

Introduction to SKILL[®] Programming

Version 6.1

Lab Manual

April 30, 2007

© 1990-2007 Cadence Design Systems, Inc. All rights reserved.
Printed in the United States of America.

Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA

Cadence Trademarks

Trademarks and service marks of Cadence Design Systems, Inc. (Cadence) contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address above or call 800.862.4522.

Allegro®	Incisive®	Silicon Express™
Accelerating Mixed Signal Design®	InstallScape™	SKILL®
Assura®	IP Gallery™	SoC Encounter™
BuildGates®	NanoRoute®	SourceLink® online customer support
Cadence® (brand and logo)	NC-Verilog®	Specman®
CeltIC®	NeoCell®	Spectre®
Conformal®	NeoCircuit®	Speed Bridge®
Connections®	OpenBook® online documentation library	UltraSim®
Diva®	OrCAD®	Verifault-XL®
Dracula®	Palladium®	Verification Advisor®
ElectronStorm®	Pearl®	Verilog®
Encounter®	PowerSuite®	Virtuoso®
EU CAD®	PSpice®	VoltageStorm®
Fire & Ice®	SignalStorm®	Xtreme®
First Encounter®	Silicon Design Chain™	
HDL-ICE®	Silicon Ensemble®	

Other Trademarks

Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission.

All other trademarks are the property of their respective holders.

Confidentiality Notice

No part of this publication may be reproduced in whole or in part by any means (including photocopying or storage in an information storage/retrieval system) or transmitted in any form or by any means without prior written permission from Cadence Design Systems, Inc. (Cadence).

Information in this document is subject to change without notice and does not represent a commitment on the part of Cadence. The information contained herein is the proprietary and confidential information of Cadence or its licensors, and is supplied subject to, and may be used only by Cadence's customer in accordance with, a written agreement between Cadence and its customer. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

RESTRICTED RIGHTS LEGEND Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

UNPUBLISHED This document contains unpublished confidential information and is not to be disclosed or used except as authorized by written contract with Cadence. Rights reserved under the copyright laws of the United States.

Table of Contents

Introduction to SKILL Programming

Module 1 About This Course

Lab 1-1	Locating SKILL Functions with the SKILL Finder	1-1
Lab 1-2	Locating SKILL Functions with the Search Assistant.....	1-3
	Starting the Cadence Software.....	1-3
	Using the Search Assistant	1-4
Lab 1-3	Locating SKILL Solutions and Examples	1-6

Module 2 SKILL Programming Fundamentals

Lab 2-1	Using the Command Interpreter Window.....	2-1
	Exploring the CIW Pull-Down Menus	2-1
	Setting the Log File Display Filter Options.....	2-1
	Scrolling the CIW Output Pane	2-2
	Using Your Favorite Text Editor	2-3
	Editing Your <i>.cdsinit</i> File.....	2-3
	Entering SKILL Expressions in the CIW Input Pane	2-4
	Examining the Log File	2-4
	Pasting Previous commands into the CIW Input Pane	2-4
	Modifying Previous Commands in the CIW Input Pane	2-5
Lab 2-2	Exploring SKILL Numeric Data Types.....	2-6
	Entering Numeric Data	2-6
	Using Arithmetic Operators.....	2-6
	Observing Operator Precedence	2-7
	Observing SKILL Evaluation	2-7
	Controlling the Order of Evaluation	2-8
Lab 2-3	Exploring SKILL Variables.....	2-11
	Initializing a Variable	2-11
	Retrieving the Value of a Variable	2-11
	Incrementing a Variable.....	2-11
	Checking the Type of a Variable	2-12
Lab 2-4	Displaying Data in the CIW.....	2-13
	Using the <i>println</i> Function.....	2-13
	Using the <i>printf</i> Function.....	2-13

Lab 2-5	Solving Common Input Errors.....	2-16
	Confirming that the SKILL Evaluator Is Available.....	2-16
	Resolving Unbalanced String Quotes and Parentheses	2-16
	Resolving Problems with Inappropriate White Space	2-17
	Inappropriate Space Again.....	2-18
	Resolving Data Error Messages.....	2-18

Module 3 Lists

Lab 3-1	Creating New Lists	3-1
	Creating a New List Using the ' Operator	3-1
	Controlling the Number of List Items per Line	3-2
	Creating a New List Using the <i>list</i> Function.....	3-3
	Building a List Using the <i>cons</i> Function	3-4
	Building a List Using the <i>append</i> Function	3-5
	Comparing the <i>cons</i> and <i>append</i> Functions.....	3-5
	Exploring Restrictions for the <i>cons</i> and <i>append</i> Functions	3-5
	Adding an Element to the End of a List.....	3-6
Lab 3-2	Extracting Items from Lists	3-10
	Using <i>car</i> and <i>cdr</i> to Extract Items from a List	3-10
	Extracting Coordinates from a Bounding Box	3-12

Module 4 Windows

Lab 4-1	Opening Windows	4-1
	Opening a Design Window.....	4-1
	Opening a Text Window.....	4-2
Lab 4-2	Resizing Windows	4-3
	Retrieving the Bounding Box of a Window	4-3
	Resizing a Window	4-4
Lab 4-3	Storing and Retrieving Bindkeys.....	4-6
	Locating an Available Key	4-6
	Establishing the Bindkey Definitions	4-7
	Testing Schematic and CIW Bindkeys	4-8
	Testing Layout and CIW Bindkeys	4-8
Lab 4-4	Defining a Show File Bindkey.....	4-10
	Defining Your Bindkey	4-10
	Testing Your Bindkey Definition	4-10

Module 5 Database Queries

Lab 5-1	Querying Design Databases.....	5-1
	Opening a Design.....	5-1
	Retrieving the <i>cellView</i> Database Object.....	5-2
	Making Queries.....	5-2
	Querying the Other Designs	5-4

Module 6 Developing a SKILL Function

Lab 6-1	Developing a SKILL Function	6-1
	Requirements	6-1
	Suggestions	6-1
	Setting the <i>writeProtect</i> Switch	6-2
	Examining the SKILL Path.....	6-2
	Editing the Source Code File	6-3
	Writing the Source Code.....	6-3
	Loading Your Function.....	6-4
	Testing Your Solution.....	6-4
	Test Case Results	6-5
	Sample Solution	6-6
	Optional Enhancement for Advanced Students	6-7

Module 7 Flow of Control

Lab 7-1	Writing a Database Report Program	7-1
	Requirements	7-1
	Suggestions	7-2
	Testing Your Solution.....	7-3
Lab 7-2	Exploring Flow of Control.....	7-6
	Requirements	7-6
	Recommendations.....	7-6
	Testing Your Solution.....	7-7
	Sample Solution	7-8
Lab 7-3	More Flow of Control	7-9
	Requirements	7-9
	Testing Your Solution.....	7-10
	Solution.....	7-11

Lab 7-4	Controlling Complex Flow	7-12
	Requirements	7-12
	Suggestions	7-13
	Testing Your Solution.....	7-14
	Sample Solution	7-15

Module 8 List Construction

Lab 8-1	Revising the Layer Shape Report	8-1
	Requirements	8-1
	Testing Your Solution.....	8-1
	Sample Solution	8-2
Lab 8-2	Describing the Shapes in a Design	8-3
	Requirements	8-3
	Recommendations.....	8-3
	Testing Your Solution.....	8-4
	Sample Solution	8-5

Appendix A Menus

Lab A-1	Exploring Menus.....	A-1
	Examining the Code for a Pop-Up Menu	A-1
	Running the Code	A-2
	Examining the Code for a Pull-Down Menu	A-2
	Running the Code	A-3

Appendix B Customization

Lab B-1	Defining Bindkeys in the <i>.cdsinit</i> File	B-1
	Editing Your <i>.cdsinit</i> File.....	B-1
	Testing the <i>.cdsinit</i> File.....	B-3

Appendix C File I/O

Lab C-1	Writing Data to a File	C-1
	Obtaining an Output Port on a File.....	C-1
	Writing Data to the File	C-1
	Closing the File	C-2
	Viewing the File.....	C-3

Lab C-2	Reading Data from a Text File	C-4
	Obtaining an Input Port on a File.....	C-4
	Reading the File a Line at a Time	C-5
	Closing the File	C-6
	Reading the Data From a Text File.....	C-6
	Closing the File	C-7
	Reading Numeric Data from a Text File	C-7
Lab C-3	Writing Output to a File.....	C-9
	Recommendations.....	C-10
	Testing Your Solution.....	C-11
	Solutions	C-13

Labs for Module 1

About This Course

The image displays a composite of three screenshots related to Cadence SKILL programming and documentation.

