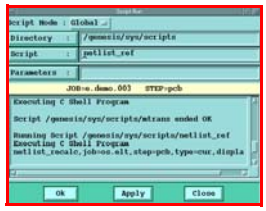
PCB SOLUTIONS

An Orbotech Mentor Graphics Company

GENESIS 2000



SCRIPTS

Software Version 10.0

Document 0210.0313
Published March 2013

Frontline PCB Solutions LP
2 Oppenheimer Street
76701 Rehovot, ISRAEL
Voice: 972-8-932-2183
Fax: 972-8-932-2186
web page: http://www.frontline-pcb.com
email: info@frontline-pcb.com

Table of Contents

Chapter 1 *Overview*

Introduction

This manual describes the scripting capabilities of Genesis 2000 which enable you to automate and customize the application. The manual describes script writing and debugging as well as how to run and bind scripts to function keys.

Each operation in the Genesis 2000 application is performed via a "line mode command" engine. When working interactively with the application, these commands are generated and can be recorded.

Line mode commands are one-line ASCII representations of the operation to be performed. This includes the command name followed by the command parameters. All fields of the command are separated by a comma (,) and each parameter is described in the format "param_name=value". The parameters can appear anywhere in the parameter list without any specific order.
For a complete set of line mode commands, please refer to the Doc.0206, Line Mode Commands.

These line mode commands can be put in files that can be run using a scripting mechanism. In order to give the scripting mechanism computational power and use of variables, the line mode commands are combined with one of a number of script programming languages such as csh or Perl. This allows you to write programs in various scripting languages and integrate line mode commands to drive the application.

The script writer can also make use of some line mode commands which supply information stored in the system as well as commands and utility programs that are used to interact with the operator while the script is running.

This manual will guide you through the various windows that control script management and describe some of the advanced scripting features and how to integrate scripts with various scripting languages.

Intended Users

This manual is intended for the system administrator and people in charge of automation. The regular user will also find some parts of this manual helpful in learning how to operate (run, record.. etc) scripting functions.

It is assumed that the operator has had minimum training on the Genesis 2000 and does not require elementary instruction. If this is not the case, it is recommended that the operator acquire some experience on the Genesis 2000 before performing the tasks described here.

It is also recommended to read the Documentation Basics (Doc. 0103) for conventions and terminology.

Scope

The "Scripts" Manual (Doc. 0204) is a part of the System Administrator book set (02). This manual, together with System Management (Doc. 0203) and The DFM Environment (Doc. 0205) are used to perform all automation and system administration tasks.

Organization of this Manual

Chapter 1 Overview - provides an overview of the use of scripts in the Genesis 2000 system, intended users, manual organization and conventions (terminology) specific to scripts.

Chapter 2 User Operation - Is a user guide to the general script operations including how to record, run, bind to keys and debug scripts within the Genesis 2000 system.

Chapter 3 Advanced Script Writing - Describes the advanced script writing features of the system with a description of the interfaces to various script programming languages.

Chapter 4 PERL Scripts - Their use in the Genesis 2000 environment.

Chapter 5 The INFO Command - A description of the INFO line mode command. This special command was created for the script writer to access information in the database.

Appendices are common to all manuals and are described in Doc.0103, Documentation Basics.

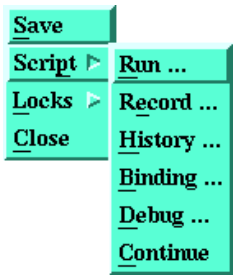
Chapter 2 *User Operations*

Accessing Script Functions

It is possible to access the scripting capability of Genesis 2000 through the main menu of the Graphic Editor, Engineering Toolkit, etc.

File > Script

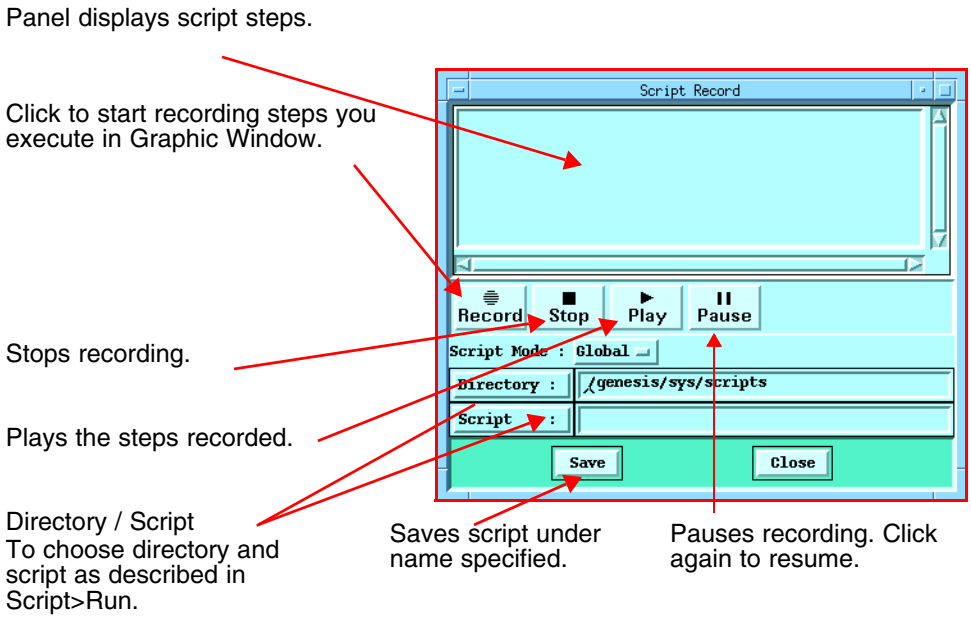
From the Main Menu, open **File**, then **Script** displays the following sub-menus:



Recording a Script

Script > Record

Displays the Script Record popup to record a new script by executing the steps:



Step 1. **Open the Graphic Editor on a step with several layers, open the Script sub menu and Click on the Record Option.**

A new window will be displayed, the Script Record popup.
Using this popup, you can define Scripts by performing the operation(s), and "recording" them.

Two Script Modes are available: Local or Global.

In the Local Mode, scripts will be saved in your local (private) directory. This mode is usually used for new, not yet debugged scripts.

In the Global Mode, you can save scripts in the global directory (scripts shared by all users).

Step 2. **Click on the Record Button.**

A red Rec sign appears (to the right). Close the Script Record Window.

Step 3. **Display a layer, Zoom in to a small area.**

Step 4. **Using the Local Edit Buttons, delete a pad, move another pad to a new location, and then copy a line to a new location.**

Note A Rec red sign appears now (window top right side), indicating that any operation performed now is recorded.

Step 5. **Open the Script Record Popup again (Alt <f>, <p>, <e>).**

In the window, you can see the Script that was just created. Within it appear all the operations that were performed in the Record action.

Step 6. **Click on the Stop Button.**

Step 7. **Close the Script Record Window, and then Undo several times, until a message: "Nothing left to undo" appears (Click OK).**

The displayed layer has returned to its original layout.

Step 8. **Open the Script Record Popup again, and Click on the Play Button.**

Your script is run again.
During a Record session, you can skip several operations, by using the Pause Button, and then resume recording by clicking again on the Record Button.

Step 9. **Save your script in local mode.**

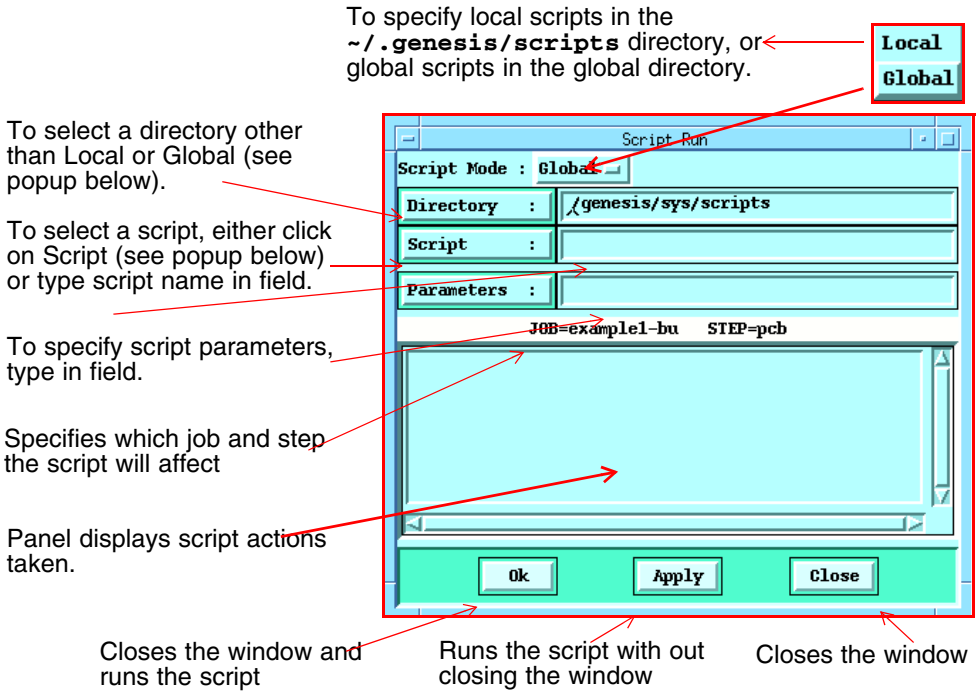
The script will be saved in your **`~/.genesis/scripts`** directory with the name you gave it.

Note You can edit your Script by using any external text editor.

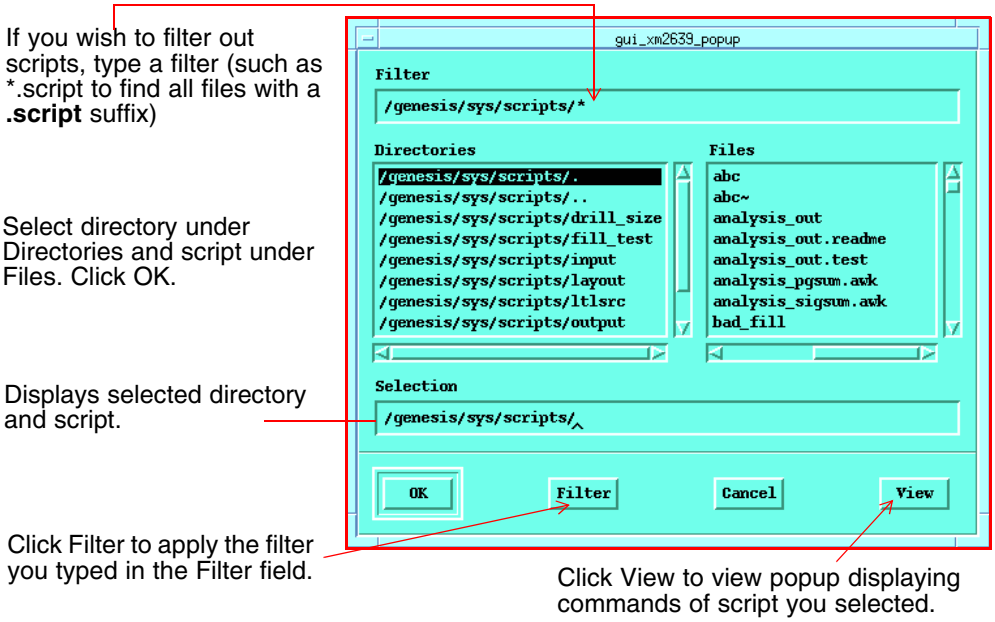
Running a Script

Script > Run

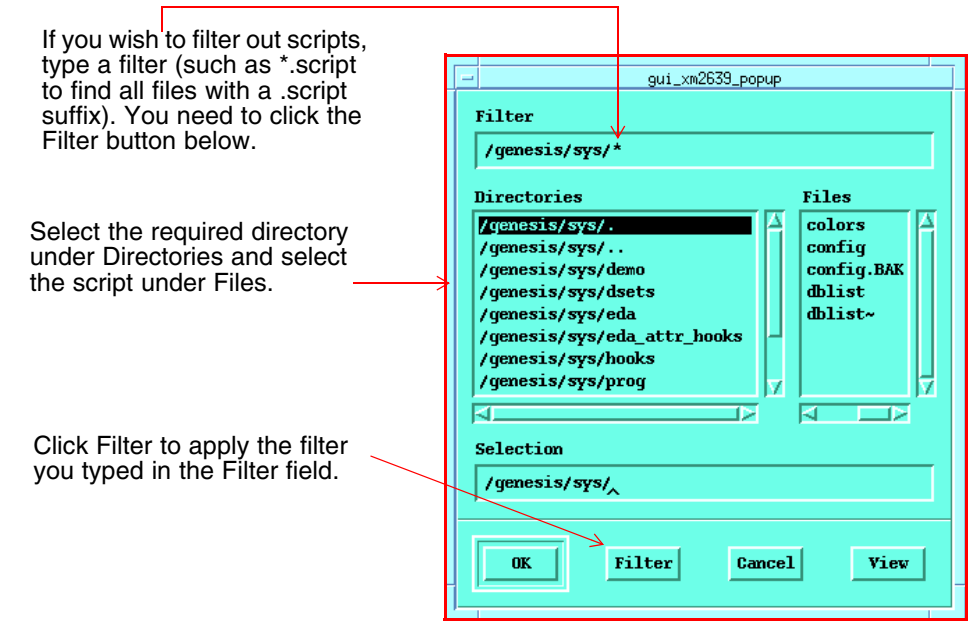
Displays the Script Run popup for selecting and running scripts:



Directory - Displays a popup that allows the selection of a script directory other than Local or Global:



Script - Displays a popup that allows the selection of a script for activation:



Continuing the example from before:

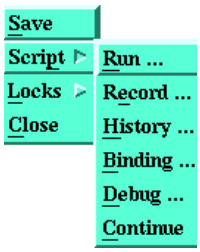
Step 1. **Undo several times, until a message: "Nothing left to undo" appears (Click OK).**

Step 2. **Select local mode and enter the name of the script you saved before.**

Step 3. **Click apply and watch the script executing. Commands are shown in the Script Run window.**

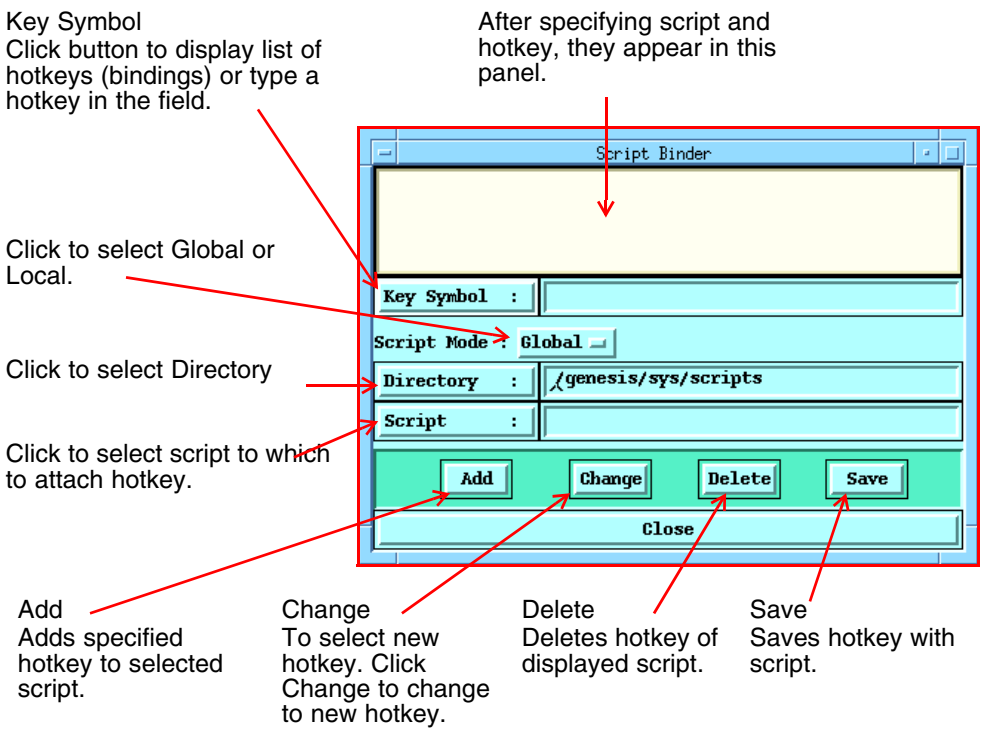
Binding a Script to a Hotkey

In order to attach scripts to function keys, the Script Binder needs to be accessed.



