime meets the subsequent goal of TNS = 0ns.

History	Change	2: TNS	Worst Slack	Worst Setup	Worst Hold	Worst Pulse Width	Area	Power	Fmax	Runtime	Start Time
cv		-2840.72	-2.992	-2.992	0.121	0.822	91		127.1	00:00:10	2017-05-26
hot_start_1	770.66	-2070.06	-2.369	-2.369	0.091	0.807	91		135.7	00:05:17	2017-05-26
calibrate_25	2022.94	-817.775	-1.769	-1.769	0.043	0.809	91		150.76	00:06:29	2017-05-26
calibrate_13	1948.25	-892.465	-2.043	-2.043	0.034	0.807	91		144.76	00:04:19	2017-05-26
calibrate_38	1885.08	-955.644	-1.91	-1.910	0		8			00:06:41	2017-05-26
calibrate_36	1696.72	-1144	-2.018	-2.018	0.113	0.807	94		146.54	00:05:14	2017-05-26
calibrate_39	1675.49	-1165.23	-2.074	-2.074	0.023	0.816	84		145.79	00:05:29	2017-05-26
calibrate_10	1661.26	-1179.46	-2.151	-2.151	0.000	0.815	91		144.65	00:05:04	2017-05-26

Figure 3. InTime result after example script run completed

Note: *You can use* `intime.sh -help` *to find out more details about InTime command-line switches.*

Run from Command-line

To run the Tcl script in a batch script or command line,

1. Extract `an_pin003_autorun_multi_recipes.zip`

2. At command-line, change directory to `eight_bit_uc_quartusii_16p0_std` where the Quartus project is located

```
cd eight_bit_uc_quartusii_16p0_std
```

3. Run the following command at the command line

For Linux

```
<intime_installed_dir>/intime.sh -project eight_bit_uc.qpf -mode batch -
s ../autorun_multi_recipes.tcl -toolchain quartusii -toolchain_version 16.0.0 -
tclargs "-output_dir <output directory>"
```

For Windows

```
<intime_installed_dir>\bin\intime.exe -project eight_bit_uc.qpf -mode batch -
s ../autorun_multi_recipes.tcl -toolchain quartusii -toolchain_version 16.0.0 -
tclargs "-output_dir <output directory>"
```

After running the script, you should see that it starts to compile the project as shown in Figure 4a.

```
INFO: ----------------------------------------------------------------------
INFO: INITIALIZE JOB COMPLETED
INFO: after 00:00:01 (Local Job Id: 38, Design Id: 1496386665981)
INFO: ----------------------------------------------------------------------
INFO: ----------------------------------------------------------------------
INFO: ANALYZE PARENT REVISION (cv)
INFO: ----------------------------------------------------------------------
INFO: ANALYZE PARENT REVISION (1/4): Verifying state of parent revision
INFO: ----------------------------------------------------------------------
INFO:
INFO: No parent revision results. With initial compilation = "local", result extraction is required.
INFO: Successfully updated parent revision data in local database.
INFO:
INFO: ----------------------------------------------------------------------
INFO: ANALYZE PARENT REVISION (2/4): Extracting results
INFO: ----------------------------------------------------------------------
INFO:
INFO: Extracting results for parent revision (cv) from local build reports.
INFO: Build results for parent revision are incomplete. A local initial compilation will be run before attempting to ext
ract build results again.
INFO: cv : Starting (f:/38/cv.log)
INFO: cv : Synthesizing
```

Figure 4a. Output at command-line terminal after ran the example script

When the script runs finishes, it will output the results at the output directory. If you have not specify the `output_dir` option when executing the `autorun_multi_recipes.tcl`, then you should be able to see a folder named `results` generated in the project directory `<working_dir>/eight_bit_uc_quartusii_16p0_std/results` as shown in Figure 4b. Otherwise, the results will be kept at the output directory that you specified.

Under the output directory, you should see `pass` or `fail` file. If the end goal is met, you should able to see `pass` file in the output directory. Otherwise, you should see a `fail` file instead. The `best_<job_id>_<strategy_name>.tcl` script is an exported strategy Tcl script which reproduces the best timing result among the generated strategies.

Meanwhile, the folder `export_strategies_tcl` contains the exported strategy Tcl scripts of all the other strategies that are compiled successfully.