Top Screenshot: Cadence SourceLink Website

The website header includes the Cadence logo and "SOURCELINK". Navigation links include "SourceLink home", "Cadence home", "SourceLink sitemap", and a search bar. A welcome message "Welcome, Stephen" is visible. A sidebar on the left contains links for "SourceLink home", "My information", "Service requests", "Software update", "Design topics", and "Help & feedback". The main content area is titled "SKILL INFORMATION" and includes a section for "SKILL Library Pages" with links to "Allegro SKILL Code", "Concept SKILL Code", and "Custom IC Design SKILL Code". Below this is a section for "Custom IC Design Solution information for SKILL" displaying results 1 to 5 of 793.

Bottom Left Screenshot: Cadence SKILL API Finder

This window shows a search for "ciw" across all categories. It lists 10 matches found, including "ciwMenuInit", "ciwHiExit", "ciwCreatePulldown", "ciwCreateMenu", and "ciwAddTranslatorItem". The description for "ciwMenuInit" is shown: "Initializes and installs the CIW pulldown menus based on the menu files found in the hierarchy. This function will be called instead of hiInstallCIWMenus. It..."

Bottom Right Screenshot: Virtuoso Layout Suite L Reading: ether AEQ_Dec layout

This window shows a search for "ciw" within the Virtuoso Layout Suite L. It displays 10 results for SKILL Functions, including "ciwAddTranslatorItem", "ciwCreateMenu", "ciwCreatePulldown", "ciwHiExit", "ciwMenuInit", "ciwMenuInstall", "ciwMenuRead", "ciwRemove", "hiFocusTo", and "hiGetCIW". The description for "ciwMenuInit" is highlighted: "Initializes and installs the CIW pulldown menus based on the menu files..."

Lab 1-1 Locating SKILL Functions with the SKILL Finder

Objective: Open and use the SKILL Finder to locate syntax and descriptions of SKILL functions. Use the CDSDoc online documentation system to locate additional details about the functions.

1. In a terminal window enter

```
cdsFinder &
```

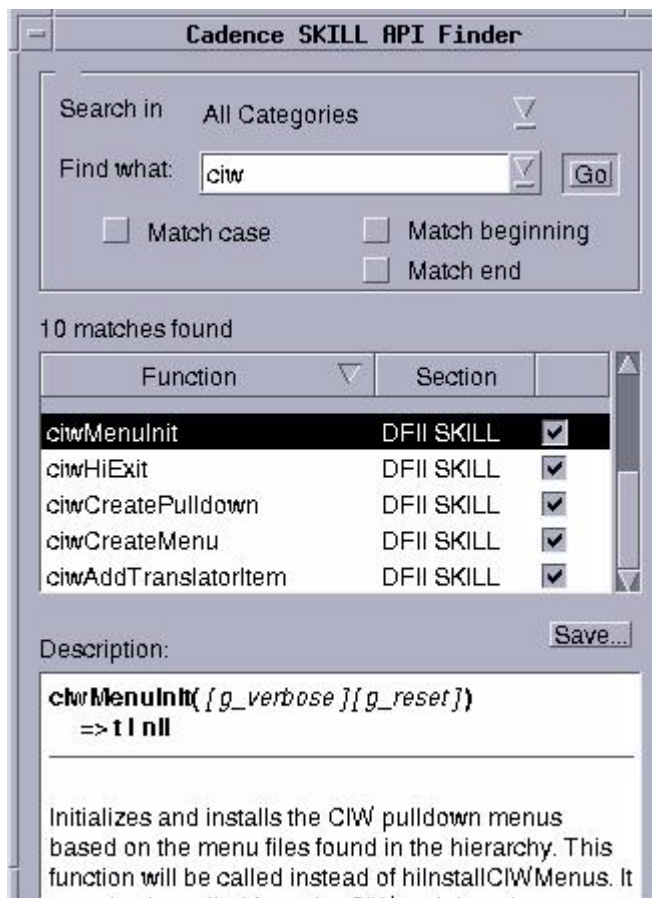
The Cadence SKILL API Finder form appears

2. In the Find What field enter

```
ciw
```

and click **Go**.

The Finder returns a list of matches.



How many matches appear in the window?

How many of the commands start with ciw?

3. Click the **Match beginning** button.

Did the Finder filter out those commands that did not have ciw at the beginning?

4. Select **ciwAddTranslatorItem** and read the syntax and description.

Do you have enough information to use this function?

5. In a terminal window enter

```
cdsDoc &
```

The Cadence documentation opens.

Note: If you are on Linux you may need to open the Mozilla web browser prior to invoking cdsDoc.

6. In the CDSDoc:Library window click **Search**.

A web browser window titled Product Documentation appears.

7. In the Search For field enter

```
ciwAddTranslatorItem
```

8. Click **Go** and view the results.

9. Select the match that is NOT the table of contents.

10. At the top of the new page next to the flashing Search Match item, click on the **Jump to first word that matches search string** link.

11. Read the description for *ciwAddTranslatorItem*.

Is this a little clearer?

cdsDoc gives you a detailed description of the function and its arguments.

Lab 1-2 Locating SKILL Functions with the Search Assistant

Objective: Start the Cadence software, open a session window, and use the SKILL lookup feature of the Search assistant from the session window.

Starting the Cadence Software

1. In a terminal window, enter the command to change to the class lab directory:

```
cd SKILL
```

2. Start the Cadence software in background mode in the same window. Enter

```
virtuoso &
```

When the Virtuoso® Design Environment starts, the Command Interpreter Window (CIW) opens at the bottom of the screen. The CIW is numbered 1 in the lower-left corner. The CIW banner displays the names of a number of pull-down menus.

3. Choose **File—Open** from the CIW menus.

The Open File form appears.

4. In the Open File form, make the following choices:

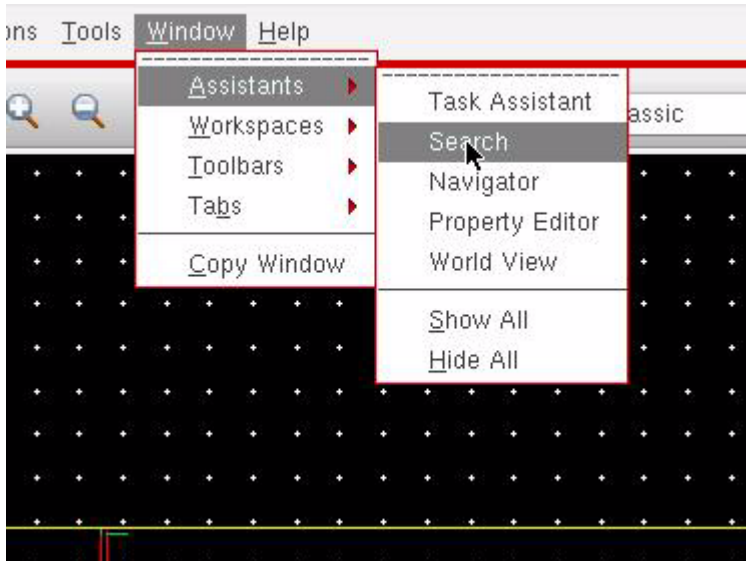
Library	master
Cell	skilltest
View	layout
Open with	Layout L

5. Click **OK** in the Open File form.

The new (blank) cellview opens in a session window.

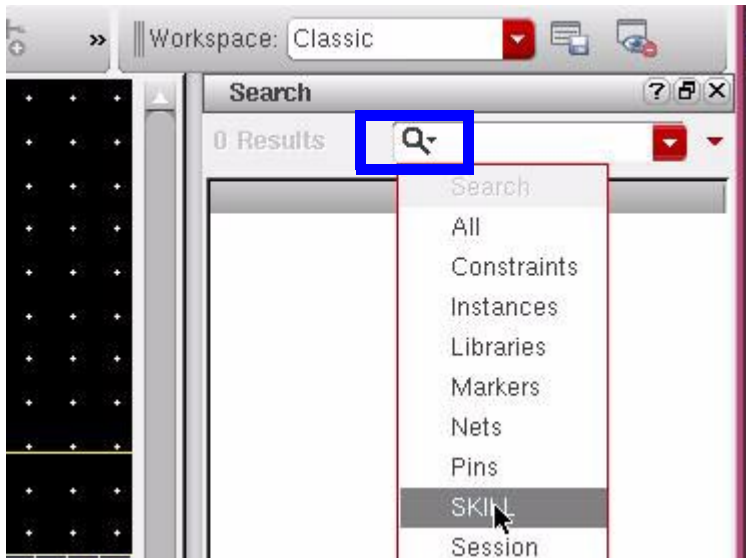
Using the Search Assistant

1. In the session window, select **Window—Assistants—Search**, as shown below.

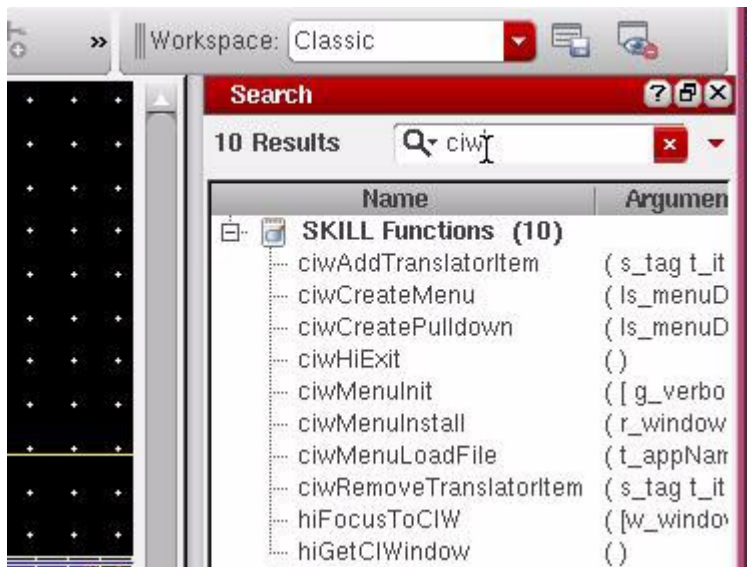


The search pane appears in the session window. A magnifying glass icon indicates the place to enter search criteria.