Script > Binding

Displays the Script Binder popup to define hotkeys that activate scripts:



Continuing the example from before:

Step 1. **Select F9 as key symbol**

Step 2. **Select local mode**

Step 3. **Enter the name of the script you have saved before**

Step 4. **Click Add and Save**

Step 5. **Undo several times, as before**

Step 6. **Click F9 and watch the script executing**

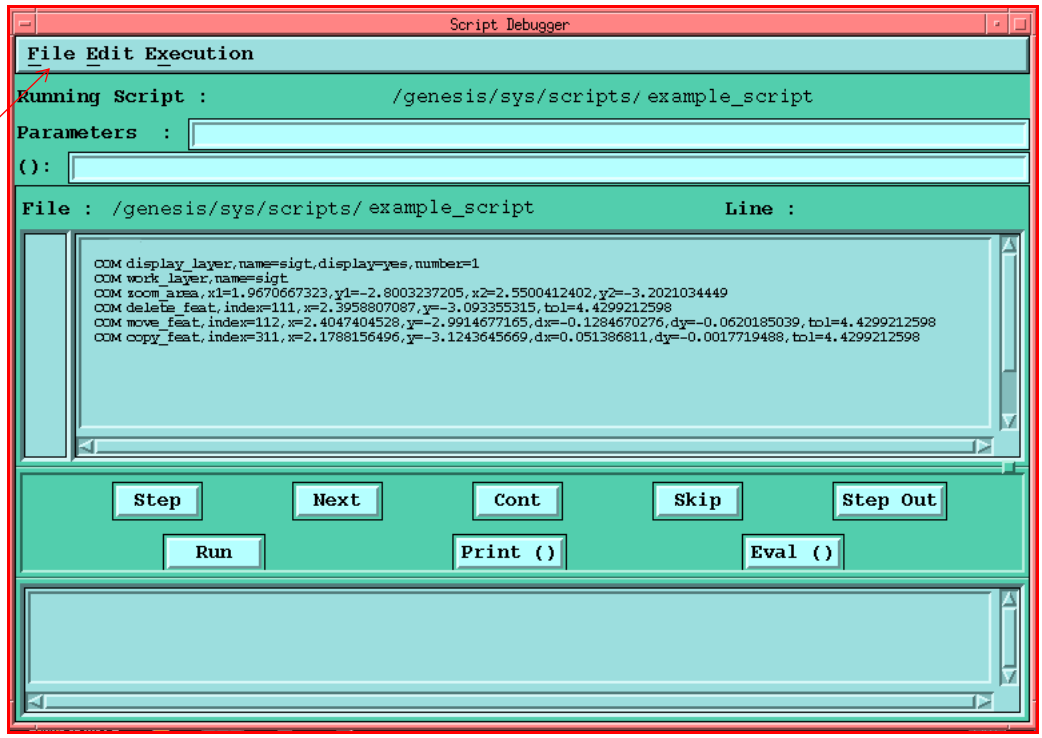
Debugging a Script

Script > Debug

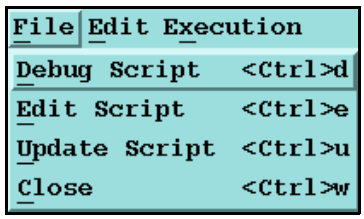
The debugger is accessed through the **File >Script > Debug** submenu, where the following window opens up (shown below with a script selected).

The main menu of the Script Debugger has three selections, shown below, each of which is a dropdown menu.

Main Menu:
File
Edit
Execution

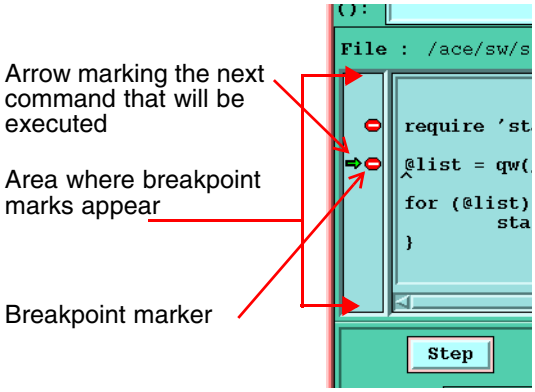


Choosing File opens up the dropdown menu shown below:

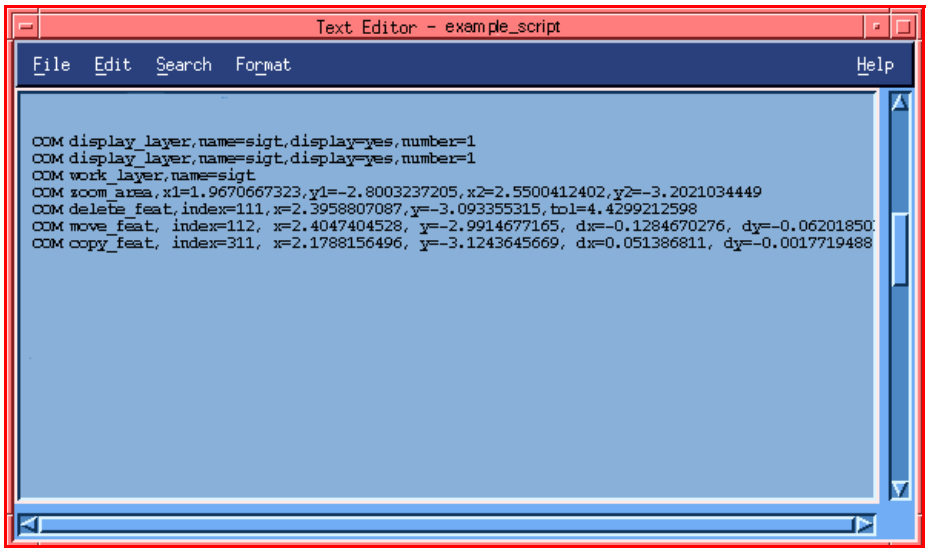


Debug Script - Loads the script, marking on the side window with an arrow where the script is located (which line, as shown in the truncated figure below). Break

points for debugging are also added in this window (by clicking to the side of the line).

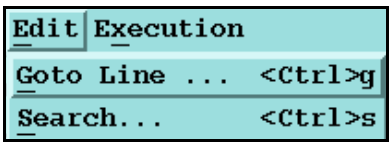


Edit Script - Brings up the user selected text editor with the selected script (the editor can be set with the configuration parameter **`gns_editor_text`**).

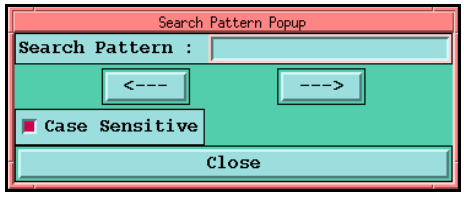


Update Script - Reloads the script after changes are made to disk.

Choosing Edit opens the dropdown menu shown below:

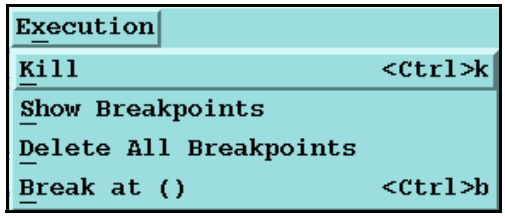


Goto Line - Brings up the popup shown below, where the user adds the line number he wishes to go to.



Search - Brings up the popup shown below, where patterns can be searched for in the script:

Choosing Execution opens up the drop down menu below:



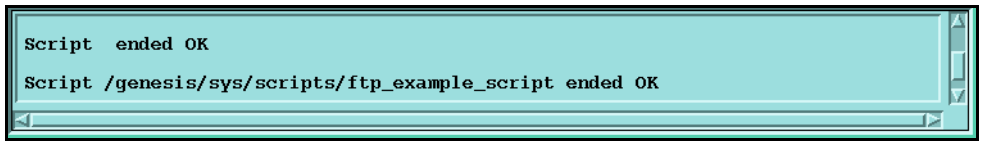
Kill - used to stop a script while in a break point.

Show Breakpoints - Lists all set breakpoints

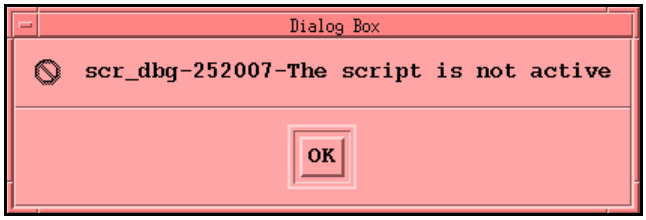
Delete All Breakpoints - Deletes the set breakpoints

Break at () - Sets a breakpoint

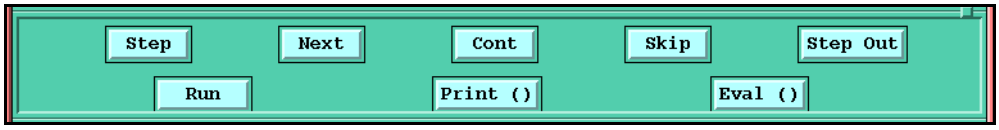
At the bottom of the Debugging window is a frame that shows a progress report of the script being run. Below is an example of a script that was run.



If an attempt is made to debug a script that is not running, a message box appears describing the error, as shown below.



The lower part of the Script Debugger has a series of push-buttons which control the use of the debugger.



Step - To advance one script command.

Next - Advance one script command without entering source scripts.

Cont - Continue after a breakpoint.

Skip - Skips the current command.

Step Out - Exits from a sourced file to the previous level.

Run - Starts script execution.

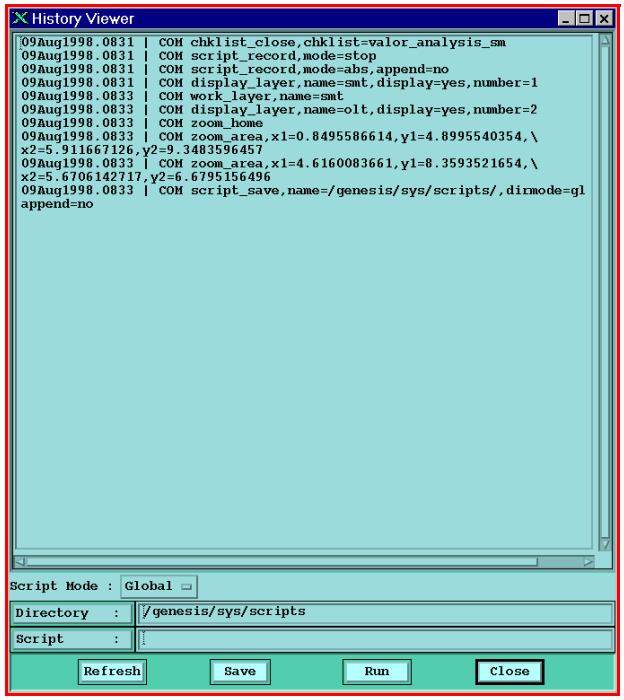
Print () **-** Prints a variable written in the () field above.

Eval() **-** Executes a legal shell command written in the () field above.

File>Script>History

This is a limited function that displays a list of the actions recorded for a certain script. It is enabled when the configuration variable **`scr_history`** is set to Yes. After recording a script you can display a log of the recorded commands with this function.

- Select File>Script>History to display the Script History popup:



Up to version 4.3, the Save and Run buttons in the above popup are inactive.

Chapter 3 *Advanced Script Writing*

The Script Mechanism

The script which was recorded and run in the previous chapter was merely a list of commands, executed sequentially. This script, although performing some tasks automatically, does not represent the full power of the script mechanism in the Genesis 2000 system.

In order to be a powerful mechanism, a script should have the following capabilities:

- Ability to define variables and arrays
- Looping mechanisms
- Conditional Statements
- Ability to perform operating system commands for file access, text manipulation, etc.
- Ability to query the user for input dynamically
- Ability to receive parameters

All these mechanisms, and more, are available with many scripting languages: **`sh, csh, ksh, Perl, Tcl/Tk, python`** and more. However, these languages are general purpose languages and as such do not recognize the specific application line mode commands we have recorded earlier.

In order to mix the benefits of both environments, Frontline has a simple interface through which the script interfaces with the application and 'submits' to it line mode commands for execution. This interface is based on standard channels: stdin (standard input) and stdout (standard output).

The script works normally, using its native syntax and commands. At any time the script needs to interface with the application, it writes a special string to the stdout channel (which can be done by simply echoing the string, although some shortcuts are provided). The application interprets the command and returns one or more response values, which the script can read in the stdin channel. One of these responses is the execution status of the command which can be used by the script to stop the operation.

Skipping Line Hooks

The configuration parameters **gen_line_skip_pre_hooks** and **gen_line_skip_post_hooks** enable skipping line hooks (pre-hooks or post-hooks) when running from the Graphic Editor or from a script. For more information, see Configuration Parameters in Doc. 0203, System Management.

Script Interface Commands

The following commands are supported by the application:

COM
command

Output	**@%#%@COM** <line mode commands with parameters>
Response	1. **STATUS** - Status of the line mode command after execution 2. **COMANS** - A response generated by the command itself
Description	**COM** is used to launch a line mode command for the application. This is the most common use of the interface.
Example	**COM delete_feat,x=2.3,y=-3.1**

AUX
command

Output	**@%#%@AUX** <auxiliary commands with parameters>
Response	1. **STATUS** - Status of the line mode command after execution 2. **COMANS** - A response generated by the command itself
Description	**AUX** is similar to COM in syntax. It is generated by the script mechanism itself when the user switches between graphic editors. This implies a change of context for all active commands.
Example	**AUX set_group,group=1**

PAUSE
command

Output	**@%#%@PAUSE** <pause message>
Response	1. **STATUS** - Status of the operation 2. **READANS** - An empty response generated immediately after the command 3. **PAUSEANS** - An empty response generated after the user continues
Description	PAUSE stops the script execution and opens a popup with a message. The user can continue or abort the script. While the script pauses, the user can use the application in a regular way (even activating other scripts).
Example	**PAUSE Verify the results**

MOUSE
command

Output	**@%#%@MOUSE [p|r]<message>**
Response	1. **STATUS** - Status of the line mode command after execution 2.. **READANS** - An empty response generated immediately after the command 3. MOUSEANS - A string indicating the mouse input 2 or values)
Description	**MOUSE** stops the script execution and opens a popup prompting the user to click M1 in the Graphic Editor area. When the **MOUSE** commands is followed by p, the user should click on a point, and 2 values will be included in MOUSEANS. When the **MOUSE** command is followed by **r**, the user should select a rectangle, and 4 values will be included in **MOUSEANS** (**x1, y1, x2, y2**).
Example	**MOUSE r Define an area**

VOF
command

Output	**@%#%@VOF**
Response	None
Description	**VOF** indicates to the script mechanism that the script writer wishes to handle bad status conditions on commands. Normally, the script mechanism will abort the script whenever a command ends with a non zero status.
Example	**VOF** **COM open_job,job=aaa** **if ($STATUS!= 0) PAUSE Job does not exist**

VON
command

Output	**@%#%@VON**
Response	None
Description	**VON** restores the script mechanism to the default mode of checking the status condition of commands. Users who use VOF should resort to VON as soon as possible to avoid a script 'running wild'.
Example	**VOF** **COM open_job,job=aaa** **if ($STATUS!= 0) PAUSE Job does not exist** **VON**

SU_ON
command

Output	**@%#%@SU_ON**
Response	None
Description	**SU_ON** gives the user running the script temporary super user privileges. This allows him to run scripts prepared by the system manager which use privileged commands. Super user privileges are denied as soon as the script ends as well as during pauses.
Example	**SU_ON**

SU_OFF
command

Output	**@%#%@SU_OFF**
Response	None
Description	**SU_OFF** cancels temporary privileges granted by **SU_ON**.
Example	**SU_OFF**

Script Invocation

The following actions take place when a script is being run (either directly from the Script Run script, by a key binding or from a form callback):

- The following environment variables may be loaded, depending upon the context in which the script is run (in the Graphic Editor only):

  $JOB - Job name in which the script is run

  $STEP - Step name in which the step is run

  $ELEM - Name of the element (in Forms only)

  $VALUE - Value of the element (in Forms only)

  $FORM - Form name (in Forms only)

- The C-shell (/bin/csh) is executed, with 2 parameters:
  - The file **scr_start.csh** in **$GENESIS_DIR/e$GENESIS_VER/all**
  - The user script
- **scr_start.csh** (see Appendix D for listing) performs the following:

  Expand the path environment variable to include

  $GENESIS_DIR/e$GENESIS_VER/all

  Define aliases for the interface commands described above

  Execute a user hook script:

  $GENESIS_DIR/sys/hooks/scripts_start.csh

  This script can be set by the system administrator for common aliases.