Note that the output directory is cleaned up whenever this example script is executed. Please back up this folder if necessary.

```
$ ls -R results
results:
best_job496_placement_effort_14.tcl  export_strategies_tcl/  fail  summary_result.rpt

results/export_strategies_tcl:
job492_cv.tcl                  job493_calibrate_46.tcl        job495_placement_effort_2.tcl
job492_hot_start_1.tcl         job493_calibrate_47.tcl        job495_placement_effort_20.tcl
job493_cal_speed_tns_low.tcl   job493_calibrate_49.tcl        job495_placement_effort_3.tcl
job493_calibrate_10.tcl        job493_calibrate_5.tcl         job495_placement_effort_4.tcl
job493_calibrate_13.tcl        job493_calibrate_6.tcl         job495_placement_effort_5.tcl
job493_calibrate_15.tcl        job493_calibrate_7.tcl         job495_placement_effort_6.tcl
job493_calibrate_16.tcl        job493_calibrate_8.tcl         job495_placement_effort_7.tcl
```

Figure 4b. Results directory after example script run completed

Understanding the example Tcl script

The `autorun_multi_recipes.tcl` example script is divided into five different parts:

A. Variable declaration for important information like the recipes to use, TNS goal, number of runs per rounds, etc.

B. InTime flow configuration and recipe execution.
C. Results verification to either stop or execute subsequent recipes.
D. Export strategies to Tcl scripts.
E. Summarize and print results.

Variable Declaration

Figure 5a describes what recipes to use and in what order of execution. In this example (Quartus), the order of execution is:

Hot Start -> InTime Default -> Deep Dive -> Seed Effort Level Exploration

You can modify this sequence to use different recipes or to change the order of execution.

```
# Define order of recipes to execute.
# -> Type 'flow recipes -supported' in Tcl console to show all available recipe's name
set current_toolchain [project info toolchain]
if { [string equal "$current_toolchain" "quartusii"] } {
        # Execution Order : hot_start > intime_default > deep_dive >
seeded_effort_level_exploration
        set recipes_list [list "hot_start" "intime_default" "deep_dive"
"seeded_effort_level_exploration" ]
} elseif { [string equal "$current_toolchain" "vivado"] } {
        set recipes_list [list "intime_default" "deep_dive" "vivado_explorer"
"extra_opt_exploration"]
} else {
        set recipes_list [list "intime_default"]
}
```

Figure 5a. Define recipes and their execution order

Figure 5b shows how to define the goals for Total Negative Slack(TNS), Worst Negative Slack (WNS) for each recipe, number of runs per round, number of rounds. `end_tns_goal` contains the final TNS goal. Upon reaching the final TNS goal, there can be various follow-on actions, for example generate bitstream, copy files, and so on.

The `recipe_target_result_tns(…)` defines a recipe goal that tells InTime to switch to a subsequent recipe if it meets this TNS target. Typically, the earlier goals are set at a worse level compared to the later goals.

```
# Define end goal
set end_tns_goal 0
set end_wns_goal "*" ; #Don't Care

# Define tns goal for each recipe run
set recipe_target_result_tns(hot_start) "-2500"
set recipe_target_result_tns(intime_default) "-1000"
set recipe_target_result_tns(deep_dive) "-500"
set recipe_target_result_tns(auto_placement) "0"
set recipe_target_result_tns(seeded_effort_level_exploration) "0"
set recipe_target_result_tns(vivado_explorer) "0"
set recipe_target_result_tns(extra_opt_exploration) "0"

# Define runs_per_round for each recipe run
set recipe_target_runs_p_round(hot_start) 50
set recipe_target_runs_p_round(intime_default) 10
set recipe_target_runs_p_round(deep_dive) 10
```

```
set recipe_target_runs_p_round(seeded_effort_level_exploration) 10
set recipe_target_runs_p_round(auto_placement) 10
set recipe_target_runs_p_round(vivado_explorer) 10
set recipe_target_runs_p_round(extra_opt_exploration) 10

# Define number of rounds for each recipe run
set recipe_target_rounds(hot_start) 1
set recipe_target_rounds(intime_default) 3
set recipe_target_rounds(deep_dive) 1
set recipe_target_rounds(seeded_effort_level_exploration) 2
set recipe_target_rounds(auto_placement) 1
set recipe_target_rounds(vivado_explorer) 1
set recipe_target_rounds(vivado_placement_exploration) 1
set recipe_target_rounds(extra_opt_exploration) 1
```

Figure 5b. Define end goal, recipe goal, runs per round and rounds

Flow Execution and Configuration

The InTime flow configuration and recipe execution are outlined in Figure 5c and 5d. In Figure 5c, `flow reset` is used to reset the internal flow history. It is a recommended practice to always reset the internal flow history before running any recipe.

`flow set <property> <value>` is the command to configure InTime flow settings. For example, setting `flow set control_stop_when_goal_met` to `true` enables InTime to stop the current running recipe when the goal is met. Otherwise, InTime allows the recipe to continue running even after the goal is met.