2. Select **SKILL** from the **Search** filter menu in the search pane.



3. Enter the string *ciw* and press **Return**.



4. Examine the results.
- Stretch the search pane to the left to expand it.
 - Resize the columns using the mouse.
 - Mouse-over a SKILL® function to see a pop-up describing it.
5. Close the cellview and exit the Cadence software.



Lab 1-3 Locating SKILL Solutions and Examples

Objective: Log in to SourceLink online customer support and search for information about a specific SKILL command. Also view the SKILL information link to see problem solutions containing SKILL code as well as documented SKILL code examples.

Important

You can only complete this lab if you have access to the internet and a SourceLink account. If you do not, your instructor might be able to perform a demo of this lab for the class.

1. In a web browser enter

`http://sourcelink.cadence.com`

2. Log in to SourceLink with your user name and password.

Note: You can easily get a SourceLink account by providing Cadence with your license server host ID. If you do not know this, contact a CAD administrator in your company.

3. Makes sure the following boxes have checkmarks in them:

All Products

All Document types

Supported releases

4. In the Search field enter

`ciwAddTranslatorItem`

and click the **Search** button. A window containing the search results opens.

5. Click the match called:

Error when using `ciwAddTranslatorItem()` in IC61

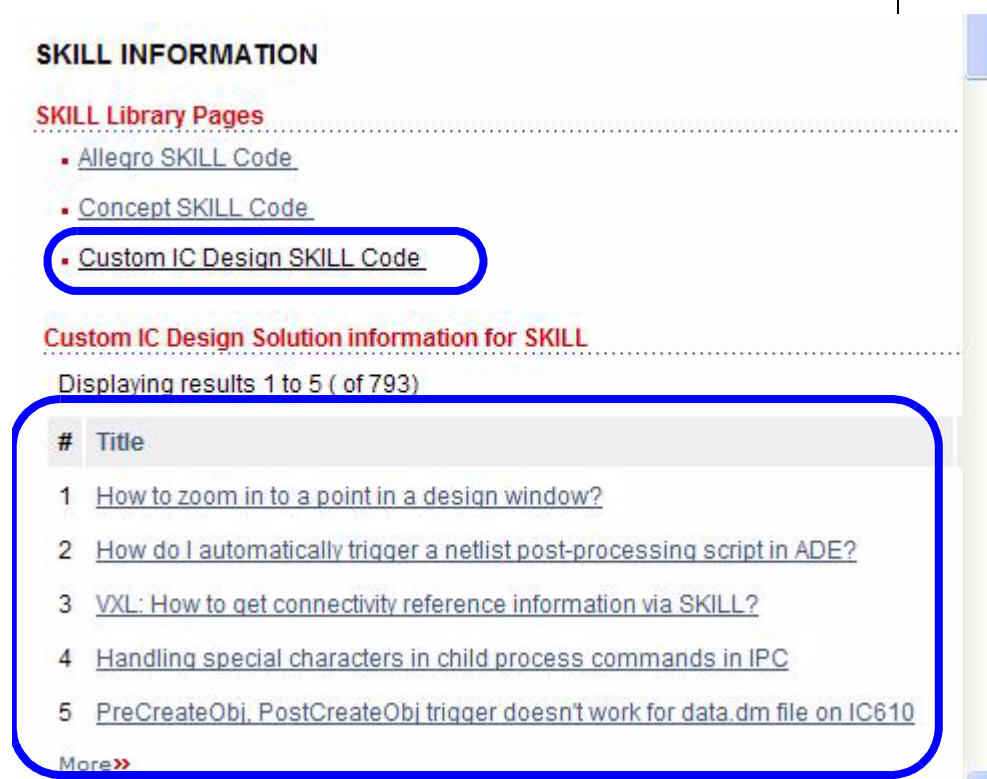
This opens a solution describing the problem, the fix and a workaround. An example of using the command is also provided.

6. Close the solution window.

7. Select **SourceLink home** on the top of the search results page.
8. Under Resource Library on the right side of the page, select **SKILL Information**.

A new window appears titled SKILL INFORMATION. You will see two main areas for Custom IC Design:

- A link to *Custom IC Design SKILL Code*.
- A list of solutions under *Custom IC Design information for SKILL*.



SKILL INFORMATION

SKILL Library Pages

- [Allegro SKILL Code](#)
- [Concept SKILL Code](#)
- [Custom IC Design SKILL Code](#)

Custom IC Design Solution information for SKILL

Displaying results 1 to 5 (of 793)

#	Title
1	How to zoom in to a point in a design window?
2	How do I automatically trigger a netlist post-processing script in ADE?
3	VXL: How to get connectivity reference information via SKILL?
4	Handling special characters in child process commands in IPC
5	PreCreateObj, PostCreateObj trigger doesn't work for data.dm file on IC610

[More >>](#)

9. View the list of solutions on this page. Select several of interest or click **More** to see more solutions.
10. Click the **Custom IC Design SKILL Code** link on the SKILL Information page.

A page with SKILL program examples appears. There are detailed descriptions for each program. Clicking on the program link allows you to view or save the source code.

11. Click on a few items of interest to view the source code.
12. Scroll to the bottom of the Custom IC Design SKILL Code window and note the line:

Get all the files (Compressed tar archive 65673 bytes) (Zipped archive 73477 bytes)

This allows you to download all the examples to your system.

13. You may also submit SKILL code you would like to share using the link provided.



Labs for Module 2

SKILL Programming Fundamentals

Lab 2-1 Using the Command Interpreter Window

Objective: To start the Virtuoso Design Environment and learn about the CIW.

Exploring the CIW Pull-Down Menus

1. In a terminal window, change the working directory:

```
cd SKILL
```

2. Start the Cadence software:

```
virtuoso &
```

The CIW opens.

Here is a summary of the CIW pull-down menus for the *virtuoso* executable.

Menu Name	Most Common Uses
File	Open or create a design. Manage bookmarks
Tool	Launch the Library Manager. Launch the SKILL Development Environment.
Options	Manage bindkeys Manage technology files

Which menu has the User Preferences menu item?

Which menu has the Licenses item?

How do you view a graph of the technology file information?

Setting the Log File Display Filter Options

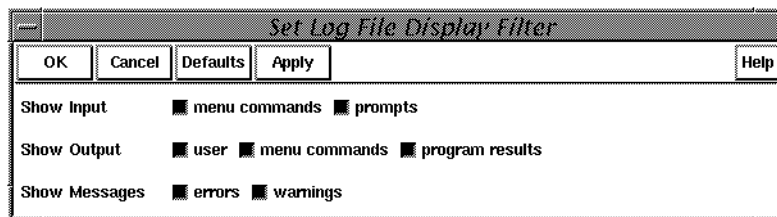
1. Resize the CIW so that you can see at least 10 lines in the output field and 20 lines in the input field of the CIW.
2. Set the CIW so there are 15 buffer lines in the input field.

Tip: See the User Preferences form.

3. In the CIW window banner, select **Options—Log Filter**.

The Set Log File Display Filter form appears.

4. Set the toggles buttons as shown.



5. Click **OK** to execute the command and remove the form from the screen.

The CIW output pane shows

- SKILL® function calls
- SKILL output
- SKILL function results
- Warnings
- Error messages

Scrolling the CIW Output Pane

The CIW has a scroll bar on the right side.

1. Position the mouse pointer over the triangle at the top of the scroll bar and click the left mouse button.
2. With the pointer still over the triangle at the top, press and hold the left mouse button until you scroll all the way back.
3. Move the pointer over the rectangle in the scroll bar. Press the middle mouse button and drag the rectangle to the bottom of the scroll bar.