Execute the user script itself, either using Perl, Wish, or by using the 'source' command for c-shell scripts.

- The user's script can either be a Perl script, a csh script, or execute another interpreter such as **Tcl/Tk** or **Python**.
- Once the script was invoked, the application waits for input from the script, in one of the formats of the interface commands described earlier. Each time a legal input arrives, the application responds (if needed) and the script reads the response and continues.
- When the script finishes and exits, the application detects this condition and stops waiting for input from the script.

An example of a simple csh script:

```
1  PAUSE Open analysis checklist?
2  VOF
3  COM chklist_open,chklist=analysis
4  set STAT = $STATUS
5  VON
6  if ($STAT!= 0) then
7     COM chklist_from_lib,chklist=analysis
8     COM chklist_open,chklist=analysis
9  endif
10 COM chklist_show, chklist=analysis
```

Note The numbers on the left side are **NOT** a part of the script.

This script does the following:

1. Confirms that the user really wants to open a checklist.

2. Turns off verification since we are not sure if the checklist is already inside the job.

3 - 4.Tries to open the checklist in the job. If the checklist exists, STATUS is set to zero. Otherwise, a non zero status will be returned.

5. Turns verification back on. (This is good practice.)

6 - 9.If the checklist does not exist, copy it from the library and open it. Note that if one of these commands fail, this indicates a real failure (checklist is not in library?) and the script will stop.

10. Shows the checklist, which is now inside the job.

Using Off-line Programs in Scripts

Scripts can include commands which invoke programs. Any operating system command can be executed from a script (i.e., **cp, mv, rm**, etc.). In addition, the user can make use of the following off-line programs developed by Frontline:

gui - This program provides a simple and elegant way to create an input window, containing labels, text fields, radio and set fields, lists, bitmap icons and more.

gui produces output in a form of a shell script, so its output can be redirected to a file which can then be 'sourced' and consequentially set all the correct variables with values entered by the user.

Important:
gui should always run in foreground and not in background. Failing to do so may affect the script synchronization mechanism. Use PAUSE as an alternative if the script has to be paused while the application should be free to run.

For more information regarding the **gui** program, please refer to Doc.0203 , System Management.

dbutil - This program provides various ways to perform database operations on the Genesis distributed database. **dbutil** accesses the database index, making sure concurrency considerations are maintained. It also can provide information, such as which jobs are in which database and the full path to the job.

For more information regarding the **dbutil** program, please refer to Doc.0203, System Management.

INFO - INFO is not a program, it is actually a line mode command which allows extracting data from Genesis entities into a file which can then be 'sourced', and then sets the parameters used by the script. Due to the large usage and scope of INFO, it has a dedicated chapter later in this book.

Note The Genesis installation routine for Windows NT installs a Unix-like environment for running scripts: thus c-shell scripts that run on Unix systems may also run on Windows NT.

Implementing Scripts in Languages Other than csh

The **scr_start.csh** file contains the full implementation of the interface commands (**COM, PAUSE**,...) using **csh** aliases. This mechanism can be easily transformed to any scripting or even high level language such as C or C++. The following chapters describe the implementation provided by Frontline of the Perl scripting language. Other users have implemented Genesis 2000 scripts in **Tcl/Tk** and **Python**. Your local representative may be able to help you in implementing the language of your choice.

Genesis has a new version of C-shell

As of version 7.2, Genesis is provided with a new version of C-shell based on tcsh version 6.09.00. Due to the way this version of C-shell handles file associations, to run tcl scripts on Windows NT (or Windows 2000) you will have to remove the file association for tcl scripts. To do this...

1. Open Windows NT Explorer
2. View -> Options -> File Types
3. Select "Tcl script" and [Remove]

Scripts to Ensure Compatibility with Enterprise

The scripts **change_version.csh** and **change_version.pl** are provided with Genesis v8.2b. These scripts are located in the **$GENESIS_EDIR/all/new_hooks** directory. These scripts change the version number of ODB++ from

version 6.2 to version 6.1 for saved data. This is to ensure that new Enterprise jobs are compatible with ODB++ v6.1

Valor released Enterprise v7.0, which exports ODB++ v6.2 jobs. However, Genesis still exports jobs using ODB++ version v6.1. To ensure that new Enterprise jobs are compatible with ODB++ v6.1, these scripts were created by Valor and added to the 8.2b version of Genesis.

You can run either script from within a shell, or from within a Genesis script. In either case, the affected job must be imported first, and the job must be closed (i.e. it *must not* be open in Genesis) while running the Genesis script.

- When the script is run directly from a shell, use the script **`change_version.csh JOB_NAME`**.

  OR, when you have the PERL interpreter installed, use the script **`perl change_version.pl JOB_NAME`**.
- When the script is run from within Genesis :
  1. In the Engineering Toolkit, select **File > Script > Run** from the menubar. See "Script > Run" on page 10 for details.
  2. Select the location where the script is located from the popup list that appears when you press **Directory**.
  3. Enter **`change_version.csh`** in the **Script** field, or select the script name from the popup list.
  4. Enter the job name in the **Parameters** field.
  5. Click **Apply**.

User Restrictions for Running and Debugging Scripts

Permission to run and debug arbitrary scripts may now be controlled using the **`$GENESIS_DIR/share/privs`** file. This file enables the system administrator to enable/disable certain line mode commands according to the privilege level of the user or according to their application user name.

The following new line mode commands are supported. For details, see Doc. 0206 - Line Mode Commands.

script_debug_show - Shows the C-shell debugger

script_debug_hide - Hides the C-shell debugger

script_bind_show - Shows the script binding window

script_bind_hide - Hides the script binding window

script_run_show - Shows the script run window

script_run_hide - Hides the script run window

Adding one or more of these line mode commands to the **privs** file will enable or disable users from running, debugging, or binding scripts.

To control recording scripts, the line mode command **script_record** already exists.

Example

Add the following commands to the **privs** file to permit running, debugging, or binding scripts only to users with a privilege level of 95 or higher.

```
group scriptcontrol {
global_priv 95
users {
}
commands {
   script_run_show
   script_bind_show
   script_debug_show
}
}
```

Notes regarding Actions/Script Action

- If you do not have permission for the **script_run_show** line mode command, you cannot open the Script Action.
- If "Script Action" appears in a checklist, you can change the parameter of the script path to run any desired script from the checklist action. A "pre" line hook added to the line mode command **chklist_cupd**, which lists what scripts you are allowed to run, could be used to deal with this problem.

Chapter 4 *Perl scripts*

The following steps are needed to enable Perl scripts under Genesis:

Step 1. **Obtain Perl**

Step 2. **Install Perl**

Step 3. **Install the Perl files enabling sockets**

Step 4. **Inspect the files in /genesis/eNN/all/perl**

Step 5. **Install Genesis.pm**

Step 6. **Read rules regarding Perl scripts**

Step 7. **Run the example script**

Step 8. **Debugging**

Step 9. **Remote debugging**

Obtaining Perl

A recent version of Perl must be available on your computer. Version 5.003 is recommended (there are many sites where you can obtain Perl, one example site is ftp://sunsite.auc.dk/pub/languages/perl).

Installing Perl

You need to decompress the Perl tar file, read the INSTALL file, run Configure, and then run make. For this you will need a C compiler.

Install the Perl files enabling sockets

Once Perl has been installed, you must ensure that you run the Perl program h2ph. This creates a number of Perl files which are necessary to enable the use of sockets.

```
cd /usr/include; h2ph * sys/*
```

Inspect the files in $GENESIS_DIR/eNN/all/perl

Have a look in the directory **`/$GENESIS_DIR/eNN/all/perl`**, where NN is the version number. You will find the following files:

```
Genesis.pm
server.pl
example.pl
```

Genesis.pm is the interface from Perl to Genesis and must be used in all scripts.

server.pl is the script to run from Genesis to allow perl scripts to be debugged.

example.pl is an example of a Perl script.

All these files are scripts and are readable.

Install Genesis.pm

Genesis.pm is a Perl library, and as such, Perl must know where to find it. The simplest option is to install the file in the site_perl directory which is under the lib/perl/site_perl directory of where Perl has been installed.

Alternatively, you can leave it where it is and define the global variable PERLLIB to point to the correct directory or you can define the library area by placing the line

```
use lib qw($GENESIS_DIR/e30/all/perl);
```

before "use Genesis" in your script.

See the Perl manual for a fuller description of these options.

Read rules regarding Perl scripts

All scripts that interface to Genesis 2000 must contain the lines

```
!# /usr/local/bin/perl
use Genesis;
```

at the beginning.

!# /usr/local/bin/perl - must be the correct path for your installation

use Genesis;- defines the interface to Genesis

$f = new Genesis;- starts up the connection to Genesis

Then you can use **COM, VOF** etc. just like in csh with some minor syntactic differences.

To use **VOF**, just enter **$f->VOF**;

To use **COM**, enter **$f->COM("line_mode_command", param1 => "value1", param2 => "value2", ...);**

Note If any of the names of parameters are reserved Perl words, they must also be quoted.

The interface is very similar to the csh interface, and an example of its use can be found in the file example.pl.

It is essential that any Perl script contains the word "perl" in the first line, otherwise Genesis will think that it is a csh and will fail to properly carry out the command.

Once the script is ready, you just run the script like any other script.

Run the example script

Run the script example.pl exactly as you would a csh script and see what results it produces.

Debugging

Perl has it's own native, very powerful debugger which you can use. If you have a script you want to debug, you first have to run the script server.pl from Genesis.

Once this script has been run, Genesis waits for commands from your Perl script. To start your own Perl script you must open a shell such as xterm, hpterm or EMACS, and run the command

```
perl -d example.pl
```

This puts you into Perl debug mode, and you can single step from there.

You don't have to use the -d option. Without it, the script simply runs to completion.

Remote debugging

Because communication between the Perl script and Genesis uses sockets, it is just as easy to run the Perl script from a remote computer.

On the remote computer you type

```
perl -d example.pl hostName
```

where hostName is the name of the computer running Genesis, and you will be able to debug the application remotely.

Additional Documentation

The Genesis.pm is self-documented using **pod**.

BUGS

If you receive the following output then the file **scr_start.csh** is not up to date and does not work with Perl scripts:

```
=head: Command not found
Set: Command not found
Most: Command not found
=cut: Command not found
use: Command not found
use: Command not found
sub: Command not found
sub: Command not found
ARGV: Undefined variable.
```

Package for Perl scripts in an Genesis 2000 environment.

Usage:

```
use Genesis;
use Genesis('122.12.1.87');
$f = new Genesis;
$f->COM($command);
```

The idea behind this module is that you can run a Perl program from within Genesis, or debug it from an xterm, or even remotely. When running the Perl script from inside Genesis, simply choose the relevant script from the "Script Run" screen. When running or debugging a script from an xterm, you still have to go the "Script Run" screen, but this time you have to choose the script called server.pl. This script sets up a socket which waits for commands from the Perl script to be debugged. Having started the script server.pl, open up an xterm and start debugging the script.

The conventions in the Perl script are slightly different from the csh equivalent.

The script is written using object oriented techniques.

The start of the script must begin with **`use Genesis`**;

If the Genesis module Genesis.pm is not in the normal Perl library, you can add it the path to the environment variable **`PERL5LIB`**, or add the line use: **`lib qw(/pathname)`** where **`/pathname`** must be the directory where the file **`Genesis.pm`** resides. The line '**`use lib`**' must appear before the line '**`use Genesis`**'.

The next line should be

```
$f = new Genesis;
```

`$f` is simply a variable that you can choose. This starts up the socket if necessary and also imports the correct environment variables from Genesis 2000, to ensure that all variables are accessible.

The public functions are **`VON, VOF, SU_ON, SU_OFF, PAUSE, MOUSE, COM, AUX`** and **`DO_INFO`**.

They are invoked in the object oriented way. Here is an example of the **`PAUSE`** command

```
$f->PAUSE($text);
```

Variables are created when using **`DO_INFO`**.

Unlike the **`csh`**, the variables are put into the structure pointed to by **`$f`**. So suppose a variable came into existence, it would be referenced as:

`$f->{gEXISTS}`.

If an array were to be received, the elements could be referenced as:

`$f->{gWIDTHS}[$i]`.

Similarly, the return results are called **`STATUS, READANS, PAUSANS, MOUSEANS and COMANS`**. These can be read straight from the structure as **`$f->{STATUS}`** etc.

Debugging of Genesis 2000 Scripts

Genesis 2000 comes with a Perl script that has the ability to debug programs (on any network connected station) used by Genesis 2000.

Chapter 5 *Tcl/Tk Scripts*

Obtaining Tcl/Tk

Tcl/Tk can be downloaded from many sites. The current version, 8.1, at the time of writing, can be downloaded from:

http://sunscript.sun.com/TclTkCore/8.1.html

Installing Tcl/Tk

Once you downloaded the **Tcl/Tk gzip** files, gunzip them and follow the instructions in the README file:

Tcl/Tk Scripts Rules

All Tcl/Tk scripts must contain the following line at the beginning, pointing to your "wish" Tcl/Tk program, e.g.

```
#!/usr/bin/wish
```

Then you can use all regular **COM, MOUSE, VON, VOF** commands.

When a script command needs parameters to be passed, enclose them in quotation marks, as in the following example :

```
COM "info,out_file=/tmp/test,args= -t eda -e tstjob/pcb -d COMP
-o bottom"
```

Tcl/Tk Sample Script

This is a small script in **Tcl/Tk** that can be executed from a job.

It is intended to output to the file **/tmp/xxx** all the steps defined in the current job, and then wait for user confirmation.

```
.................
#!/sw/bin/wish
VOF
COM "info,out_file=/tmp/xxx,args= -t job -e $env(JOB) -d
STEPS_LIST"
puts "status is $STATUS"
if {$STATUS != 0} {
  puts "Something is wrong here..."
}
MOUSE "waiting for confirmation..."
...................
```

Running Tcl scripts on Windows NT or Windows 2000

Genesis v7.2 is provided with a new version of C-shell based on tcsh version 6.09.00. Due to the way this version of C-shell handles file associations, to run tcl scripts on Windows NT (or Windows 2000) you will have to remove the file association for tcl scripts. To do this...

1. Open Windows NT Explorer
2. View -> Options -> File Types
3. Select "Tcl script" and [Remove]

Chapter 6 *The Info Command*

The line mode command '**info**' is a powerful tool used to retrieve information from database entities to a file. Info works on entities in memory, thus its output represents the entity current data, even if the entity was not yet saved to disk. This is the main reason why it is implemented as a line mode command and not as a separate program, with no access to memory space.

Usage

```
COM info, [out_file=<filename>,]                    \
          [write_mode=<replace/append>,]           \
           [units=<inch/mm>,]                        \
          args = -t entity_type                    \
                  [-e entity_path (mandatory]       \
                  [-m output_method]                \
                  [-d data_type]                    \
                  [-p parameters]                   \
                  [-s serial_number]                \
                  [-o options]                      \
                   [-help]
```

Info receives 4 parameters:

out_file - name of the file to create with the information requested by the **args** parameters (see below).

write_mode - <*replace*> overwrite existing **out_file** or <*append*> add to end of existing **out_file**

units - <*inch*> or <*mm*>

args - compound parameter with arguments, as described below.