Setting `flow set control_create_bitstreams` to `true` enables bitstream files to be created for every revision. **Note:** *This takes up more time to complete each strategy*.

```
# Configure InTime Flow settings
# -> Type 'flow properties' in Tcl console to shows all the available flow property to
configure
flow reset                      ; # Reset Intime internal flow
flow restore_defaults           ; # Restore all flow property to default value
flow set run_target local       ; # Set to run strategies on local machine
flow set goal speed_tns         ; # Set goal type as speed_tns for timing optimization
flow set concurrent_runs 3      ; # Number of builds to run in parallel
flow set control_stop_when_goal_met true ; # Stop current recipe run when goal is met
flow set control_create_bitstreams false ; # Set to false to save compute time
```

Figure 5c. InTime flow configuration

To start a recipe, use the command `flow run_recipe <recipe_name>` as shown in Figure 5d. If the recipe run completes, the `flow run_recipe` command returns 0, otherwise it returns 1.

```
# Run the current recipe
if { [catch { flow run_recipe $current_recipe }] } {
        puts "ERROR: Recipe $current_recipe failed, continuing with the rest of the flow...
${::errorInfo}"
        set recipe_run_fail 1
        set return_code 1
}
```

Figure 5d. Run recipe command

Results Verification

Figure 5e shows how to verify your result. In this section, the script checks if any revision in this round meets the target goal. If yes, it stops, otherwise it continues to execute the subsequent recipes until all user-defined recipes are executed.

```
# Check if the end goal was met. Stop this script run if goal met
set job_id [flow get local_job_id]
if { $flow_continue && !$recipe_run_fail } {
        puts "INFO: Checking results in $current_recipe recipe run \(job $job_id \) "
        results clear
        results add job $job_id
        set best_revision_name [lindex [results summary best -list] 0]
        catch { strategy unset_active }
        catch { strategy set_active $best_revision_name $job_id }
        set best_revision_tns  [ strategy results -field "TNS" ]
        set best_revision_wslack [ strategy results -field "Worst Slack" ]
        puts "INFO: -> Best result in job \($job_id\) is $best_revision_name revision with
TNS = $best_revision_tns and Worst Slack = $best_revision_wslack "
        if { [is_job_met_criteria $job_id $end_tns_goal $end_wns_goal] } {
        puts "INFO: -> Goal met! .. exiting optimization"
        set flow_continue 0
        set goal_met 1
        }
}
```

Figure 5e. Verify the results of child revisions for each recipe run

Export Strategies into Tcl Scripts

Figure 5f shows how to export strategy settings for each strategy into a Tcl script. As shown in Figure 5f, the command `strategy export <export_tcl_name> -script_tcl` is used to export settings for the current strategy into a Tcl script file. In this example, the script only exports strategies that compiled successfully, using the command `results summary success -list` to obtain a list of such strategies. You must always set the "active strategy" using the command `strategy set_active <strategy_name> <job_id>` before running the `strategy export <export_tcl_name> -script_tcl` command.

```
# Export strategies settings in tcl for success revisions
results clear
catch { strategy unset_active }
set count 0
foreach id $jobs_ran {
        results add job $id
        set stratname_list_success [results summary success -list]
        set best_revname_per_job [lindex [results summary best -list] 0]
        foreach stratname $stratname_list_success {
        strategy set_active $stratname $id
        strategy export "$export_settings_tcl_dir/job${id}_${stratname}.tcl" -script_tcl
        catch { strategy unset_active }
        }
```

Figure 5f. Export strategies that compiled successful into Tcl scripts

Results Summary

Lastly, print a summary of the results. Select all the relevant results using their job IDs: *results add job <job_id>* , then print revisions that compiled successfully via *results summary success* and save the output into `<working_dir>/eight_bit_uc_quartusii_16p0_std/results/summary_result.rpt`

```
# Export summary of results in summary_result.rpt
foreach id $jobs_ran {
        results add job $id
}

set summary_result [results summary success]
if { [catch { open $summary_result_rpt w } fh] } {
        puts "ERROR: Couldn't open file: $fh"
        set return_code 1
} else {
        puts $fh "$summary_result"
        catch { close $fh }
}
```

Figure 5g. Print summary of obtained results

Conclusion

InTime provides custom Tcl scripting capabilities to enable users to automate their InTime runs. For more detailed information about the Tcl commands, please refer to the online reference at http://www.plunify.com/docs/intime/flow_properties.html.

Document Revision History

Table 1. Revision history for this application note

No	Date	Changes Made
3	01 July 2017	1. Enabled `autorun_multiple_recipes.tcl` to return 0 if the script runs OK otherwise, return 1. 2. Added option `output_dir` for autorun_multiple_recipes.tcl script to allow user to control the output directory path (Default: `<project directory>/results`).
2	06 June 2017	1. Added version 2.0 of `an_pin003_autorun_multi_recipes.zip`. In Version 2.0, the following features are added into `autorun_multi_recipes.tcl`: ● Generate `pass` or `fail` file to indicate if the end_goal was met or not. ● Export strategies for those compiled successful into Tcl scripts. ● Generate a separate Tcl script for the strategy which gave the best timing result. ● Generate a summary of results into a file named `summary_result.rpt`. 2. Added new sub-section "Export strategies into Tcl Scripts" under "Understanding the example Tcl script". 3. Corrected typo for intime.sh path. The path should be `<intime_installed_dir>/intime.sh` instead of `<intime_installed_dir>/bin/intime.sh`
1	05 June 2017	Initial version