Using Your Favorite Text Editor

Cadence allows you to use any text editor. You will change the *editor* variable in the *.cdsinit* file. It is set to *gvim* by default for this class.

1. In a terminal window open the *.cdsinit* file in your editor.
2. Locate the *editor =* line and change the statement referencing *gvim* to the name of your editor.

Note: *gvim* is a commonly used editor for SKILL code. You can get more information on *gvim* at <http://www.gvim.org>.

3. Save and close the *.cdsinit* file.
4. In the CIW, enter

```
load ".cdsinit"
```
5. Correct any errors and reload as needed.

Editing Your *.cdsinit* File

You'll use *vi* to edit your *.cdsinit* file so that the next time you enter the environment, the CDS log file display filter will be set as it is now (which is not the default).

1. In the CIW input pane, enter

```
edit( ".cdsinit" )
```

An editor window appears.
2. Find this line:

```
;;; hiSetFilterOptions(t t t t t t t)
```
3. By deleting the three semi-colons, change the line to

```
hiSetFilterOptions(t t t t t t t)
```

4. Save your edits and exit the editor.
You have uncommented a call to the *hiSetFilterOptions* function.

5. To have your changes take effect immediately, enter the following in the CIW:

```
load ".cdsinit"
```

Note: The next time you enter the Cadence environment, the *hiSetFilterOptions* function will be invoked with your new settings.

Entering SKILL Expressions in the CIW Input Pane

The Cadence® SKILL programming language supports the full-range of arithmetic operations.

1. In the CIW input pane, enter

```
6*5
```

The SKILL Evaluator executes *6*5* and displays your input, *6*5*, and the return result, *30*, in the CIW output pane.

Examining the Log File

Use the *view* function to open a text file.

1. In the CIW input pane, enter

```
view( "~/CDS.log" )
```

A Show File window appears displaying the file.

Can you find your input lines in the log file?

Pasting Previous commands into the CIW Input Pane

1. With the left mouse button, click and drag over the line *6*5* in the CIW input pane.
2. Go to a new line at the bottom of the CIW input pane and click the middle mouse button.

The line is pasted into the CIW input pane.

3. Press **Return** to execute the command.

The SKILL Evaluator executes $6*5$ a second time and displays the return result, 30 , in the CIW output pane.

Modifying Previous Commands in the CIW Input Pane

1. Click the left mouse button on the uppermost line $6*5$ in the CIW input pane.
2. Using the cursor or mouse, move to the number 5 and change it to the number 4. So the expression is now $6*4$.
3. Press **Return** to execute the command.

The SKILL Evaluator executes $6*4$ and displays the return result, 24 , in the CIW output pane. You did not have to cut and paste the expression. You were able to edit it inline.

Lab Summary

In this lab, you

- Started the Virtuoso® Design Environment.
- Examined pull-down menus in the CIW.
- Set the Log File Display Filter options interactively.
- Scrolled the CIW output pane.
- Entered SKILL expressions in the CIW input pane.
- Pasted a line from the CIW output pane into the input pane to execute a SKILL expression again.
- Modified a SKILL expression inline in the CIW input pane.



Lab 2-2 Exploring SKILL Numeric Data Types

Objective: Become familiar with fixed-point and floating-point numbers and arithmetic operators.

Entering Numeric Data

The SKILL language provides a variety of means of denoting numeric data, including scaling factors and scientific notation.

1. In the CIW, enter

5
5.3
1e10
5M
3m
2.0n

What is displayed in the CIW output pane?

Using Arithmetic Operators

The SKILL language provides familiar operators for arithmetic computation. When all the operands are integers, the SKILL Evaluator returns an integer result.

1. In the CIW, enter

7/5

What is displayed in the CIW output pane?

2. In the CIW, enter

9/5

What is displayed in the CIW output pane?

If the operands include both integer and floating-point numbers, the SKILL Evaluator returns a floating-point result.

3. In the CIW, enter

$1+0$

What is displayed in the CIW output pane?

4. In the CIW, enter

$1+0.0$

What is displayed in the CIW output pane?

5. In the CIW, enter

$5 + 4.1$

$5 + 4$

$5 + 4.0$

What is displayed in the CIW output pane?

Observing Operator Precedence

SKILL operators have precedence. Can you predict the values of the following expressions?

1. In the CIW, enter

$5+3/2$

Which operator has higher precedence?

2. In the CIW, enter

$5*3/2$

3. In the CIW, enter

$5*3**2$

4. In the CIW, enter

$20-9/3*2**3$

Observing SKILL Evaluation

You can use the SKILL trace facility to verify the order of evaluation.

When the SKILL trace facility is active, the SKILL Evaluator notifies you in the CIW output pane before it evaluates an expression.

Note: The notification includes one or more vertical bars that include the nesting depth of the expression within in the current top-level expression under evaluation.

After evaluating the expression, the SKILL Evaluator notifies you of the return result.

Note: The SKILL Parser translates each operator expression into a corresponding SKILL function call. The trace output displays the SKILL function that implements the operator.

1. In the CIW, enter

```
tracef( t)
```

The trace facility is now active.

2. In the CIW, enter

```
20-9/3*2**3
```

The system displays the following in the CIW output pane.

```
20-9/3*2**3
| (9 / 3)
| quotient --> 3
| (2**3)
| expt --> 8
| (3 * 8)
| times --> 24
| (20 - 24)
| difference --> -4
-4
```

3. In the CIW, enter

```
untrace()
```

The SKILL trace facility is now inactive.

Controlling the Order of Evaluation

You can control the order of evaluation by enclosing expressions in parentheses.

1. In the CIW, enter

$(5+3)/2$

2. In the CIW, enter

$(5*3)**2$

Nesting parentheses redundantly can cause unpredictable results.

3. In the CIW, enter

$((5+3))/2$

The SKILL Evaluator responds with the following error

```
*Error* eval: not a function - (5 + 3)
```

Can you correct this error?

Place parentheses appropriately in the expression

$20-9/3*2**3$

- to make it return

-196

- to make it return

2744

- to make it return

24

⇒ Check your work against the solution on the next page.

Answers

1. In the CIW, enter

$(5+3)/2$

2. In the CIW, enter

$20 - (9/3*2)**3$

3. In the CIW, enter

$(20-9/3*2)**3$

4. In the CIW, enter

$(20-9)/3*2**3$

Lab Summary

In this lab, you

- Explored arithmetic computations with fixed-point and floating-point numbers.
- Used the SKILL trace facility to observe the order of evaluation.
- Used parentheses to control the order of evaluation.



Lab 2-3 Exploring SKILL Variables

Objective: Store and retrieve values in SKILL variables.

Initializing a Variable

Use the = operator to assign a value to a SKILL variable.

1. In the CIW, enter
`lineCount = 0`

Retrieving the Value of a Variable

Use the variable name to retrieve the current value of the variable.

1. In the CIW, enter
`lineCount`

Incrementing a Variable

Use the SKILL pre-increment operator ++ to increment the value of a variable and return the new value.

1. In the CIW, enter
`++lineCount`
`lineCount`

Use the SKILL postincrement operator ++ to increment the value of a variable and to return the old value.

2. In the CIW, enter
`lineCount = 0`
`lineCount++`
`lineCount`
`lineCount++`
`lineCount`

Checking the Type of a Variable

A SKILL variable can store any type of data. Use the *type* function to tell what kind of data is currently in a variable.

1. In the CIW, enter

```
lineCount = 5
```

2. In the CIW, enter

```
type( lineCount )
```

What is displayed in the CIW output pane?

3. In the CIW, enter

```
lineCount = "many lines"
```

4. In the CIW, enter

```
type( lineCount )
```

What is displayed in the CIW output pane?

Lab Summary

In this lab, you manipulated SKILL variables.



Lab 2-4 Displaying Data in the CIW

Objective: To use *printf* and *println* functions.

In this lab, you use the *printf* and *println* to display the values of variables in the CIW output pane.

Store values in *x*, *y*, and *z*.

1. In the CIW, enter

```
x = 4
y = 5.3
z = "layout"
```

Using the *println* Function

The *println* function uses the argument's data type to determine how to display its value.

1. In the CIW, enter

```
println( z )
```

The SKILL Evaluator displays the following in the CIW output pane:

```
"layout"
nil
```

nil is the return result.

Using the *printf* Function

You supply the desired format to the *printf* function.

1. In the CIW, enter

```
printf( "The value of x is %n" x )
```

The SKILL Evaluator displays the following in the CIW output pane:

```
The value of x is 4
t
```

t is the return result.