The '**args**' parameter must contain the following mandatory fields:

-d data_type - defines the specific data type to be retrieved (the default is to show all the entity data types). A special type, '**exists**', is common to all entities and is used to check if the specified entity exists. Data_type can be ERF (see "Data Types" on page 32). For severity level filtering with the **-d MEAS** parameter see "Severity Level Extraction" on page 47.

-t entity_type - one of the entity types described in "Entity Types" on page 34.

-e entity_path - The combination of the job name and the specified entity name (e.g job/step/layer or job/matrix) or the name of a standard symbol in order to extract its limits (bounding box). For example:

```
info, out_file=..., args= -t symbol -e <rect100x200>
```

will generate the limits of the bounding box of this standard symbol.

Note You cannot use the <job-name>/<entity-name> notation to get info on standards. Only limits can be retrieved for standard symbols.

`-m output_method -` where output_method =
<*display*> - requested data recorded as text
or
<*script*> outputs the data with C-shell - '**`set`**' commands, allowing immediate usage in scripts using the 'source' command.

`-p parameters -` set of parameters that are part of the specified data type. This is a single string that contains the parameter names, separated by '+' characters. If the parameters are not specified, all of them will be displayed.

`-s serial_number -` used for array data types. Specifies a single array entry to be displayed. If not specified the whole array is displayed.

`-o options -` string of special options that are separated by '+'characters (for severity level option see "Severity Level Extraction" on page 47).

`-h help -` to get this help

Example

Info, in the following command, will return in **`gNETS_LIST`** a list of the net names for active net lists that exist for the given step (specified in **`$step_name`**) for the job (specified in **`$job_name`**).

```
.... -t step -e $job_name/$step_name -d NETS_LIST ...
```

Possible net names are:

cadnet curnet refnet

Note A utility for generating all the Genesis INFO commands is located at **`http://www.frontline-pcb.com/support/genesis/articles/scripting/infowiz/infowiz.html`**
This utility may be downloaded to your computer for easy access.

Data Types

-d data_type - data type can an ERF: **`-d ERF`**, as in the following command (all categories will be listed):

```
COM info,args=-t  check -e $JOB/pcb/$CHK_NAME -d ERF \
-o action=2,out_file=/tmp/mk1,write_mode=replace
```

where the response can be, for example:

```
p2p 4 6 8
```

Example where one category is listed:

```
COM info,args=-t  check -e $JOB/pcb/$CHK_NAME -d ERF \
-o action=2+category=ar,out_file=/tmp/mk1,write_mode=replace
```

where the response can be, for example:

```
ar 6 8 10
```

The out_file Format

The table below describes how the info command for **DATA_TYPE** is expressed in the **out_file** when the **-m output_method** is *display* **or** *cshell*.

Simple DATA_TYPE	
-m output_method	**Expression in Out_File**
<*display*>	**DATA_TYPE =** *info_val*
<*cshell*>	**set gDATA_TYPE =** *'info_val'*

Array DATA_TYPE(array)	
-m output_method	**Expression in Out_File**
<*display*>	**DATA_TYPE[***1***] =** *info_val1* **DATA_TYPE[***2***] =** *info_val2* ... **DATA_TYPE[***n***] =** *info_valn*
<*cshell*>	**set gDATA_TYPE = (***'info_val1' 'info_val2'...'info_valn'***)**

Simple DATA_TYPE with parameters: par1, par2	
-m output_method	**Expressed in Out_File**
<*display*>	**DATA_TYPE:***par1 = val_par1,par2=val_par2*
<*cshell*>	**set gDATA_TYPE***par1* **=** *'val_par1'* **set gDATA_TYPE***par2* **=** *'val_par2'*

DATA_TYPE(array) with parameters: par1, par2	
-m output_method	**Expressed in Out_File**
<*display*>	**DATA_TYPE[1]:** *par1***=** *val_par1***,** **DATA_TYPE[2]:** *par2***=** *val_par2*
<*cshell*>	**set gDATA_TYPE***par1* **= (***'val1' 'val2' 'valn'***)** **set gDATA_TYPE***par2* **= (***'val1' 'val2' 'valn'***)**

Entity Types

-t entity_type - One of the following types:

attributes	layer	netlist	step
camtek_aoiset	mania_aoiset	notes	symbol
check	matrix	panel_classes	wheel
etset	ncrset	root	
job	ncset	stackup	

Note The **`y2k_info_4`** configuration parameter determines the use of 4-digit year format with the **`-t entity_type`** option (where **`entity_type`** can be **`check`** or **`notes`**)

The following table describes all supported entities, their data types, and parameters.

:

Entity	Data Type	Parameters
`attributes`	EXISTS FORCE_LIB NUM_ATR ATR (array)	 - (Options: deleted) - name, changed, context, type, entity, def, min_txt_len, max_txt_len, options, options_del, min_int_val, max_int_val, min_flt_val, max_flt_val (Options: deleted)
`camtek_ aoiset`	EXISTS NUM_X_FRAMES NUM_Y_FRAMES SCAN_AREA	 - xmax, xmin, ymax, ymin
`check`	CHK_ATTR (array) DURATION ERF EXISTS LAST_TIME MEAS (free text output) MEAS_DISP_ID (free text output) NUM_ACT REPORT (free text output) STATUS TITLE	- name, val, exists

Entity	Data Type	Parameters
etset	ADAPTER_LIM	- xmin, ymin, xmax, ymax
	ADAPTER_NAME	
	EXISTS	
	GRID_LIM	- xmin, ymin, xmax, ymax
	SPLITS_HIST	
	TESTER_TYPE	
job	ATTR (array)	- name, val
	CHANGES	
	EXISTS	
	FLOWS_LIST (array)	
	FORMS_LIST (array)	
	IS_CHANGED	
	MATRIX_LIST (array)	
	STACKUPS_LIST (array)	
	STEPS_LIST (array	
	SYMBOLS_LIST (array)	
	TEMPLATES_LIST (array)	
	WHEELS_LIST (array)	

Entity	Data Type	Parameters
eda	NCOMPS	- (options: top, bottom)
	TH	- (options: select, top, bottom)
	COMP (array)	- (options: select, top, bottom) refdes, centroidx, centroidy, npins, pin0x, pin0y, side, rotation, mirror, color, serial, limitminx, limitmaxx, limitminy, limitmaxy, cpn, ipn mpn, vendor, bompackage*, package**..
comp	PART	
	REFDES	
	X	
	Y	
	NPINS	
	SIDE	
	ROTATION	
	MIRROR	
	COLOR	
	SERIAL	
	PIN0X	
	PIN0Y	
	LIMITMINX	
	LIMITMAXX	
	LIMITMINY	
	LIMITMAXY	
	PIN (array)	- num, x, y
	ATTRIBUTE (array)	- name, val
	PROPERTIES (array)	- name, val
	CPN	
	IPN	
	MPN	
	VENDOR	
	BOMPACKAGE*	
	PACKAGE**	

* BOMPACKAGE is the package name derived from the BOM (Bill of Materials) module.

** PACKAGE is the package name received from CAD.

Entity	Data Type	Parameters
layer	CONTEXT	
	TYPE	
	POLARITY	
	SIDE	
	DRL_START	
	DRL_END	
	FOIL_SIDE	
	SHEET_SIDE	
	ROW	
	LIMITS	- xmin, ymin, xmax, ymax
	SYMS_HIST (array)	- symbol, line, pad, arc (options: break_sr)
	FEAT_HIST	- line, pad, surf, arc, text, total (options: select, break_sr)
	SLOT_HIST (1)	
	NUM_TOOL	
	TOOL (array)	- num, count, type, type2, min_tol, max_tol, finish_size, drill_size, bit
	TOOL_THICK	
	TOOL_USER	
	ATTR (array)	- name, val
	FEATURES (free text output)	- (Options: break_sr, break_feat, select)
	NCSETS_LIST (array)	
	AOISETS_LIST (array)	
	NCRSETS_LIST (array)	
	LPD	- was_input, is_defined, polarity, speed, xstretch, ystretch, xshift, yshift, xmirror, ymirror, copper_area, xcenter, ycenter, xcenter_inch, ycenter_inch, plot_kind1, plot_kind2, minvec, advec, minflash, adflash, conductors1, conductors2, conductors3, conductors4, conductors5, media, resolution, smoothing, swap_axes See more information on LPD at “Layer Data Type LPD” on page 40.
	LPM	- was_input, is_defined, polarity, speed, xstretch, ystretch, xshift, yshift, xmirror, ymirror, copper_area, xcenter, ycenter, xcenter_inch, ycenter_inch, plot_kind1, plot_kind2, minvec, advec, minflash, adflash, conductors1, conductors2, conductors3, conductors4, conductors5, media, resolution, smoothing, swap_axes See more information on LPM at “Layer Data Type LPM (LPD Multiple)” on page 40.

- (1) Describes the Slot Histogram. The Slot Histogram describes the size, length, and angle of each slot, and the number of slots in a drill layer.

Entity	Data Type	Parameters
	WIDTH	
	HEIGHT	
	TARGET_THICK	
	TARGET_POS_TOL	
	TARGET_NEG_TOL	
	VENDOR	
	COST	
	THICK	
	IS_MIRROR	
	CONSTRUCT	
	PLATE_THICK	
	MASK_THICK	
	THICK_TYPE	
stackup	PILE (array)	cost, material, weave, thick, pos_tol, neg_tol, name, cat_num, vendor, upsidedown
	PRESSED (array)	material, thick, known
	LAYER (array)	side, serial, name
	VALID	
	IMP (array)	model, ref1, ref2, lyr, dual_lyr, imp, imp_pos_tol, imp_neg_tol, target, target_tol, orig_width, curr_width, orig_space, curr_space, width_plus, width_minus
	ALLOW_CHANGE	
	WIDTH_VAR	
	ETCH_FACTOR	
	MASK_ER	
	FREQ	
	RESIN_ER	
wheel	DCODE (array)	num, symbol, angle, mirror
	NUM_ROWS	
	NUM_COLS	
	NUM_LAYERS	
	NUM_STEPS	
matrix	ROW (array)	row, type, name, context, layer_type, polarity, side, drl_start, drl_end, foil_side, sheet_side
	COL (array)	col, type, step_name
	ATTR (array)	val
	CHK_ATTR (array)	name, val, exists
	LAST_TIME	
	DURATION	
	NUM_ACT	
check	TITLE	
	STATUS	
	ERF	
	MEAS (free text output)	
	REPORT (free text output)	
	MEAS_DISP_ID (free text output)	

Entity	Data Type	Parameters
attributes	FORCE_LIB NUM_ATR ATR (array)	(Options: deleted) name, changed, context, type, entity, def, min_txt_len, max_txt_len, options, options_del, min_int_val, max_int_val, min_flt_val, max_flt_val, (Options: deleted)
panel_ classes	NUM_CLS CLS (array)	name, width, height, t_active, b_active, l_active, r_active, sr_margin, sr_orient
ncset	MACHINE FORMAT REG NUM_SPLITS NUM_STAGES	
ncrset	MACHINE FORMAT REG	
netlist		Example **COM info,out_file=$GENESIS_TMP/ info.out.$$, ** **args= -t netlist -e $JOB/$STEP/ <netlistname>** **-d <NUM_NETS, END_PT_TOP, END_PT_BOT, END_PT_TH, NET_PT_TOP, NET_PT_BOT, NET_PT_TH>** **<netlistname>** can be one of the following: **refnet**, **curnet**, **cadnet**, or **curcadnet**. For end point information, use **refnet**. **Available Netlist Options** NUM_NETS = Total number of nets END_PT_TOP = Number of end points on the board's top side END_PT_BOT = Number of end points on the board's bottom side END_PT_TH = End point through holes NET_PT_TOP = Number of net points on the board's top side NET_PT_BOT = Number of net points on the board's bottom side NET_PT_TH = Net point through holes
notes	NOTE (array)	date, time, user, x, y, text

Layer Data Type LPD

Setting the **Info** *command to return specific LPD parameters correctly*

Script behavior in Genesis prior to version 8.1 - When you run the INFO command to get the layer production data (LPD) from a specific layer in which the stretch center X and Y are not zero, then the value of the stretch center in reported in mils instead of inches. The stretch center data is input in inches but is read and reported as mils. All other layer production data parameters are input, read, and reported in inches.

Script behavior in Genesis as of version 8.1 - To ensure that the value of the stretch center (X and Y) is reported in inches, use the parameters **`xcenter_inch`** and **`ycenter_inch`** as part of the script arguments. See the example below.

Example

```
COM info,out_file=/tmp/aaa, \
args= -t layer -e $JOB/$STEP/gerber_2 -d LPD -p
xstretch+ystretch+xcenter+ycenter \
+xcenter_inch+ycenter_inch
source /tmp/aaa
```

Layer Data Type LPM (LPD Multiple)

Data type **`LPM (LPD Multiple`**) was added to the layer entity information. It is an array data type. It generates variables similar to the layer data type **`LPD`**, but the variable name includes **`LPM`** instead of **`LPD`**.

For example: LPD generates **`set gLPDpolarity = 'positive'`**
while LPM generates

```
set gLPMpolarity = ('positive' 'positive' 'positive' 'positive'
'positive')
```

The **`'-s serial_number'`** directive can be used to extract information that describes the specific device type
`(1-EITHER_TYPE, 2-LP7008, 3-XPRESS, 4-LP5008, 5-DP100)`.

Examples

The command

```
COM info, out_file=/tmp/info_out, write_mode=replace, args=-
t layer -e JOB/STEP/LAYER -d LPM -s 2
```

will retrieve Layer Production Data information from the LP7008 device type.

The output from the above INFO command looks like this:

```
set gLPMwas_input    = ('no'          )
set gLPMis_defined   = ('yes'         )
set gLPMpolarity     = ('positive'    )
.
.
set gLPMresolution   = ('eighth_mil' )
set gLPMsmoothing    = ('smooth'      )
set gLPMswap_axes    = ('no_swap'     )
.
```

```
.
set gLPMenl_by10      = ('0'          )
set gLPMenl_ctr_by    = ('0'          )
set gLPMdevice_type   = ('lp7008'     )
```

The command

```
COM info, out_file=/tmp/info_out, write_mode=replace, args=-
t layer -e JOB/STEP/LAYER -d LPM
```

will retrieve the Layer Production Data information from all device types.