Use the *printf* Function to Format Your Output

1. Assign values to the *libName*, *cellName*, *viewName*, and *shapeCount* variables and use the *printf* function to display the following output in the CIW.

```
The design Cells inverters schematic has 200
shapes
```

2. Modify your solution to display the following line in the CIW output pane.

```
The design Cells inverters schematic has many
shapes
```

3. Modify your solution to display either a string or a number with the same *printf* format string.

```
The design Cells inverters schematic has 200
shapes.
```

```
The design Cells inverters schematic has
"many" shapes
```

Tip: Use the format character that will work for any type of data.

🔗 Check your work against the solution on the next page.

Answers

1. In the CIW, enter

```
libName = "Cells"
cellName = "inverters"
viewName = "schematic"
shapeCount = 200
printf( "\nThe design %s %s %s has %d shapes"
        libName cellName viewName shapeCount )
```

2. Change the value of *shapeCount* and change the format specification from *%d* to *%s*.

```
libName = "Cells"
cellName = "inverters"
viewName = "schematic"
shapeCount = "many"
printf( "\nThe design %s %s %s has %s shapes"
        libName cellName viewName shapeCount )
```

What happens if you do not change the %d to a %s in the above example? What is the error message?

3. Change the format specification for *shapeCount* to *%L*. Change the value of *shapeCount* to 100. Execute the *printf* function. Then change the value of *shapeCount* to *many* and execute it again. No error message should appear.

```
printf( "\nThe design %s %s %s has %L shapes"
        libName cellName viewName shapeCount )
```

Note: The print statement will work for both integers and strings. *%L* uses the default type of the data.

Lab Summary

In this lab, you used the *println* and *printf* functions to display data in the CIW.



Lab 2-5 Solving Common Input Errors

Objective: To trigger common errors and to solve them.

Confirming that the SKILL Evaluator Is Available

1. In the CIW input pane, enter

1+2

You see the following in the CIW output pane

3

If you see a 3, then the SKILL Evaluator is available. You are ready to perform the next section of this laboratory exercise.

Resolving Unbalanced String Quotes and Parentheses

1. Enter the following into the CIW exactly as shown. Notice there is a missing " after the word *lamb*.

```
strcat( "A SKILL function automates" " tasks)
```

2. Press **Return**.

Nothing happens.

3. In the CIW, enter

1+2

4. Press **Return**.

Nothing happens.

Is the SKILL Evaluator occupied and therefore unable to respond?

Enter several characters to complete the SKILL expression.

5. In the CIW, type

]

Nothing happens.

6. In the CIW, type

"

Nothing happens.

7. In the CIW, type

]

You see the following return value in the CIW:

```
"A SKILL function automates tasks) 1+2 ]"
```

8. Confirm that the SKILL Evaluator is available for your next command. In the CIW input pane, enter

```
1+2
```

You see the following in the CIW output pane:

```
3
```

The CIW is available for your next command.

Resolving Problems with Inappropriate White Space

1. Enter the following into the CIW. Notice there is space after the word *strcat* and the (



```
strcat ("A SKILL function automates" "tasks")
```

2. You see the following error message in the CIW:

```
*Error* eval: illegal function - "A SKILL function automates"
```

3. Use the mouse to select your input line in the CIW input pane and edit it.
4. Remove the offending space character.

5. Press **Return**.

You see the following in the CIW:

```
"A SKILL function automates tasks"
```

Inappropriate Space Again

1. In the CIW, enter

```
exampleLine = strcat ( "A SKILL function automates" " tasks)
```

You see the following error message in the CIW:

```
*Error* eval: unbound variable - strcat
```

2. Edit the line in the input pane.

3. Remove the offending space.

4. Press **Return**.

You see the following in the CIW:

```
"A SKILL function automates tasks"
```

5. Check the value of the *exampleLine* variable. In the CIW, enter

```
exampleLine
```

You see the following in the CIW:

```
"A SKILL function automates tasks"
```

Resolving Data Error Messages

1. In the CIW, enter

```
strcat( "A SKILL function automates" 5)
```

You see the following error message on a single line:

```
*Error* strcat: argument #2 should be either a string or a symbol
(type template = "S") - 5
```

2. Enter the following into the CIW to correct the error:

```
strcat( "A SKILL function automates" " 5" )
```

You see the following in the CIW:

```
"A SKILL function automates 5"
```

Summary

In this lab, you triggered common input errors and resolved them. The errors were due to

- Unbalanced string quotes and unbalanced parentheses.
- Inappropriate space.
- Mismatched data.



Labs for Module 3

Lists

Lab 3-1 Creating New Lists

Objective: To build lists with the `'` operator and the *list*, *cons*, and *append* functions.

Creating a New List Using the `'` Operator

1. In the CIW, enter

```
numbers = ' ( 1 2 3 )  
numbers  
type( numbers )
```

By entering these expressions, you have

- Built a list containing the integers *1*, *2*, and *3*.
 - Stored the list in the variable *numbers*.
 - Fetched the value of *numbers*.
 - Used the *type* function to confirm that *numbers* contains a list.
2. Create a new list containing the numbers *4*, *5*, and *6*. Store the list in the variable *myNumbers*.
- ⇒ Check your work against the solution on the next page.

Solution

- In the CIW, enter

```
myNumbers = ' ( 4 5 6 )  
myNumbers
```

Did you remember to include the ' operator in front of (4 5 6)?

Continue with the lab.

Controlling the Number of List Items per Line

The global variable `_itemsperline` governs how many list items per line the SKILL Evaluator displays for a return result. Examine the current value of the variable.

1. In the CIW, enter

```
_itemsperline
```

2. In the CIW, enter

```
' (1 2 3 4 5 6 7 )
```

The SKILL Evaluator displays the list on two lines.

A variable whose name starts with an underscore character is generally a global variable you should not set. However, it is OK to set this global variable. Set the global variable to 15.

3. In the CIW, enter

```
_itemsperline = 15
```

4. In the CIW, enter

```
' (1 2 3 4 5 6 7 )
```

The SKILL Evaluator displays the list on a single line.

Creating a New List Using the *list* Function

1. In the CIW, enter

```
one = 1
two = 2
three = 3
moreNumbers = list( one two three )
moreNumbers
```

What is the value of moreNumbers?

2. Use the variables *one*, *two*, and *three* to create a list containing *1,2,3,2,1* in that order. Store the list in the variable *evenMoreNumbers*.

🔗 Check your work against the solution on the next page.

Solution

- In the CIW, enter

```
evenMoreNumbers = list( one two three two one )
evenMoreNumbers
```

Did you remember to use the list function instead of the ' operator?

Continue with the lab.

Building a List Using the *cons* Function

1. In the CIW, enter

```
result = nil
result = cons( 1 result )
result = cons( 2 result )
result = cons( 3 result )
```

You have incrementally built up a list in *result*.

2. In the CIW, enter

```
reverse( result )
result
```

You have built a reversed copy of the list in *result*. Yet, as you next verified, the list in *result* is untouched.

What do you need to do so that the variable result contains the reversed list?

⇒ Check your work against the solution on the next page.

Solution

- In the CIW, enter

```
result = reverse( result )
```

Continue with the lab.

Building a List Using the *append* Function

1. In the CIW, enter

```
left = ' ( 1 2 3 )  
right = ' ( 4 5 6 )  
leftRight = append( left right )  
left  
right
```

Can you describe the result?

Comparing the *cons* and *append* Functions

1. In the CIW, enter

```
append( left right )  
append( right left )
```

Can you describe the difference between the two results?

2. In the CIW, enter

```
cons( left right )  
cons( right left )
```

Can you describe the difference between the two results?

Exploring Restrictions for the *cons* and *append* Functions

The second argument to the *cons* function must be a list.

1. In the CIW, enter

```
cons( 7 right )  
cons( right 7 )
```

You see the following error message:

```
*Error* cons: argument #2 should be a list (type template = "gl") - 7
```

Both arguments to the *append* function must be lists.

2. In the CIW, enter

```
append( right 7 )
```

The following error message appears:

```
*Error* append: argument #2 should be a list - 7
```

Adding an Element to the End of a List

If we put 7 into a list, we could use the *append* function to add 7 to the end of the list *right*.

How can we turn 7 into a list?

1. In the CIW, enter

```
list( 7 )  
append( right list( 7 ))  
right  
right = append( right list( 7 ))
```

2. Add 4 to the end of the list *left*.

⇒ Check your work against the solution on the next page.

Solution

- In the CIW, enter

```
left = append( left list( 4 ))
```

Continue with the lab.

Building Hierarchical Lists (Optional)

By using the *list*, *cons* and *append* functions, you use construct lists of arbitrary depth and complexity.

In this section you will construct three lists. When done, check your work against the solutions that follow.

1. In the CIW, enter

```
left = '( 1 2 3 )
right = '( 4 5 6 )
```

2. In the CIW, enter this example:

```
cons( left right )
cons( cons( left right ) right )
```

You see the following results:

```
((1 2 3) 4 5 6)
(((1 2 3) 4 5 6) 4 5 6)
```

3. Build list A with various combinations of the *list* function, the *cons* function, the *append* function and the variables *left* and *right*.