The output from the above INFO command looks like this:

```
set gLPMwas_input = ('no' 'no' 'no' 'no' 'no')
set gLPMis_defined = ('yes' 'yes' 'yes' 'yes' 'yes')
set gLPMpolarity = ('positive' 'positive' 'positive' 'positive'
'positive')
set gLPMspeed = ('0' '0' '0' '0' '0')
set gLPMxstretch = ('100.0000' '100.0000' '100.0000' '100.0000'
'100.0000')
set gLPMystretch = ('100.0000' '100.0000' '100.0000' '100.0000'
'100.0000')
set gLPMxshift = ('0' '0' '0' '0' '0')
set gLPMyshift = ('0' '0' '0' '0' '0')
set gLPMxmirror = ('0' '0' '0' '0' '0')
set gLPMymirror = ('0' '0' '0' '0' '0' )
set gLPMcopper_area = ('0.0000' '0.0000' '0.0000' '0.0000'
'0.0000')
set gLPMxcenter = ('0' '0' '0' '0' '0')
set gLPMycenter = ('0' '0' '0' '0' '0')
set gLPMxcenter_inch = ('0' '0' '0' '0' '0')
set gLPMycenter_inch = ('0' '0' '0' '0' '0')
set gLPMplot_kind1 = ('56' 56' '56' '56' '56')
set gLPMplot_kind2 = ('56' 56' '56' '56' '56')
set gLPMminvec = ('0' '0' '0' '0' '0')
set gLPMadvec = ('0' '0' '0' '0' '0')
set gLPMminflash = ('0' '0' '0' '0' '0')
set gLPMadflash = ('0' '0' '0' '0' '0')
set gLPMconductors1 = ('0' '0' '0' '0' '0')
set gLPMconductors2 = ('0' '0' '0' '0' '0')
set gLPMconductors3 = ('0' '0' '0' '0' '0')
set gLPMconductors4  = ('0' '0' '0' '0' '0')
set gLPMconductors5  = ('0' '0' '0' '0' '0')
set gLPMmedia = ('first' 'first' 'first' 'first' 'first')
set gLPMresolution = ('quarter_mil' 'eighth_mil'  'quarter_mil'
'quarter_mil' 'quarter_mil')
set gLPMsmoothing = ('smooth' 'smooth' 'smooth' 'smooth'
'smooth')
set gLPMswap_axes = ('no_swap' 'no_swap' 'no_swap' 'no_swap'
'no_swap')
set gLPMdef_ext_lpd  = ('yes' 'yes' 'yes' 'yes' 'yes')
set gLPMres_value = ('0.250000' '0.062500' '0.250000'
'0.250000'    '0.250000')
set gLPMres_units = ('mil' 'mil' 'mil' 'mil' 'mil')
```

```
set gLPMenl_pol = ('both' 'both' 'both' 'both' 'both')
set gLPMenl_other = ('leave_as_is' 'leave_as_is' 'leave_as_is'
'leave_as_is' 'leave_as_is')
set gLPMenl_panel = ('no' 'no' 'no' 'no' 'no')
set gLPMoverlap = ('no' 'no' 'no' 'no' 'no')
set gLPMenl_img_sym  = ('no' 'no' 'no' 'no' 'no')
set gLPMenl_0_vecs = ('no' 'no' 'no' 'no' 'no')
set gLPMenl_sym = ('none' 'none' 'none' 'none' 'none')
set gLPMenl_sym_by = ('0' '0' '0' '0' '0')
set gLPMsym_name1 = ('' '' '' '' '')
set gLPMenl_by1 = ('0' '0' '0' '0' '0' )
set gLPMsym_name2 = ('' '' '' '' '')
set gLPMenl_by2 = ('0' '0' '0' '0' '0')
set gLPMsym_name3 = ('' '' '' '' '')
set gLPMenl_by3 = ('0' '0' '0' '0' '0')
set gLPMsym_name4 = ('' '' '' '' '')
set gLPMenl_by4 = ('0' '0' '0' '0' '0')
set gLPMsym_name5 = ('' '' '' '' '')
set gLPMenl_by5 = ('0' '0' '0' '0' '0')
set gLPMsym_name6 = ('' '' '' '' '')
set gLPMenl_by6 = ('0' '0' '0' '0' '0')
set gLPMsym_name7 = ('' '' '' '' ' )
set gLPMenl_by7 = ('0' '0' '0' '0' '0')
set gLPMsym_name8 = ('' '' '' '' ' )
set gLPMenl_by8 = ('0' '0' '0' '0' '0')
set gLPMsym_name9 = ('' '' '' '' '')
set gLPMenl_by9 = ('0' '0' '0' '0' '0')
set gLPMsym_name10 = ('' '' '' '' '')
set gLPMenl_by10 = ('0' '0' '0' '0' '0')
set gLPMenl_ctr_by = ('0' '0' '0' '0' '0')
set gLPMdevice_type = ('either_type' 'lp7008' 'xpress' 'lp5008'
'dp100' )
```

The Basic Script

Let us create a simple script using the INFO command, and gradually enhance it with the options provided.

Open a new script with the following commands:

```
COM info, out_file=/tmp/info,write_mode=replace, \
          args= -t job -e $JOB

cat /tmp/info
```

Remember that no characters should follow the '\' character at the end of the first line.

- Open the Script Run popup, **be sure to open it from within an open job**.
- Click apply in the Script Run popup and watch the transcript.

About 10 '**set**' commands are listed, each specifying a data element from within the entity. The listing is in 'Script mode' meaning that a C-shell script can 'source' this file and utilize the resulting variables.

```
set gSTEPS_LIST = ('pcb')
set gSYMBOLS_LIST = ()
set gWHEELS_LIST = ('aperture_table_report' 'fablink_tmpfile'
                    'wheel')
set gMATRIX_LIST = ('matrix')
set gSTACKUPS_LIST = ()
set gTEMPLATES_LIST = ()
set gATTRname =     ('.customer' '.comment' 'connector'
                    'target' 'component' 'comment' 'hole_type'
                    'serial_number')
set gATTRval = ('' '' 'no' 'no' '' '' 'plated' '15' )
set gFORMS_LIST = ()
set gFLOWS_LIST = ('f1' 'f1+1' 'fixturing_flow' 'master')
```

Let us see the alternative mode (Display):

Display vs. Script Mode

Change the script to the following:

```
COM info, out_file=/tmp/info,write_mode=replace, \
args= -t job -e $JOB -m display
cat /tmp/info
```

This format opens each array of values into multiple lines.

```
STEPS_LIST[1]=pcb
WHEELS_LIST[1]=aperture_table_report
WHEELS_LIST[2]=fablink_tmpfile
WHEELS_LIST[3]=wheel
MATRIX_LIST[1]=matrix
ATTR[1]:name=.customer,val=""
ATTR[2]:name=.comment,val=""
ATTR[3]:name=connector,val=no
ATTR[4]:name=target,val=no
```

```
ATTR[5]:name=component,val=""
ATTR[6]:name=comment,val=""
ATTR[7]:name=hole_type,val=plated
ATTR[8]:name=serial_number,val=15
FLOWS_LIST[1]=f1
FLOWS_LIST[2]=f1+1
FLOWS_LIST[3]=fixturing_flow
FLOWS_LIST[4]=master
```

Retrieving Only One Data Type

The number of data types in an entity can be quite large. To improve performance, you can retrieve only one data type at a time.

Let us change the script as follows:

```
COM info, out_file=/tmp/info,write_mode=replace, \
args= -t job -e $JOB -d STEPS_LIST
cat /tmp/info
```

The resulting output is now quite terse:

```
set gSTEPS_LIST = ('pcb')
```

Retrieving Numeric Values

Numeric values can be retrieved in the same manner as textual values. This example retrieves the profile limits from a step. We will use the **-t**, **-e** and **-d** parameters for this purpose.

Note You must activate this script from within an open Graphic Editor so the current step will be defined.

```
COM info, out_file=/tmp/info,write_mode=replace, \
args= -t step -e $JOB/$STEP -d PROF_LIMITS
cat /tmp/info
```

The output of this data element has 4 values:

```
set gPROF_LIMITSxmin = '-0.4'
set gPROF_LIMITSymin = '-4'
set gPROF_LIMITSxmax = '4.1'
set gPROF_LIMITSymax = '-1'
```

Since this output is unit sensitive, we can add the units parameter of the info command to receive it in millimeters:

```
COM info, out_file=/tmp/info ,write_mode=replace, units=mm, \
args= -t step -e $JOB/$STEP -d PROF_LIMITS
cat /tmp/info
```

The result will be displayed in millimeters:

```
set gPROF_LIMITSxmin = '-10.16001'
set gPROF_LIMITSymin = '-101.60001'
set gPROF_LIMITSxmax = '104.14001'
set gPROF_LIMITSymax = '-25.39999'
```

Using parameters, the user can limit the output to a subset of the values. Parameters contain a + separated list of sub_fields of the data types.

```
COM info, out_file=/tmp/info ,write_mode=replace, units=mm, \
args= -t step -e $JOB/$STEP -d PROF_LIMITS -p+xmin+ymin
cat /tmp/info
```

The result:

```
set gPROF_LIMITSxmin = '-59.69001'
set gPROF_LIMITSymin = '-6.35001'
```

Checking For Existence

A special data type '**-d exists**' can be used to verify the existence of an entity.

Let us check for the existence of a layer in the step:

```
COM info, out_file=/tmp/info ,write_mode=replace, units=mm, \
args= -t layer -e $JOB/$STEP/not_likely -d exists
cat /tmp/info
```

The result is a variable set to either '**yes**' or '**no**'.

```
set gEXISTS = no
```

More About Parameters

The output of some data types can be fine tuned with options and parameters. Assuming you have a layer called '**drill**' in the step, enter the following script:

```
COM info, out_file=/tmp/info ,write_mode=replace, \
args= -t layer -e $JOB/$STEP/drill
cat /tmp/info
```

The output shows how many lines, pads and arcs you have for each symbol:

```
set gSYMS_HISTsymbol = ('r33' 'r30' 'r93')
set gSYMS_HISTline = ('0' '0' '0' )
set gSYMS_HISTpad = ('209' '9' '3' )
set gSYMS_HISTarc = ('0' '0' '0' )
```

If you wish to receive only the **line** and **arc** list (e.g., to verify there are none in the drill layer), do the following:

```
COM info, out_file=/tmp/info ,write_mode=replace, \
args= -t layer -e $JOB/$STEP/drill -d syms_hist -p +line+arc
cat /tmp/info
```

Output

```
set gSYMS_HISTline = ('0' '0' '0')
set gSYMS_HISTarc = ('0' '0' '0')
```

Both **-p** (parameters) and **-o** (options) are represented as a list of '**+**' separated options.

Reading Checklists

Information of a checklist is always retrieved according to a specific action. Therefore, the option: **action=n** should be included, where **n** is the serial number of an action within a checklist. If you have a checklist called **my_check** inside the job, try the following script:

```
COM info,args=-t check -e $JOB/$STEP/my_check -d LAST_TIME \
-o action=1, out_file=/tmp/info, write_mode=replace
cat /tmp/info
```

Output

```
set gLAST_TIME = '28 Apr 97 06:55 PM'
```

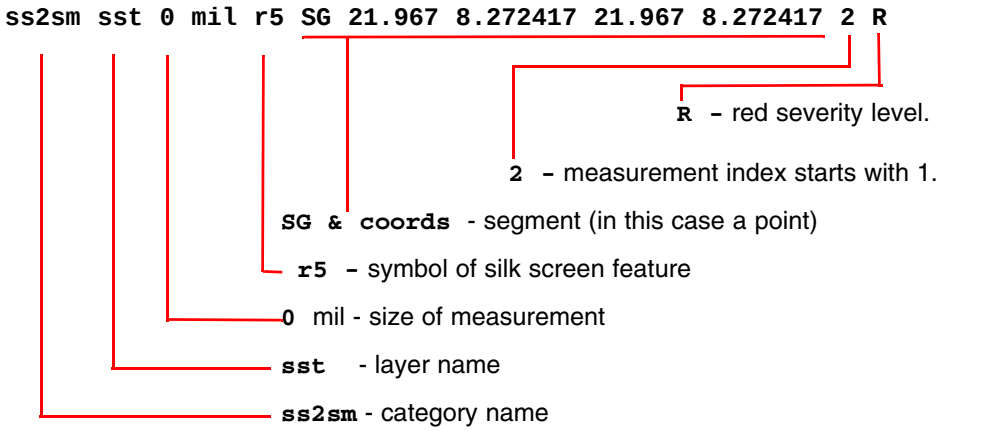
For some data types, line **MEAS** (measurements, for further details see The Results Viewer chapter in Doc.0501 Checklist Operations) you can further classify the output by specifying a category or a layer.

```
COM info,args=-t check -e $JOB/$STEP/my_check -d MEAS \
-o action=1+category=p2c, out_file=/tmp/info,write_mode=replace
cat /tmp/info
```

Output

```
p2c sigt 5.04 mil r60 r12 SG 0.3 -3.07 0.3 -3.06496 1 R
p2c sigt 4.91 mil r60 r12 SG 2.4 -3.33 2.4 -3.33491 1 R
p2c sigt 4.91 mil r60 r12 SG 3.3 -3.33 3.3 -3.33491 1 R
p2c sigt 4.91 mil r60 r12 SG 3.9 -3.33 3.9 -3.33491 1 R
p2c sigt 4.91 mil r60 r12 SG 2.5 -3.33 2.5 -3.33491 1 R
p2c sigt 3.36 mil r60 r12 SG 4.06422 -3.34691 4.06422 -3.35027 1 R
p2c sigt 3.36 mil r60 r12 SG 1.91113 -3.9749 1.91113 -3.97826 1 R
p2c sigt 3.36 mil r60 r12 SG -0.38294 -3.06496 -0.38294 -3.06832 1 R
p2c sigt 3.37 mil r60 r12 SG 0.92429 -1.01439 0.92429 -1.01776 1 R
```

Output Description



Shape Types

Shapes are represented in the output data as follows:

- LN - line (size = width)
- AR - arc (size = width)
- CU - curve (size = smaller size of bounding box limits)

- RC - rectangle (size = smaller size of bounding box limits)
- SQ - square (size = size of bounding box limits)
- PT - point (size = 0)
- SCR - circle (size = diameter)
- SG - segment (size = segment length)

Severity Level Extraction

In order to extract the severity level of measurements, the parameter:

`-o severity=<x>`

can be specified under the **`-d MEAS`** parameter.

This allows the extraction of measurements with severity <x>, where <x> is a combination of the characters R, Y, G, B, O (Red, Yellow, Green, Blue, Other).

In each output the severity of the measurement is indicated by one of the following letters: (R)ed, (Y)ellow, (G)reen, (B)lue or (O)ther)

Examples:

Command:

```
COM info,args=-t  check -e $JOB/pcb/$CHK_NAME -d MEAS \
-o action=2+category=p2c+severity=R,out_file=/tmp/mk1, \
write_mode=replace
```

Output response:

```
p2c top 3.75 mil rect100x65 r6 SG 4.646273 7.335 4.65 7.335 1 R
```

Note that only red measurements are listed. R is appended to the end of the line.

Command:

```
COM info,args=-t  check -e $JOB/pcb/$CHK_NAME -d MEAS \
-o action=2+category=p2c+severity=RYG,out_file=/tmp/mk1,\
write_mode=replace
```

Output response:

```
p2c bottom 7 mil r40 r6 SG 4.752 6.866 4.752 6.873 1
p2c top 3.75 mil rect100x65 r6 SG 4.646273 7.335 4.65 7.335 1
p2c top 7.607 mil rect100x65 r6 SG 4.755379 7.270379 4.75 7.265 1
p2c top 5 mil r24 r6 SG 4.8 7.180 4.8 7.185 1
```

Note that the last letter indicates severity.

Specifying where index is located

The option **`index`** (**`-o index`**) was added to the **MEAS** data type of the "info" command. When this option is specified, each measurement contains its index at the end of the line, just after the severity sign. An example is given below.

```
COM info,out_file=$GENESIS_TMP/info.out.$$, \
args= -t check -e $JOB/$STEP/fabrication -d MEAS \
-o action=2+category=npth2p+severity=R+index
```

Checklist Data Types

The following data types are available for checklists:

CHK_ATTR	
Options	The name of the attribute wanted may be indicated. If no attribute name appears, then all attribute values will be returned. More than one attribute name may appear.
Output	Three arrays are returned: 1. Name of attribute 2. Value of attribute 3. Indication if attribute exists

NUM_ACT	
Options	None
Output	The number of actions in the checklist. This is the only type which does not require specifying an action.

LAST_TIME	
Options	None (besides action number)
Output	The date in the form: "dd mmm yy"

DURATION	
Options	None (besides action number)
Output	The duration of the check in seconds

STATUS	
Options	None (besides action number)
Output	One of the following: - UNKNOWN - if for some reason the run status is unknown - UNDONE - the action has not been run yet - OUTDATE - the action has been run, but since then, its parameters have changed - DONE - the action completed successfully - ALARM - for future use - ERROR - the action completed with an error (no results available)

TITLE	
Options	None (besides action number)
Output	The title which appears at the top of the action window

MEAS	
Options	layer=lname - indicating that the results of only one layer will be returned category=cname - indicating that the results of only one category will be returned
Output	The measurements of an action as a text file. Each line of the text file represents one measurement of the action. Each line is of the form: category layer measurement location

This file is intended for further processing, to allow full customization of DFM reports.

REPORT	
Options	None (besides action number)
Output	The full text of the report of the action.

The DO_INFO shortcut

It is recommended to define an alias **DO_INFO** in the file:

$GENESIS_DIR/sys/hooks/script_start.csh

The alias will be defined as follows:

```
alias DO_INFO 'set IFILE = /tmp/info.$$; \\
COM info,out_file=$IFILE,write_mode=replace,args= \!:* ; \\
source $IFILE; rm $IFILE'
```

This alias can be activated in scripts as in the example below:

```
DO_INFO -t job -e $JOB -d STEPS_LIST
```

As a result, the variable **gSTEPS_LIST** will be set automatically.