Use the *println* function to display the list on a single line.

List A

```
( 1 2 3 ( 4 5 6 ) 1 2 3 )
```

4. Build list B with various combinations of the *list* function, the *cons* function, the *append* function and the variables *left* and *right*.

Use the *println* function to display the list on a single line.

List B

```
( ( 1 2 3 ( 4 5 6 ) ) 1 2 3 )
```

5. Build list C with various combinations of the *list* function, the *cons* function, the *append* function and the variables *left* and *right*.

Use the *println* function to display the list on a single line.

List C

```
( ( 1 2 3 ) ( 4 5 6 ) ( 1 2 3 ) )
```

⇒ Check your work against the solution on the next page.

Solutions

- To build list A, in the CIW enter

```
println( append( left cons( right left )))
```

- To build list B, in the CIW enter

```
println( cons( append( left list( right))  
left))
```

- To build list C, in the CIW enter

```
println( list( left right left ) )
```

How did you do?

Lab Summary

In this lab, you

- Built lists with the ' operator.
- Built a list with the *list* function.
- Used *cons* to add an element to the front of a list.
- Merged two lists with the *append* function.
- Compared the *cons* and *append* functions.
- Used the *append* function to add an element to the end of a list.
- Built various hierarchical lists with the *list*, *cons* and *append* functions.



Lab 3-2 Extracting Items from Lists

Objective: To extract items from a list using *car*, *cdr* and related commands.

Using *car* and *cdr* to Extract Items from a List

In this section of the lab you create a list, then do three extraction exercises.

To create the list, enter the following in the CIW:

```
numbers = ' ( 1 ( 2 3 ) )  
numbers  
type( numbers )
```

Then perform each of the following. When done, check your work against the solutions on the next page.

1. Extract the number *1* from the list *numbers*.
2. Extract the sublist *(2 3)*.

Note: Two functions are required to perform this step

3. What is the abbreviated function for the two functions from step 3?

Solutions

1. In the CIW, enter

```
car( numbers )
```

The number *1* is returned.

2. In the CIW, enter

```
z = cdr( numbers )
```

```
car( z )
```

The *cdr* returns *((2 3))*, which is a list containing the sublist *(2 3)*.

Why is another car needed after the cdr?

Answer: To get just the sublist you must use *car*. You can also nest the functions as *car(cdr(numbers))*.

3. In the CIW, enter

```
cadr( numbers )
```

Is it the same result as in Answer 2?

Answer: Yes. *car(cdr(numbers))* is equivalent to *cadr(numbers)*.

Continue with the lab.

Extracting Coordinates from a Bounding Box

A bounding box is defined by entering a pair of X,Y coordinates in a list. The coordinates represent the lower left and upper right corners of the box.

In this section you define a bounding box, then do six extraction exercises.

To define the bounding box, enter the following in the CIW:

```
bbox = '( (0 10) (20 30) )  
bbox
```

Then do each of the following. When done, check your work against the solutions on the next page.

1. Extract the lower left X,Y coordinate from *bbox* using *car* and/or *cdr*.
2. Extract the same coordinate pair as in the previous exercise using a more readable function.
3. Extract the upper right X,Y coordinate from *bbox* using *car* and/or *cdr*.
4. Extract the same coordinate pair using a more readable function.
5. Extract the Y coordinate of the upper right X,Y coordinate using *car* and/or *cdr*.
6. Extract the same coordinate using a more readable function.

Solutions

1. To extract the lower left X,Y from *bbox*, in the CIW, enter

```
car( bbox )
```

The list *(0 10)* is returned.

2. Using a more readable function, in the CIW, enter

```
lowerLeft( bbox )
```

The same list *(0 10)* is returned.

3. To extract the upper right X,Y from *bbox*, in the CIW, enter

```
cadr( bbox )
```

The list *(20 30)* is returned.

4. Using a more readable function, in the CIW, enter

```
upperRight( bbox )
```

The same list *(20 30)* is returned.

5. To extract the Y coordinate of the upper right XY pair, in the CIW, enter

```
cadadr( bbox )
```

The value *30* is returned.

6. Using more readable functions, in the CIW, enter

```
yCoord( upperRight( bbox ) )
```

The same value, *30*, is returned.

Lab Summary

In this lab, you

- Built a list and extracted a number and a sublist from it
- Defined a bounding box and extracted six coordinates or coordinate pairs from it



Labs for Module 4

Windows

Lab 4-1 Opening Windows

Objective: To use the *geOpen* and *view* functions to open windows.

Opening a Design Window

1. In the Command Interpreter Window (CIW), enter

```
geOpen( ?lib "master" ?cell "mux2" ?view "layout" )
```

The design appears in a window in edit mode.

The *geOpen* function returns the window ID.

Note the window number for the next step.

2. Look at the window and use its window number in place of ***INSERT THE WINDOW NUMBER*** below. In the CIW, enter

```
designWindow = window( INSERT THE WINDOW NUMBER )
```



Type the window number here.

The *window* function returns the window ID.

The SKILL Evaluator assigns the window ID to the *designWindow* variable. You will refer to *designWindow* in a subsequent lab.

3. Open a window to edit the *master Inv layout* design by entering

```
geOpen ( )
```

Because you haven't specified certain required parameters, the Open File form appears.

4. Supply the library name, cell name, and view name and click on **OK**.
The design appears.

Opening a Text Window

1. In the CIW, enter

```
view( "~/SKILL/.cdsinit" )
```

A window displays the *~/SKILL/.cdsinit* file.

The *view* function returns the window ID of the new window. You cannot use this window to edit the file.

2. Use the *view* function to display your *.cshrc* file.
3. Assign the window ID of the text window to the variable *textWindow*.

You will refer to *textWindow* in a subsequent lab.

Lab Summary

In this lab, you used the

- *geOpen* function to open two designs.
- *view* function to view a text file.



Lab 4-2 Resizing Windows

Objective: To use the *hiGetAbsWindowScreenBBox* and *hiResizeWindow* functions to resize windows.

Retrieving the Bounding Box of a Window

1. In the CIW, enter

```
ciwBBox = hiGetAbsWindowScreenBBox( window(1) t )
```

The *hiGetAbsWindowScreenBBox* function returns the bounding box for window 1. This is the CIW.

2. Retrieve the bounding box of the text window you previously opened. Assign it to the variable *textBBox*.
3. Retrieve the bounding box of the first design window you previously opened. Assign it to the variable *designBBox*.

⇒ Check your work against the solution on the next page.

Answers

1. In the CIW, enter

```
textBBox = hiGetAbsWindowScreenBBox( textWindow t )
```

2. In the CIW, enter

```
designBBox = hiGetAbsWindowScreenBBox( designWindow t )
```

Resizing a Window

1. In the CIW, enter

```
hiResizeWindow( window(1) designBBox )
```

The CIW changes to match the size and position of the design window exactly.

2. Resize the CIW to match the size and position of the text window exactly.

⇒ Check your work against the solution on the next page.

Answers

1. In the CIW, enter

```
hiResizeWindow( window(1) textBBox )
```

Lab Summary

In this lab, you

- Retrieved the bounding box of a window with the *hiGetAbsWindowScreenBBox* function.
- Resized a window with the *hiResizeWindow* function.



Lab 4-3 Storing and Retrieving Bindkeys

Objective: To use the *hiGetBindKey* and *hiSetBindkey* functions to store and retrieve bindkey definitions.

You establish a bindkey definition for both the *Schematics* and *Layout* applications to raise the Command Interpreter Window.

You establish another bindkey definition for the *Command Interpreter* application to raise the current window.

After defining these two bindkeys, you can press a single key to view the CIW or to view the current window.

You can thereby use a large CIW. Conversely, you can hide the CIW until you need it.

Locating an Available Key

1. In the CIW, select the **Options–Bindkey** command to verify that the **F8** key has no bindkey definition for the following applications:

Command Interpreter
Schematics
Layout

2. Use the *hiGetBindKey* function to verify that the **F8** key has no bindkey definition for the following applications:

Command Interpreter
Schematics
Layout

⇒ Check your work against the solution on the next page.

Answers

To use the *hiGetBindKey* function to verify that the **F8** key has no bindkey definition, in the CIW enter

```
hiGetBindKey( "Command Interpreter" "<Key>F8" )
hiGetBindKey( "Schematics" "<Key>F8" )
hiGetBindKey( "Layout" "<Key>F8" )
```

The *Command Interpreter* and *Schematics* environments function call returns *nil*.

The *Layout* environment has a bindkey defined called *leToggleGuidedPathCreate()*. As you continue with this lab you will overwrite this function. Do not worry about it.

Continue with the lab.

Establishing the Bindkey Definitions

1. Use the *hiSetBindKey* function to establish the following bindkey definition for the **F8** key in the Command Interpreter application.

```
"hiRaiseWindow( hiGetCurrentWindow( ) )"
```

2. Use the *hiSetBindKey* function to establish the following bindkey definition for the **F8** key in the Schematics and Layout applications.

```
"hiRaiseWindow( window( 1 ) )"
```

⇒ Check your work against the solution on the next page.

Answers

To establish the Command Interpreter bindkey definition.

```
hiSetBindKey( "Command Interpreter"
  "<Key>F8" "hiRaiseWindow( hiGetCurrentWindow() )" )
```

To establish the Schematics bindkey definition for the **F8** key.

```
hiSetBindKey( "Schematics"
  "<Key>F8" "hiRaiseWindow( window( 1 )" )
```

To establish the Layout bindkey definition for the **F8** key.

```
hiSetBindKey( "Layout"
  "<Key>F8" "hiRaiseWindow( window( 1 )" )
```

Continue with the lab.

Testing Schematic and CIW Bindkeys

1. Enlarge the CIW to occupy the lower half of the screen.
2. Open the *master mux2 schematic* cellview. Move the design window so that it partially overlaps the CIW.
3. Move the cursor over the Schematics application window.
4. Press the **F8** key.

The CIW comes to the front of the screen. Click on the header to make it active.

5. Press the **F8** key with the mouse over the CIW.

The schematic application window comes to the front of the screen.

Testing Layout and CIW Bindkeys

Open the *master mux2 layout* cellview and perform similar tests.

When you are satisfied that your bindkey definitions work, go on to the Lab Summary.

Lab Summary

In this lab, you used the *hiSetBindkey* function to define the following bindkeys:

- A bindkey that raises the CIW for the Schematics and Layout applications.
- A bindkey that raises the current window for the Command Interpreter.

What productivity enhancements can you implement with bindkeys?



Lab 4-4 Defining a Show File Bindkey

Objective: Clear the text selection with a bindkey.

In this lab, you establish a bindkey on *<Key>F8* for *Show File* windows to clear the text selection.

Defining Your Bindkey

Use this expression in your bindkey definition to clear the text selection in the current window.

```
hiUnselectTextAll(  
  hiGetCurrentWindow())
```

Testing Your Bindkey Definition

1. Open a *Show File* application window. In the CIW, enter

```
view( "~/SKILL/.cdsinit" )
```

Use this window to test your bindkey.

⇒ Check your work against the solution on the next page.

Answers

1. In the CIW, enter

```
hiSetBindKey( "Show File" "<Key>F8"  
             "hiUnselectTextAll( hiGetCurrentWindow())"  
             )
```

Lab Summary

In this lab, you used the *hiSetBindKey* function to define a bindkey for the Show File application.



Labs for Module 5

Database Queries

Lab 5-1 Querying Design Databases

Objective: Use the ~> operator to query several design databases.

In this lab, you open several designs and make the following queries using the ~> operator.

- How many nets are in the design?
- What are the net names?
- How many instances are in the design?
- What are the instance names?
- What are the master cell names?
- How many shapes in the design?
- What kinds of shapes are in the design?

Opening a Design

1. Open a design file by selecting, **File–Open**.

The Open File form appears.

2. In the form, enter the values in this table.

Library	master
Cell Name	mux2
View Name	layout
Mode	read

3. Click **OK**.

A Layout application window appears as the current window.

Retrieving the *cellView* Database Object

To query the design, you need to retrieve the *cellView* database object and store it in a variable. The *geGetWindowCellView* function works on the current window by default.

1. In the CIW, enter

```
cv = geGetWindowCellView()
```

The system displays a value resembling *db:23480364* in the CIW.

You are now ready to make several queries regarding the design.

Making Queries

How many nets are in the design?

1. In the CIW, enter

```
length( cv~>nets )
```

The system displays 6 in the CIW. There are 6 nets in this design.

What are the net names?

2. In the CIW, enter

```
cv~>nets~>name
```

The system displays the following list in the CIW:

```
( "SEL" "B" "A" "gnd!" "Y"  
  "vdd!"  
 )
```

How many instances are there in the design?

3. In the CIW, enter

```
length( cv~>instances )
```

The system displays 17 in the CIW.

There are 17 instances in this design.

What are the instance names?

4. In the CIW, enter

```
cv~>instances~>name
```

The system displays the following list in the CIW:

```
("I35" "I31" "I30" "I32" "I36"
 "I4" "I5" "I34" "I41" "I40"
 "I37" "I38" "I6" "I3" "I18"
 "I1" "I2")
```

These are the instances names.

What are the master cell names?

5. In the CIW, enter

```
cv~>instances~>cellName
```

The system displays the following list in the CIW:

```
("M2_M1" "M2_M1" "M2_M1" "M2_M1" "M2_M1"
 "M2_M1" "M2_M1" "M1_POLY1" "M1_POLY1" "M1_POLY1"
 "M1_POLY1" "M1_POLY1" "M1_POLY1" "Inv" "nand2"
 "nand2" "nand2"
 )
```

These are the master cell names.

How many shapes are in the design?

6. In the CIW, enter

```
length( cv~>shapes )
```

The system displays 21 in the CIW.

There are 21 shapes in this design.

What kinds of shapes are in the design?

7. In the CIW, enter

```
cv~>shapes~>objType
```

The system displays the following list in the CIW:

```
("textDisplay" "textDisplay" "textDisplay" "textDisplay" "textDisplay"
 "textDisplay" "rect" "rect" "rect" "rect" "path"
 "path" "rect" "path" "path" "rect"
 "path" "path" "rect" "path" "rect"
 "rect"
)
```

These shape types describe the kind of shapes in the design.

Continue with the lab on the next page.

Querying the Other Designs

Query the following designs:

```
master mux2 schematic
master mux2 extracted
master mux2 symbol
```

In these queries, find out the following information:

- How many nets are in the design?
- What are the net names?
- How many instances are in the design?
- What are the instance names?
- What are the master cell names?
- How many shapes in the design?
- What kinds of shapes are in the design?

What queries are the most relevant to your projects at work?

Can you express your queries using the ~> operator and SKILL[®] functions, such as the length function?

Lab Summary

In this lab, you queried several designs using `~>` expressions. Each expression retrieved one or more database object attributes.



Labs for Module 6

Developing a SKILL Function

Lab 6-1 Developing a SKILL Function

Objective: To develop a SKILL function that computes the area of a bounding box.

Requirements

Develop a SKILL[®] function named *TrBBoxArea* that satisfies the following requirements:

- Make the *TrBBoxArea* function take a single argument named *bBox*.
- Make the *TrBBoxArea* function print a message in the CIW to identify the bounding box and the area.
- Make the *TrBBoxArea* function return the area of *bBox*.

For example, this function call shown here:

```
TrBBoxArea( list( 100:100 250:390 ))
```

displays the following message in the CIW and returns *43500*.

```
Box ((100 100) (250 390)) area is: 43500
```

Suggestions

Base your work on the *TrBBoxHeight* function discussed in the lecture. The source code appears here:

```
procedure( TrBBoxHeight( )
  let( ( ll ur lly ury )
    ll = lowerLeft( bBox )
    ur = upperRight( bBox )
    lly = yCoord( ll )
    ury = yCoord( ur )
    ury - lly
  ) ; let
) ; procedure
```

Make sure that you do the following:

- Write your version of the *TrBBoxArea* function in a new file called *~/SKILL/Functions/TrBBoxArea.il*.
- Use the variables *ll*, *ur*, *llx*, *lly*, *urx*, *ury*, and *area*.
- Compute the area of the bounding box with this code.

```
area = ( urx - llx ) * ( ury - lly )
```

- Use the *printf* function to display this message.

```
printf( "Box %L area is: %n" bBox area )
```

Note that you must use the *%n* specification because *area* might be a floating point number.

Setting the *writeProtect* Switch

It is important for you to be able to redefine the *TrBBoxArea* function during development.

1. Check the value of the *writeProtect* switch. In the CIW, enter

```
status( writeProtect )
```

Is it *nil*?

2. If the value of *writeProtect* is not *nil*, then set it to *nil*. In the CIW, enter

```
sstatus( writeProtect nil )
```

You can redefine any SKILL function you define while *writeProtect* is *nil*.

Examining the SKILL Path

Check to see that the *~/SKILL/Functions* directory is in the SKILL path.

1. In the CIW, enter

```
getSkillPath()
```

The system displays the current SKILL path.

Why is it important that the ~/SKILL/Functions directory be in the SKILL path?

Editing the Source Code File

1. Check the *editor* variable. In the CIW, enter

```
editor
```

Is this the editor you want?