Summary of Usage

To summarize, the format of the command is:

COM info, out_file = <fpath>, \

write_mode = [replace|append], \

units = [inch|mm], \

args = <args>

args:== -t <entity_type>

-e <entity_name>

-d <data_type>

-m [script|display]

-p parameters

-o = options

Finally, to obtain the full help text (formatted below) enter the script:

```
COM info, out_file=/tmp/info , write_mode=replace, args= -help
```

Extracting Information - Examples

In this section, we will show examples of extracting information from Genesis jobs using the '**info**' command. The examples are to make it easier to understand the structure of the script. Each script generates a file containing the information requested, as listed in the Output section.

The Root Entity

Script

```
COM info,out_file=/tmp/example,args= -t root
```

will retrieve a list of all jobs available in the Genesis 2000 database.

Output

```
set gJOBS_LIST = ('job.1' 'job.2' 'job.3')
```

shows all three jobs in the user database.

The Job Entity

Script

```
COM info,out_file=/tmp/example,args= -t job -e e.demo.z01 -d
symbols_list
```

will retrieve a list of all symbols defined in the specific job - in this example: **e.demo.z01**.

Sample Output

```
set gSYMBOLS_LIST = ('rect75x25_225' 'rect75x25_225+1'
'rect75x25_315')
```

Other information available about the job entity could be its steps_list, wheels_list, matrix_list, stackups_list, templates_list and attributes.

Note that when you request the attribute data_type, (**-d attr**), you get two lists: one containing the attribute name, the other the attribute value,

Script

```
COM info,out_file=/tmp/example,args= -t job -e e.demo.z01 -d
attr
```

will retrieve these two lists:

```
set gATTRname = ('.technology' '.primary_side')
set gATTRval  = ('TT4'         'top')
```

The Step Entity

Enables the user to receive data related to a specific step in a job.

Script

```
COM info,out_file=/tmp/example,args= -t step -e e.demo.z01/pcb
```

will retrieve a file containing data relevant to the '**pcb**' step in job **e.demo.z01** .

Output

```
set gDATUMx = '0'
set gDATUMy = '0'
set gLIMITSxmin = '-8.5'
set gLIMITSymin = '-5.54299'
set gLIMITSxmax = '8.5'
set gLIMITSymax = '5.5'
.
.
.
set gPROF_LIMITSxmin = '-3.2'
set gPROF_LIMITSymin = '-2.8'
set gPROF_LIMITSxmax = '4.4'
set gPROF_LIMITSymax = '2.8'
set gACTIVE_AREAxmin = '-3.2'
set gACTIVE_AREAymin = '-2.8'
set gACTIVE_AREAxmax = '4.4'
set gACTIVE_AREAymax = '2.8'
set gPROF_LENGTH = '2.3622051'
set gLAYERS_LIST = ('comp_+_top' 'top_paste' 'comp_resist'
'component')
set gCHECKS_LIST = ()
set gATTRname = ('.out_drill_full' '.out_drill_optional'
'.out_rout_optional')
set gATTRval  = ('no'              'no'                  'no')
```

The Symbol Entity

Enables the retrieval of data related to a specific symbol in a job, such as number of arcs, lines, surfaces etc. that created the symbol.

Script

```
COM info,out_file=/tmp/example,args= -t symbol -e e.demo.z01/
my_symbol
```

Retrieves all data regarding symbol name '**my_symbol**' in job **e.demo.z01**

Sample Output

```
set gLIMITSxmin = '-0.035355'
set gLIMITSymin = '-0.035355'
set gLIMITSxmax = '0.035355'
set gLIMITSymax = '0.035355'
set gFILLdx = '0.1'
set gFILLdy = '0.1'
```

```
set gSYMS_HISTsymbol = ()
set gSYMS_HISTline   = ()
set gSYMS_HISTpad    = ()
set gSYMS_HISTarc    = ()
set gFEAT_HISTline  = '0'
set gFEAT_HISTpad   = '0'
set gFEAT_HISTsurf  = '1'
set gFEAT_HISTarc   = '0'
set gFEAT_HISTtext  = '0'
set gFEAT_HISTtotal = '1'
set gATTRname = ('.out_break' '.out_scale' '.break_away')
set gATTRval  = ('no'         'no'         'no')
```

The EDA Entity

Enables the retrieval of data related to an EDA object in a job. This is applicable only to jobs that were sent from the Valor Enterprise 3000 system.

Script

```
COM info,out_file=/tmp/example,args= -t eda -e e.demo.z01/pcb -o
bottom
```

will retrieve all data regarding the EDA object, on the design's top side.

Note The TH (Types Histogram) is loaded into memory and reflects each site's defined **comp_type** attribute values.

Sample Output

```
set gNCOMPS = '3'
set gTHaxial       = '0'
set gTHbga         = '0'
set gTHcbga        = '0'
.
.
.
set gTHtsoic       = '0'
set gTHtsop        = '0'
set gCOMPpart      = ('C-10-1000' 'C-30-0003' 'CO-10-1032')
set gCOMPrefdes    = ('C1'        'C2'        'MP2')
set gCOMPpackage   = ('8113Z'     'C1210'     'CON32)
set gCOMPcentroidx = ('-0.3'      '-0.3'      '4.3')
set gCOMPcentroidy = ('0.5'       '1.1'       '-1.6')
set gCOMPnpins     = ('2'         '2'         '32')
set gCOMPpin0x     = ('-0.3'      '-0.305'    '4.3')
set gCOMPpin0y     = ('0.5'       '1.1'       '-1.6')
set gCOMPside      = ('top'       'top'       'top')
set gCOMProtation  = ('  0.00'    '  0.00'    '270.00')
set gCOMPmirror    = ('0'         '0'         '0')
set gCOMPcolor     = (''          ''          '')
set gCOMPserial    = ('0'         '1'         '2')
set gCOMPlimitminx = ('-0.325'    '-0.34'     '4.175')
set gCOMPlimitmaxx = ('-0.175'    '-0.135'    '4.425')
set gCOMPlimitminy = ('0.45'      '1.02'      '-1.65')
set gCOMPlimitmaxy = ('0.55'      '1.18'      '1.53')
```

The Comp Entity

Enables the retrieval of details regarding a specific component. Important data that the user must have to use this option is the component side and serial.

These data items could be obtained using the EDA entity queries.

Script

```
COM info,out_file=/tmp/example,args= -t comp -e e.demo.z01/pcb/
top/1
```

will retrieve a file with all data on a component on the top side, with **serial** = 1.

Sample Output

```
set gPART = 'C-10-1000'
set gREFDES = 'C1'
set gPACKAGE = '8113Z'
set gX = '-0.3'
set gY = '0.5'
set gNPINS = '2'
set gSIDE = 'top'
set gROTATION = '  0.00'
set gMIRROR = '0'
set gCOLOR = ''
set gSERIAL = '1'
set gPIN0X = '-0.3'
set gPIN0Y = '0.5'
set gLIMITMINX = '-0.325'
set gLIMITMAXX = '-0.175'
set gLIMITMINY = '0.45'
set gLIMITMAXY = '0.55'
set gPINnum = ('0'    '1'   )
set gPINx   = ('-0.3' '-0.2')
set gPINy   = ('0.5'  '0.5' )
set gATTRIBUTEname = ('.comp_mount_type')
set gATTRIBUTEval  = ('thmt'            )
set gPROPERTIESname = ('Package' 'Tolerance' 'Value')
set gPROPERTIESval  = ('8113Z'   '5%'        '10pf')
```

The Layer Entity

This entity provides data relating to a specific layer in the job.

Script

```
COM info,out_file=/tmp/example,args= -t layer -e e.demo.z01/
pcb/component
```

will retrieve all available information about the layer called '**top**', in step '**pcb**' in job '**e.demo.z01**'

Sample Output

```
set gCONTEXT = 'board'
set gTYPE = 'signal'
set gPOLARITY = 'positive'
set gSIDE = 'top'
set gDRL_START = ''
set gDRL_END = ''
```

```
set gFOIL_SIDE = 'top'
set gSHEET_SIDE = 'top'
set gROW = '4'
set gLIMITSxmin = '-3.125'
set gLIMITSymin = '-2.725'
set gLIMITSxmax = '4.45'
set gLIMITSymax = '2.725'
set gSYMS_HISTsymbol = ('r10' 'r12' 'r14')
set gSYMS_HISTline   = ('168' '144' '94')
set gSYMS_HISTpad    = ('18'  '32'  '20')
set gSYMS_HISTarc    = ('0'   '0'   '0')
set gSLOT_HISTsymbol = ('r10' 'r12' 'r14')
set gSLOT_HISTlen    = ('12.5' '10' '35.4')
set gSLOT_HISTangle  = ('35.00''15.00' '65.00')
set gSLOT_HISTcount  = ('40'   '3'   '22')
.
.
.
.set gFEAT_HISTtotal = '680'
set gNUM_TOOL = '11'
set gTOOLnum         = ('1'      '2'      '3')
set gTOOLcount       = ('9'      '168'    '144')
set gTOOLtype        = ('plated' 'non-plated' 'via')
set gTOOLtype2       = ('standard' 'press_fit' 'laser')
set gTOOLshape       = (’hole’ ’slot’)
set gTOOLslots       = (’yes’ ’no’ ’by length’
set gTOOLslot_len = ’ ’
.
.
.
set gTOOLbit         = (''       ''       '')
set gTOOL_THICK = '0'
set gTOOL_USER = ''
set gATTRname = ('.out_mirror' '.lpol_done' '.cu_base' )
set gATTRval  = ('no'          'no'         'no')
set gNCSETS_LIST = ()
set gAOISETS_LIST = ()
set gNCRSETS_LIST = ()
set gLPDwas_input   = 'no'
set gLPDis_defined  = 'no'
set gLPDpolarity    = 'positive'
set gLPDspeed       = '0'
set gLPDxstretch    = '0.0000'
set gLPDystretch    = '0.0000'
.
.
.
set gLPDmedia       = 'first'
set gLPDresolution  = 'half_mil'
set gLPDsmoothing   = 'smooth'
set gLPDswap_axes   = 'no_swap'
set gLPMpolarity    = ('positive' 'positive' 'positive'
'positive' 'positive')
```

The Wheel Entity

Retrieves data related to a specific wheel defined in the job.

Script

```
COM info,out_file=/tmp/example,args= -t wheel -e e.demo.z01/
1.sym
```

Sample Output

```
set gDCODEnum    = ('10'  '11'  '12'  '13')
set gDCODEsymbol = ('r12' 'r15' 'r50' 'r25')
set gDCODEangle  = ('0'   '0'   '0'   '0')
set gDCODEmirror = ('no'  'no'  'no'  'no')
```

The Matrix Entity

Retrieves data that is available through the job matrix.

Script

```
COM info,out_file=/tmp/example,args= -t matrix -e e.demo.z01/
matrix
```

Sample Output

```
set gNUM_ROWS = '42'
set gNUM_COLS = '5'
set gNUM_LAYERS = '42'
set gNUM_STEPS = '1'
set gROWrow        = ('1'          '2'            '3')
set gROWtype       = ('layer'      'layer'        'layer')
set gROWname       = ('comp_+_top' 'top_paste'
'comp_resist')
set gROWcontext    = ('board'      'board'        'board')
set gROWlayer_type = ('components' 'solder_paste'
'solder_mask')
set gROWpolarity   = ('positive'   'positive'     'positive')
set gROWside       = ('top'        'top'          'top')
set gROWdrl_start  = ('' '' '')
set gROWdrl_end    = ('' '' '')
set gROWfoil_side  = ('none'       'none'         'none')
set gROWsheet_side = ('none'       'none'         'none')
set gCOLcol       = ('1'    '2'     '3'     '4'     '5'    )
set gCOLtype      = ('step' 'empty' 'empty' 'empty' 'empty')
set gCOLstep_name = ('pcb'  ''      ''      ''      ''     )
set gATTRname = ()
set gATTRval  = ()
```

The Attribute Entity

Enables the retrieval of available attributes.

Script

```
COM info,out_file=/tmp/example,args= -t attributes -e
e.demo.z01 -d atr -p name
```

gets all attribute names available.

Sample Output

```
set gATRname = ('.smd' '.gold_plating' '.n_electric')
```

The Notes Entity

Enables the retrieval of electronic notes data in a job.

Script

```
COM info,out_file=/tmp/example,args= -t notes -e e.demo.z01/
pcb/component/notes
```

will retrieve all data related to electronic notes in the '**e.demo.z01**' job, in the '**pcb**' step, in the '**component**' layer.

The data are the electronic note date and time of creation, the user that created the note, the note location and the text in the note.

Sample Output

```
set gNOTEdate = ('03/06/98'            '03/06/98'        )
set gNOTEtime = ('11:37:57'            '12:08:00'        )
set gNOTEuser = ('zachi'                'zachi'          )
set gNOTEx    = ('-0.657454'           '2.417196'        )
set gNOTEy    = ('-1.411615'           '1.459753'        )
set gNOTEtext = ('This is a test note' 'Second test note')
```

The Stackup Entity

Enables the retrieval of stackup data from a job.

Script

```
COM info,out_file=/tmp/example,args= -t stackup -e te/stackup
```

will retrieve all stackup data in the job '**te**'.

Sample Output

```
set gWIDTH = '18'
set gHEIGHT = '24'
set gTARGET_THICK = '112'
set gTARGET_POS_TOL = '12'
set gTARGET_NEG_TOL = '12'
set gVENDOR = 'Any'
set gCOST = '11.983'
set gTHICK = '105.7'
set gIS_MIRROR = 'yes'
set gCONSTRUCT = 'FR-4'
set gPLATE_THICK = '0'
set gMASK_THICK = '0'
set gTHICK_TYPE = 'Laminate'
set gPILEcost       = ('0.091'      '1.58'    '1.58')
set gPILEmaterial   = ('Foil'       'Prepreg' 'Prepreg')
.
.
.
set gPILEupsidedown = ('no'         'no'      'no')
set gPRESSEDmaterial = ('Foil' 'Prepreg' 'Prepreg')
set gPRESSEDthick    = ('1'    '1.8'     '1.8')
set gPRESSEDknown    = ('yes'  'yes'     'yes')
```

```
set gLAYERside   = ('Top' 'Inner' 'Inner')
set gLAYERserial = ('1'   '4'     '4')
set gLAYERname   = ('l1'  'l2'    'l3')
set gVALID = 'Unknown_Material'
set gIMPmodel       = ('Surface_Microstrip' 'Dual_Stripline'
'Dual_Stripline')
set gIMPref1        = ('l2'                 'l3'
'l3')
set gIMPref2        = (''                   'l6'
'l6')
set gIMPlyr         = ('l1'                 'l4'
'l5')
set gIMPdual_lyr    = (''                   ''
'')
.
.
.
set gIMPorig_space  = ('0'                  '0'
'0')
set gIMPcurr_space  = ('0'                  '0'
'0')
set gIMPwidth_plus  = ('2'                  '1'
'1')
set gIMPwidth_minus = ('2'                  '1'
'1')
set gALLOW_CHANGE = '1Inch  '
set gWIDTH_VAR = '1Inch  '
set gETCH_FACTOR = '1'
set gMASK_ER = '1'
set gFREQ = '1'
set gRESIN_ER = '0'
```

The NCRset Entity

Enables the retrieval of **ncrsets** data associated with the job.

Script

```
COM info,out_file=/tmp/example,args= -t ncrset -e tmp/panel/
rout/new
```

will retrieve **ncrset** data in the '**tmp**' job, in the '**panel**' step, in the '**rout**' layer called '**new**'

Sample Output

```
set gMACHINE = 'excellon'
```

The NCset Entity

Enables the retrieval of data regarding ncsets associated with the job.

Script

```
COM info,out_file=/tmp/example,args= -t ncset -e tmp/panel/
drill/new
```

will retrieve **ncset** data in the job '**tmp**', in the '**panel**' step, '**drill**' layer that is called '**new**'

Sample Output

```
set gMACHINE = 'trudrill'
set gNUM_SPLITS = '0'
set gNUM_STAGES = '0'
```

The Check Entity

Enables the retrieval of data regarding checklists associated with the job. Note that you must specify the '**-o action=n**' parameter, where **n** is the number of actions.

Note The above is not true of the "**NUM_ACT**" data type.

Script

The next script line:

```
COM info,out_file=/tmp/example,args= -t check -e dic.001/pcb/
analysis -d last_time -o action=1
```

will retrieve a file containing the last time the action in the checklist was run, in the form:

Sample Output

```
set gLAST_TIME = '10 Mar 98 03:52 PM'
```

The Panel_classes Entity

Enables the retrieval of data regarding **panel_classes** (classes defined in **genesislib** that describe different types of panels).

Script

```
COM info,out_file=/tmp/example,args= -t panel_classes -e
e.demo.z01
```

Sample Output

```
set gNUM_CLS = '21'
set gCLSname      = ('18x24' '20x20-min' '21x24')
set gCLSwidth     = ('18'    '20'        '21')
set gCLSheight    = ('24'    '20'        '24')
set gCLSt_active  = ('1'     '1'         '1')
set gCLSb_active  = ('1'     '1'         '1')
set gCLSl_active  = ('1'     '1'         '1')
set gCLSr_active  = ('1'     '1'         '1')
set gCLSsr_margin = ('0.2'   '0.2'       '0.2')
set gCLSsr_orient = ('any'   'any'       'any')
```

Coordinates

Enables the retrieval of X,Y coordinates of any feature/s.

Script

```
COM info,args=-t layer -e dic.001.bini/pcb/olt -d FEATURES \
,out_file=/tmp/bini_f, write_mode=replace
```

Sample Output

Returns the following feature file:

```
### Layer - olt features data ###
#P 5.55002 6.748 r51 P 0 0 N
#P 14.31002 6.748 r51 P 0 0 N
#P 5.55002 8.098 r51 P 0 0 N
```

Where the numbers after **#P** indicate the X,Y coordinates of that feature.

A script can contain an Info command to retrieve the coordinates of any feature in the selected area.

Info Command for AOI Interface

The line mode command **info** is a powerful tool used to retrieve information from database entities to a file. **Info** works on entities in memory, thus its output represents the entity current data, even if the entity was not yet saved to disk.

This section describes a new **info** command made specifically for the Genesis to AOI Interface.

Entity name:$JOB/$STEP/$LAYER/$CDRSET

Entity: orbotech-aoiset

Options:

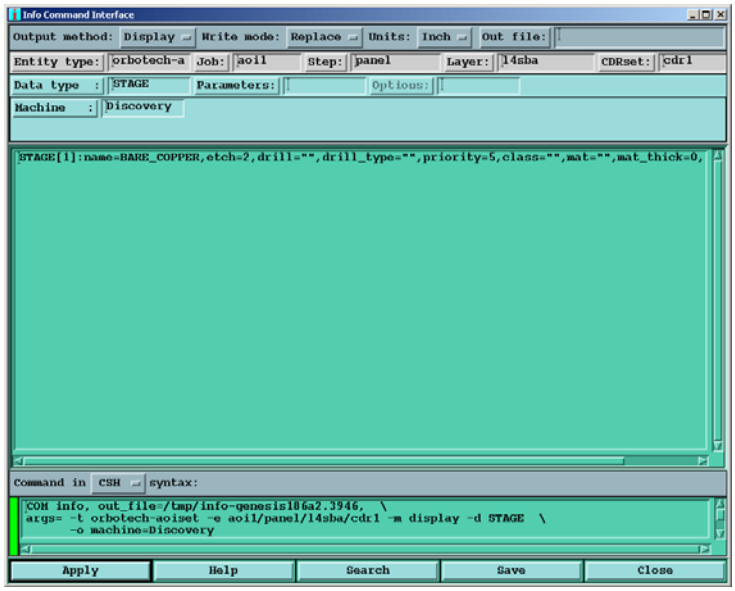
machine name of machine: vision, pc14, inspire, infinex, discovery (Case insensitive)

stage name of stage from configuration file (e.g. TIN_LEAD_BEFORE)

Data Type	Parameters	Description	Required options
STAGE		Information about stages (lists)	
	name		machine
	etch		machine
	drill		machine
	drill_type		machine
	priority		machine
	class		machine
	mat		machine
	mat_thick		machine
	cu_weight		machine
FTRS		Information about layer design	
	nom_line		
	min_line		
	nom_space		
	min_space		
	nom_ar	nominal annular ring	
	min_ar	minimal annular ring	
	nom_nfp_d	nominal nonfunctional pad spacing	
	min_nfp_d	minimal nonfunctional pad spacing	
	nom_plane_d	nominal place spacing	
PRMS		AOISET parameters	
	type		

Data Type	Parameters	Description	Required options
	pattern		
	pol		
	db		machine
	rule		machine
ALIGN		Alignment transformation	
	mirror		machine, stage
	rotate		machine, stage
	offset_x		machine, stage
	offset_y		machine, stage
ALIGN_TGT	x,y	List of alignment targets	stage
VRS_TGT	x,y	List of vrs targets	
DISH_DOWN_TGT	x,y	List of dish down targets	
REG_TGT	x,y	List of registration targets	
CALIB_TGT	x,y	List of calibration targets	
SCAN_ALIGN	x,y	List of scan alignment targets	
IMPED_TGT	x,y	List of impedance targets	
THICK_TGT	x,y	List of thickness targets	
INSPECT		Bounding box of inspection & exclusion area in panel coordinates	
	xmin		
	xmax		
	ymin		
	ymax		

Here you can see some of these new parameters in action in the Info Command Generator.



Chapter 7 *Info Command Interface*

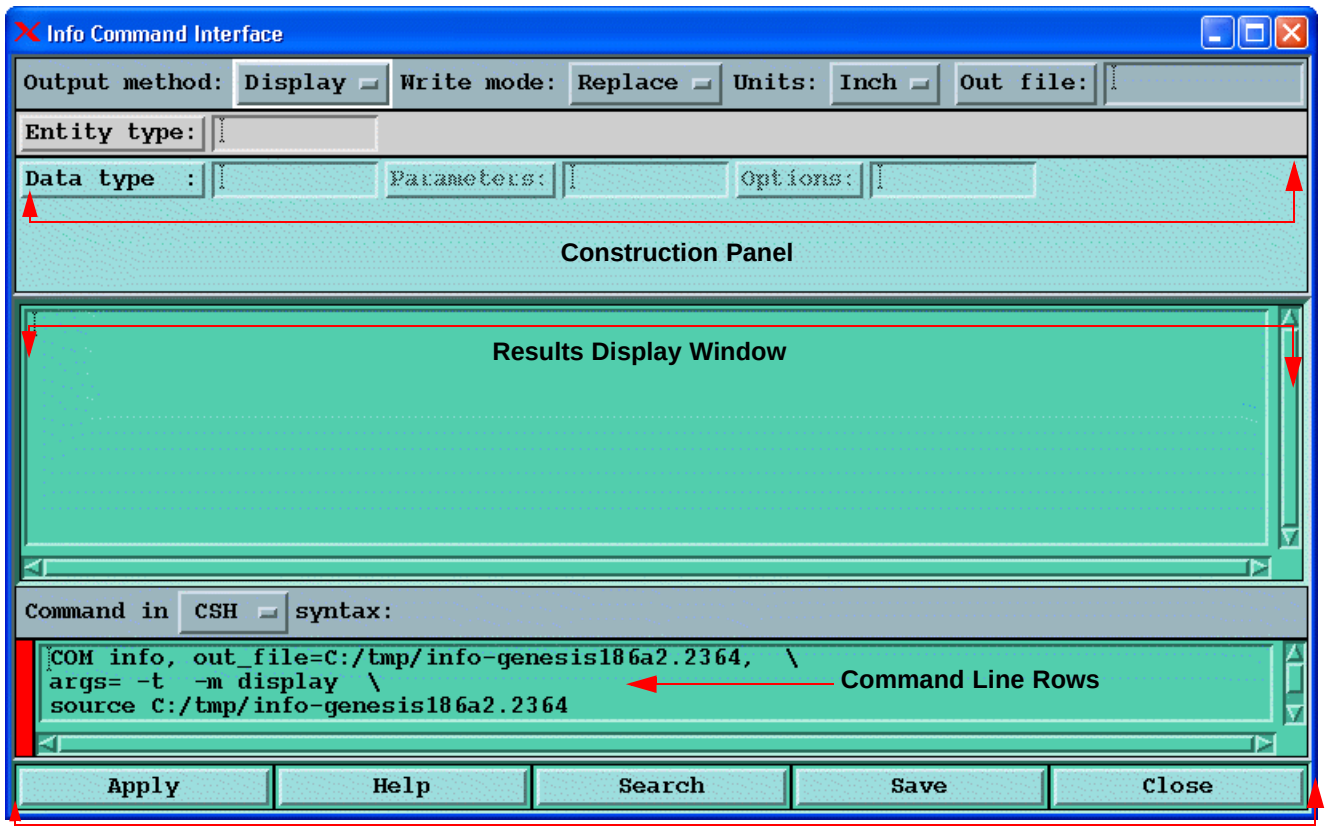
Overview

The line mode command '**Info**' is a powerful tool used to retrieve information from database entities to a file. The Info command may be composed by using the Info Command Interface (described here), or by building the info command as you would any line mode command or script. After execution of the Info command with either method, the results are identical.

This method of building an info command involves selecting options from menus and clicking **Apply** to execute; it does not require knowledge of the syntax of the command.

Opening the Info Command Interface Window

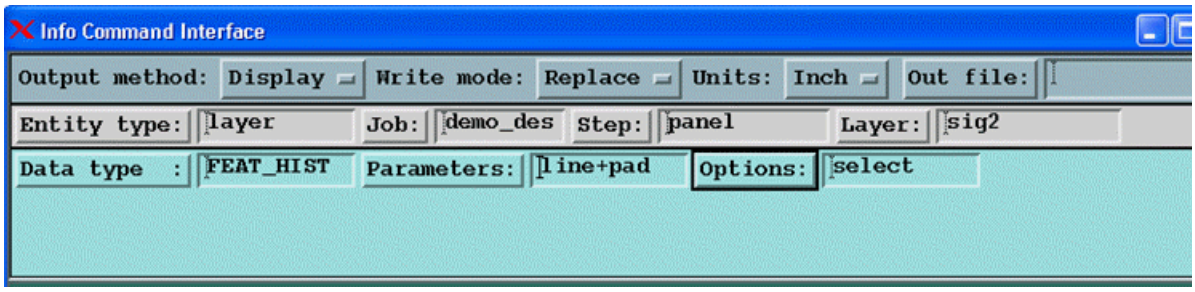
From the Engineering Toolkit window, select **Actions > Info...**
The Info Command Interface window appears.



Info Command Construction Panel

The top panel of the Info Command Interface window contains buttons that allow you to specify the parameters for the Info command. There is no 'standard' command panel. Each Info command is a different combination of Entity Type and Data Type, so the fields in this panel change accordingly. Below is an example of one such combination,
where **Entity Type = layer**, and **Data Type = FEAT_HIST**.

a:
b:
c:



a. **`Output specification:`** buttons/fields to define output parameters.

b. **`Entity specification:`** buttons/fields relating to the Entity Type.

c. **`Data type:`** offers Parameters & Options.

Output Specification Section

This row consists of buttons that specify output parameters.

Output Method - Determines the method used to display the results in the middle panel (after clicking Apply).

Display- to display results of the Info command.

Script- to display results in a format that you can incorporate into scripts.

Write Mode - Determines how additional results are added to the Output File (output file path is specified in the **`Out file`** field).

Replace - Replaces the data in the output file with the new results (existing data is overwritten by the new data).

Append - Adds the new data to the end of the existing data in the output file.

Units (Inch/MM) - Units of measurement used when specifying X and Y locations, and other distance-dependent data.

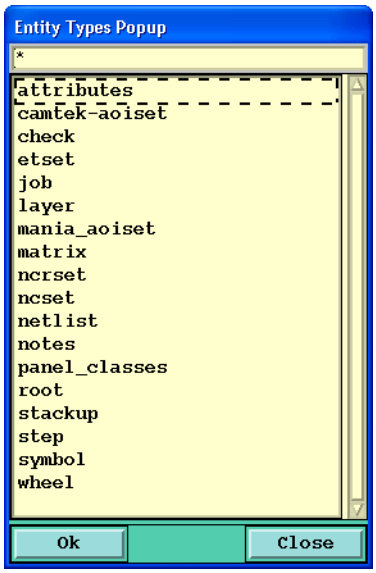
Out File - The path to the output file in which the results of the Info command are written. The same results are display in the middle panel (after clicking **Apply**).

If an Out file is *not* defined, data displayed in the results window is always replaced, unless otherwise defined.

Entity Definition Section

This row defines the entity type and entity path to be inspected by the Info command.

Most of the buttons you click will display a popup with a list of relevant items. Since the popups are similar to one another, only the Entity Type popup (with all available types) is shown here as an example.



Entity Type - Defines an entity.

Job - Name of the job.

Step/Layer - Names of the step and layer from which to extract the information. These fields may not appear in all cases, depending on the type of entities (symbol, stack, checklist, Ncset, etc.) being defined.

Notes

1. If you leave an entity name field blank, the default name, such as $JOB, $STEP, etc. will be used.
2. You can also make your own custom names for entities, preceded by the symbol **'$'**. For example: **`$new_job`**.
3. If you define a new entity without the preceding **'$'** symbol, a warning message informs you that you are creating a new entity, and then its name appears in the command window.
4. If any of the path fields is empty or not defined, the bar on the left side of the command window appears in RED. This indicates that the command is not complete, and cannot be executed by clicking the Apply button.
   See "Command Line Rows" on page 67 for an example.

Data Type Definition Section

This row defines additional parameters and options to be inspected by the Info command.

Data Type - Some Data Types have parameters, some also have options, and some have neither parameters or options. Only relevant fields will be displayed for each data type.

Parameters - Parameters that are available for the selected Data Type.

Options - Options that are available for the selected Data Type.

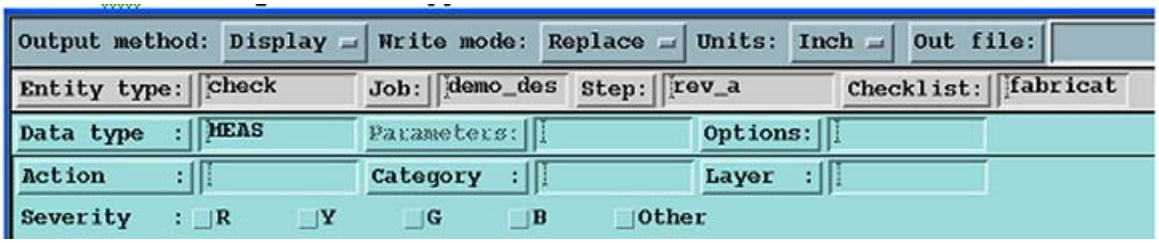
Serial - The serial number used for array Data Types. It specifies a single array entry; if not specified, the whole array is displayed.

Note In the Parameters/Options fields, you may select more than one item. The items will appear in the field box separated by '+' signs, as they would in a standard Info line mode command.

Example If you were to select these parameters…

- Entity Type = check (checklist)
- Data Type = meas

the following fields will appear…



Checklist - the checklist from which to retrieve information.

Action - the Action incorporated in the checklist you selected in the Check field (mandatory field).

Category - the category from which you wish to display its results.

Layer - the layer from which to extract the information.

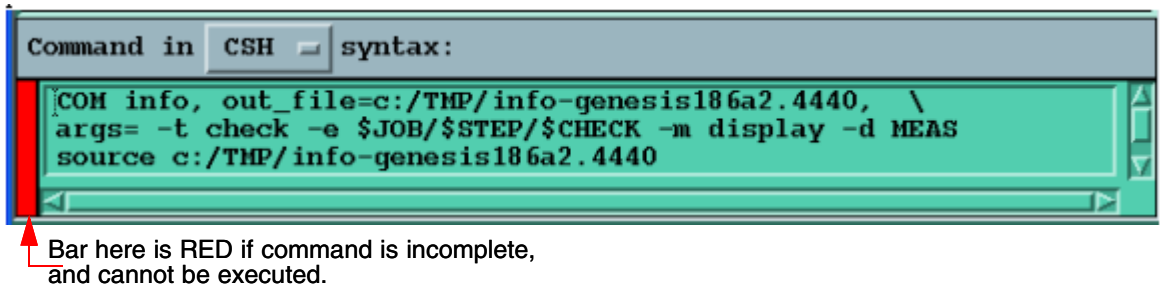
Severity - the Severity level to display. The choices are **R**ed, **Y**ellow, **G**reen, **B**lue, or **O**ther.

Notes

1. If any field is empty or not defined, the rows of available additional options (here, the action and severity rows) will not appear.
2. If any of the path fields is empty or not defined, the bar on the left side of the command window appears in RED. This indicates that the command is not complete, and cannot be executed by clicking the Apply button.

Command Line Rows

Shows the Info command that was generated using the values defined in the command construction panel. You can also define the command syntax (**CSH, Perl, Tcl**) to be used.

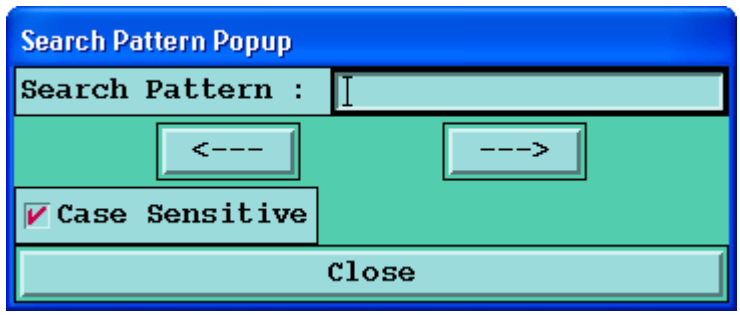


Action Buttons Row

Apply - To execute the parameters you specified in the Info Command Construction (top) panel, and display the results in the middle panel.
A red bar in the command line row indicates that the command has not been completed, and cannot be executed by clicking the Apply button.

Help - To display an explanation of the fields and options in the middle panel.

Search - To display a popup in which you can enter keywords to search within the current results displayed in the middle panel.



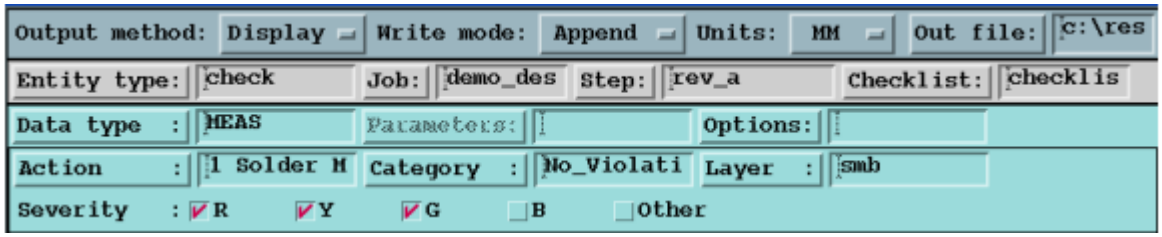
Save - To save the data to a file.

Data may be saved from the **command** window as well as from the **results** window. If the file defined in the field already exists, new data may either **replace** the existing data, or be **appended** to the end of the file.

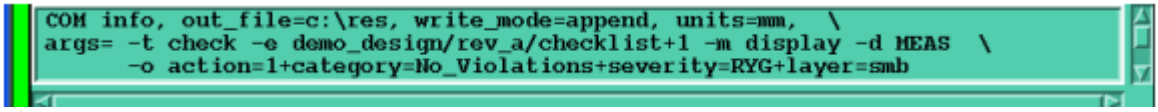
Close - To close the Info Command Interface window.

Sample Info Command and Output

Using these Data type definitions



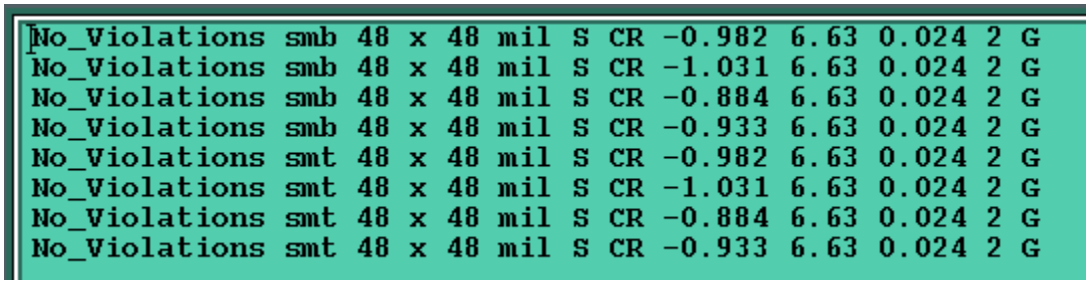
creates the Info command displayed below in the Command window.



Bar here is GREEN, indicating command is complete, and can be executed.

This command may be copied and saved in a script file, or saved with the **Save** command.

The result of this Info command is given in the Results window displayed below. All results may be saved with the **Save** command.



Updated List of Entity Types for the Info Command

The line mode command **info** is a powerful tool used to retrieve information from database entities to a file. Info works on entities in memory, thus its output represents the entity's current data, even if the entity was not yet saved to disk.

Entity types are one part of the parameters required for an **info** command. For details, see "Entity Types" on page 34.

-t entity_type - One of the following types:

attributes	layer	netlist	step
camtek_aoiset	mania_aoiset	notes	symbol
check	matrix	panel_classes	wheel
etset	ncrset	root	
job	ncset	stackup	

The following table describes all supported entities, their data types, and parameters.

:

Entity	Data Type	Parameters
attributes	EXISTS FORCE_LIB NUM_ATR ATR (array)	- (Options: deleted) - name, changed, context, type, entity, def, min_txt_len, max_txt_len, options, options_del, min_int_val, max_int_val, min_flt_val, max_flt_val (Options: deleted)
camtek_ aoiset	EXISTS NUM_X_FRAMES NUM_Y_FRAMES SCAN_AREA	- xmax, xmin, ymax, ymin
check	CHK_ATTR (array) DURATION ERF EXISTS LAST_TIME MEAS (free text output) MEAS_DISP_ID (free text output) NUM_ACT REPORT (free text output) STATUS TITLE	- name, val, exists

Entity	Data Type	Parameters
etset	ADAPTER_LIM	- xmin, ymin, xmax, ymax
	ADAPTER_NAME	
	EXISTS	
	GRID_LIM	- xmin, ymin, xmax, ymax
	SPLITS_HIST	
	TESTER_TYPE	
job	ATTR (array)	- name, val
	CHANGES	
	EXISTS	
	FLOWS_LIST (array)	
	FORMS_LIST (array)	
	IS_CHANGED	
	MATRIX_LIST (array)	
	STACKUPS_LIST (array)	
	STEPS_LIST (array	
	SYMBOLS_LIST (array)	
	TEMPLATES_LIST (array)	
	WHEELS_LIST (array)	

Entity	Data Type	Parameters
layer	CONTEXT	
	TYPE	
	POLARITY	
	SIDE	
	DRL_START	
	DRL_END	
	FOIL_SIDE	
	SHEET_SIDE	
	ROW	
	LIMITS	- xmin, ymin, xmax, ymax
	SYMS_HIST (array)	- symbol, line, pad, arc (options: break_sr)
	FEAT_HIST	- line, pad, surf, arc, text, total (options: select, break_sr)
	SLOT_HIST **(1)**	
	NUM_TOOL	
	TOOL (array)	- num, count, type, type2, min_tol, max_tol, finish_size, drill_size, bit - (Options: break_sr)
	TOOL_THICK	
	TOOL_USER	
	ATTR (array)	- name, val
	FEATURES (free text output)**(2)**	- (Options: break_sr, break_feat, select)
	NCSETS_LIST (array)	
	AOISETS_LIST (array)	
	NCRSETS_LIST (array)	
	LPD	
		- adflash, advec, conductors1, conductors2, conductors3, conductors4, conductors5, copper_area, def_ext_lpd, device_type, enl_0_vecs, enl_by1 ... enl_by_10, is_defined, media, minflash, minvec, overlay, plot_kind1, plot_kind2, polarity, quality, res_units, res_value, resolution, smoothing, speed, swap_axes, sym_name1 ... sym_name10, was_input, xcenter, xcenter_inch, xmirror, xshift, xstretch, ycenter, ycenter_inch, ymirror, yshift, ystretch
	LPM	
		- adflash, advec, conductors1, conductors2, conductors3, conductors4, conductors5, copper_area, def_ext_lpd, device_type, enl_0_vecs, enl_by1 ... enl_by_10, is_defined, media, minflash, minvec, overlay, plot_kind1, plot_kind2, polarity, quality, res_units, res_value, resolution, smoothing, speed, swap_axes, sym_name1 ... sym_name10, was_input, xcenter, xcenter_inch, xmirror, xshift, xstretch, ycenter, ycenter_inch, ymirror, yshift, ystretch

- **(1)** Describes the Slot Histogram. The Slot Histogram describes the size, length, and angle of each slot, and the number of slots in a drill layer.
- **(2)** Contains option **feat_index**.Enables output of feature indexes for each feature. This option may be used together with other options.
  For example: -o selected+feat_index
  The indices can be used only while the job is open. They will not have any meaning after the job is closed.

Entity	Data Type	Parameters
mania_ aoiset	ANGLE EXISTS MIRROR RESOLUTION X_OFFSET Y_OFFSET	
matrix	ATTR (array) COL (array) EXISTS NUM_COLS NUM_ROWS NUM_LAYERS NUM_STEPS ROW (array)	- name, val - col, type, step_name - context, drl_end, drl_start, foil_side, layer_type, name, polarity, row, sheet_side, side, type
ncrset	EXISTS FORMAT MACHINE REG	 - decimal, name, nf1, nf2, tool_unit, units, zeroes - angle, mirror, scale_x_orig, scale_y_orig, version, xoff, yoff, xorigin, yorigin, xscale, yscale
ncset	EXISTS FORMAT MACHINE NUM_SPLITS NUM_STAGES REG	 - decimal, incr, name, nf1, nf2, rep, tool_unit, units, zeroes - angle, mirror, scale_x_orig, scale_y_orig, version, xoff, yoff, xorigin, yorigin, xscale, yscale
netlist	EXISTS NUM_NETS = Total number of nets END_PT_TOP = Number of end points on the board's top side END_PT_BOT = Number of end points on the board's bottom side END_PT_TH = End point through holes NET_PT_TOP = Number of net points on the board's top side NET_PT_BOT = Number of net points on the board's bottom side NET_PT_TH = Net point through holes	Example **COM info,out_file=$GENESIS_TMP/ info.out.$$, ** **args= -t netlist -e $JOB/$STEP/ <netlistname>** **-d <NUM_NETS, END_PT_TOP, END_PT_BOT, END_PT_TH, NET_PT_TOP, NET_PT_BOT, NET_PT_TH>** **<netlistname>** can be one of the following: **refnet**, **curnet**, **cadnet**, or **curcadnet**. For end point information, use **refnet**.
notes	EXISTS NOTE (array)	 - date, time, user, x, y, text

Entity	Data Type	Parameters
panel_ classes	EXISTS NUM_CLS CLS (array)	- name, width, height, t_active, b_active, l_active, r_active, sr_margin, sr_orient
root	EXISTS JOBS_LIST (array)	
stackup	ALLOW_CHANGE COPPER_LOSS CONSTRUCT COST ETCH_FACTOR EXISTS FREQ HEIGHT IMP (array)	- model, ref1, ref2, lyr, dual_lyr, imp, imp_pos_tol, imp_neg_tol, target, target_tol, orig_width, curr_width, orig_space, curr_space, width_plus, width_minus
	IS_MIRROR LAM_THICK LAYER (array) MASK_ER MASK_LAM_THICK MASKPLATE_THICK MASK_THICK PILE (array)	- name, oz, serial, side - cost, material, weave, thick, pos_tol, neg_tol, name, cat_num, vendor, upsidedown
	PLATED_THICK PLATE_THICK PRESSED (array) RESIN_ER SEQ_LAM TARGET_NEG_TOL TARGET_POS_TOL TARGET_THICK THICK THICK_TYPE VALID VENDOR VIA_PLATE_THICK WIDTH WIDTH_VAR	- material, thick, known - layer

Entity	Data Type	Parameters
step	ACTIVE_AREA	- xmin, ymin, xmax, ymax
	ATTR (array)	- name, val
	CHECKS_LIST (array)	
	DATUM	- x, y
	ETSETS_LIST	
	EXISTS	- xmin, ymin, xmax, ymax
	IS_CHANGED	
	LAYERS_LIST (array)	
	LIMITS	- xmin, ymin, xmax, ymax
	NETS_LIST (array)	
	NUM_REPEATS	
	NUM_SR	
	PROF (free text output)	
	PROF_LENGTH	
	PROF_LIMITS	- xmin, ymin, xmax, ymax
	REPEAT (array)	- angle, flip, mirror, step, xa, ya, xmin, ymin, xmax, ymax
	SR (array)	- angle, flip, mirror, step, dx, dy, nx, ny, xa, ya, xmin, ymin, xmax, ymax
	SR_LIMITS	- xmin, ymin, xmax, ymax
symbol	ATTR (array)	- name, val
	EXISTS	
	FEATURES (free text output)	- (options: break_feat, select)
	FILL	- dx, dy
	LIMITS	- xmin, ymin, xmax, ymax
	SYMS_HIST (array)	- arc, line, pad, symbol
	FEAT_HIST	- arc, line, pad, surf, text, total (options: select)
wheel	DCODE (array)	angle, mirror, num, symbol

Appendix A *Common Examples*

Script Example - Feature Selection by Polygon Line

This example demonstrates a script using the Select Feature by Line function. Using the Polygon Area Selector icon you can select all intersecting features by clicking a line then moving over features to be selected, double click M2 at the other end.

In this example, you will supply the coordinates of two lines (using the **area_xy** command) and then perform the selection:

```
COM sel_clear_feat
COM filter_area_strt
COM filter_area_xy,x=<X1>,y=<Y1>
COM filter_area_xy,x=<X2>,y=<Y2>
COM filter_area_end,layer=filter_name=popup,operation=select,
    area_type=polygon,inside_area=yes, intersect_area=yes,
    lines_only=no, min_len=0,max_len=0,min_angle=0,max_ang
```

The **Filter_area_end** parameters: "**inside_area**" & "**intersect_area**" are not used in this mode.

Appendix B *Frequently Asked Questions*

B.1. When we try to add a symbol into an area with reference to a feature in a script we are writing, I need to know the coordinates of the feature. How do I get these coordinates?

After selecting the reference feature, use the Info command to get its coordinates, for example:

```
COM info,args=-t layer -e dic.001.bini/pcb/olt -d FEATURES \
-o select, out_file=/tmp/bini_f, write_mode=replace
```

Returns a line for each selected feature, as in the following feature file:

```
### Layer - olt features data ###
#P 5.55002 6.748 r51 P 0 0 N
#P 14.31002 6.748 r51 P 0 0 N
#P 5.55002 8.098 r51 P 0 0 N
```

Where the numbers after **#P** are the X,Y coordinates of the feature. Coordinates are given for all types of features.

B.2. Using the DO_INFO *command on measurement line output what does the RC, PT, CR stand for?*

```
pth plt 10 x 10 mil r10 RC 8.36685 7.831902 0.01 0.01 1
```

- RC = rectangle [size = width (smaller of the two)]
- PT = point (size = 0)
- CR = circle (size =diameter)
- SG = segment (size = segment length)

B.3. How can I read the results of an analysis process?

One way is to preread the values from the ERF ranges section, and then compare the value that you have in the measurement line with the ERF range:

```
ss2sm sst 0 mil r5 SG 21.967 8.272417 21.967 8.272417 2
```

where we assume $VAL to be the value of the measurement

Compare this with the ranges stored in variables.

```
if ( $VAL < $RANGE1 ) then
  set SEVERITY = R
else
  if ( $VAL < $RANGE2 ) then
     set SEVERITY = Y
    else
     set SEVERITY = G
  endif
endif
```

Appendix C *Error Messages*

To be completed.

Appendix D *System Administration Notes*

Line Mode Commands

These are the line mode commands related to Scripts. See Doc.0206, Line Mode Commands, for complete descriptions.

Configuration Parameters

These are the configuration parameters related to Scripts. See Doc.0203, System Administration, for complete descriptions.

scr_history