2. If the editor variable indicates the wrong editor for you, change the value of the *editor* variable. Assuming you prefer the *textedit* editor, in the CIW, enter

```
editor = "textedit"
```

3. Edit the new source code file. In the CIW, enter

```
edit( "~/SKILL/Functions/TrBBoxArea.il" )
```

Writing the Source Code

1. Enter the following skeletal definition in the file. You can omit the comments.

```
procedure( TrBBoxArea( bBox )  
    ;;; compute the area of bBox  
    ;;; print the messag  
    printf( "Box %L area is: %n" bBox area )  
    area      ;;; the return result  
    ) ; procedure
```

2. Add source code statements to compute *ll*, *ur*, *llx*, *lly*, *urx*, *ury* and finally *area*.
3. Save the file and exit the editor.

Loading Your Function

In the CIW, enter

```
load( "TrBBoxArea.il" )
```

A *t* is displayed in the CIW output pane.

*What does the *t* indicate?*

Testing Your Solution

Try the following test cases. The bounding boxes have been chosen so that you can compute their area easily yourself.

1. In the CIW, enter

```
aSquare = list( 100:100 200:200 )  
TrBBoxArea( aSquare )
```

2. In the CIW, enter

```
aHorizontalLine = list( 100:100 250:100 )  
TrBBoxArea( aHorizontalLine )
```

3. In the CIW, enter

```
aVerticalLine = list( 100:100 100:390 )  
TrBBoxArea( aVerticalLine )
```

⇒ Check your work against the solution on the next page.

Test Case Results

■ The square

```
Box ((100 100) (200 200)) area is: 10000
10000
```

■ The horizontal line

```
Box ((100 100) (250 100)) area is: 0
0
```

■ The vertical line

```
Box ((100 100) (100 390)) area is: 0
0
```

How did you do?

⇒ Check your work against the solution on the next page.

Sample Solution

The sample solution in the `~/SKILL/Functions/Solutions/TrBBoxArea.il` file is shown here for your convenience.

```
procedure( TrBBoxArea( bBox )
  ll    = lowerLeft( bBox )
  ur    = upperRight( bBox )
  lly   = yCoord( ll )
  ury   = yCoord( ur )
  llx   = xCoord( ll )
  urx   = xCoord( ur )
  area  = ( urx - llx )*( ury - lly )
  printf( "Box %L area is: %n" bBox area )
  area
) ; procedure
```

How did you do?

How would you define local variables in the above solution?

To run the sample solution, enter the following line in the CIW:

```
load( "Solutions/TrBBoxArea.il" )
```

Continue with the lab on the next page.

Optional Enhancement for Advanced Students

You compute the area of a bounding box that the user enters with the mouse. You define a *Layout* bindkey that prompts the user for a bounding box and calls the *TrBBoxArea* function to compute its area.

The *enterBox* function prompts the user to enter a box.

- The *?prompts* argument determines the prompts to show to the user.
- The *?doneProc* argument is the name of the function that the *enterBox* function calls when the user has either completed the box or aborted the operation.

1. In the CIW, enter

```
procedure( TrEnterBoxCB( wid completed? box )
  when( completed? TrBBoxArea( box ))
  ) ; procedure

procedure( TrEnterBox()
  enterBox(
    ?prompts ' ( "First Corner" "Second Corner" )
    ?doneProc "TrEnterBoxCB"
  )
  ) ; procedure

hiSetBindKey( "Layout" "<Key>F9" "TrEnterBox()" )
```

2. In the CIW, use the **File–Open** command to open *master mux2 layout*.

3. Try the bindkey several times.

Verify that your output includes items similar to the following in the CIW:

```
Box ((-0.500000 0.250000) (1.250000 1.000000)) area is: 1.312500
1.312500

Box ((-0.500000 0.500000) (0.0 1.500000)) area is: 0.500000
0.500000
```

Lab Summary

In this lab, you wrote a function that computes the area of a bounding box.



Labs for Module 7

Flow of Control

Lab 7-1 Writing a Database Report Program

Objective: To develop a SKILL® function that counts the shapes used in a design.

In this lab, you write a SKILL® function that counts the shapes used in a design. During the course, you will make several enhancements to this program.

Requirements

Make *TrShapeReport* accept a single parameter named *wid*. The parameter is the window ID of the window displaying the design.

Make the *TrShapeReport* function display a report in the CIW output pane that includes the following items:

- *libName*, *cellName* and *viewName* of the design
- *rectangle* count
- *polygon* count
- *path* count
- count of all other shapes

Here is an example report.

```
master nand2 layout contains:
Rectangles 12
Polygons   0
Paths      3
Misc       0
```

Continue with the lab on the next page.

Suggestions

Maintain the source code in the `~/SKILL/FlowOfControl/TrShapeReport.il` file.

Use the following variables, functions, and operators:

- The `geGetWindowCellView` function to retrieve the `cellView` database object from the window ID.
- The variable `cv` to store the `cellView` database object.
- The variables `rectCount`, `polygonCount`, `pathCount`, and `miscCount` to store the counts.
- The `printf` function to produce the lines of the report.
- The `~>` operator to retrieve the `libName`, `cellName`, and `viewName` attributes from `cv` to produce the first line of the report.
- A `foreach` loop to process the shapes in the design.
- The `~>` operator to retrieve the `shapes` attribute.
- The `case` function to process each shapes's `objType` attribute.
- The `++` postincrement operator to increment the variables.

Continue with the lab on the next page.

Testing Your Solution

First Test

1. In the CIW, enter

```
nand2wid = geOpen( ?lib "master" ?cell "nand2" ?view "layout" ?mode "r" )
```

2. In the CIW, enter

```
TrShapeReport( nand2wid )
```

The CIW displays the following:

```
master nand2 layout contains:
Rectangles 10
Polygons   0
Paths      3
Misc       0
```

Second Test

1. In the CIW, enter

```
mux2wid = geOpen( ?lib "master" ?cell "mux2" ?view "layout" ?mode "r" )
```

2. In the CIW, enter

```
TrShapeReport( mux2wid )
```

The CIW displays the following:

```
master mux2 layout contains:
Rectangles 8
Polygons   0
Paths      7
Misc       6
```

⇒ Check your work against the solution on the next page.

Sample Solution

The following solution does not use local variables. You can enhance this solution to make *rectCount*, *polygonCount*, *pathCount*, and *miscCount* local.

```
procedure( TrShapeReport( wid )
  rectCount = polygonCount = pathCount = miscCount = 0
  cv = geGetWindowCellView( wid )
  printf(
    "%s %s %s contains:"
    cv~>libName cv~>cellName cv~>viewName )
  foreach( shape cv~>shapes
    case( shape~>objType
      ( "rect" rectCount++ )
      ( "polygon" polygonCount++ )
      ( "path" pathCount++ )
      ( t miscCount++ )
    ) ; case
  ) ; foreach
  printf( "\n%-10s %-10d" "Rectangles" rectCount )
  printf( "\n%-10s %-10d" "Polygons" polygonCount )
  printf( "\n%-10s %-10d" "Paths" pathCount )
  printf( "\n%-10s %-10d" "Misc" miscCount )
) ; procedure
```

Did you remember to initialize your numeric variables?

The *~/SKILL/FlowOfControl/Solutions/TrShapeReport.il* file contains the solution.

Lab Summary

In this lab, you developed a SKILL function using the following:

- The *geGetWindowCellView* function
- *~>* operator with various attributes
- The *printf* function
- A *foreach* loop
- The *case* function

You use the *load*, *edit*, and *view* functions to develop your code and to test it.



Lab 7-2 Exploring Flow of Control

Objective: To validate data.

The flow of control in a program that validates data can be complex. In this lab, you write a SKILL function to validate a data item according to the following definition.

Term	Definition
<i>point</i>	A data item is a <i>point</i> only if it is a list of two elements, each of which is either an integer or a floating-point number.

You use the *TrValidatePoint* function in a subsequent lab.

Requirements

Make the *TrValidatePoint* function do the following:

- Accept one argument, *exp*.
- Return *nil*, if *exp* is not a list of two numeric arguments.
- Return *exp*, if *exp* is a list of two numeric arguments.

Recommendations

- Use either the *cond* function or nested *if-then-else* expressions.
- Use the following predicate functions to test the data type.

Predicate Function	Data Type Test
<i>numberp</i>	returns <i>t/nil</i> , depending on whether its arguments is numeric.
<i>listp</i>	return <i>t/nil</i> , depending on whether its arguments is a list.

Testing Your Solution

Use the following test cases in addition to other test cases of your own choice.

Function Call	Expected Result
<i>TrValidatePoint('(5 6))</i>	<i>(5 6)</i>
<i>TrValidatePoint(list(5:0))</i>	<i>nil</i>
<i>TrValidatePoint(list(5 "abc"))</i>	<i>nil</i>
<i>TrValidatePoint(list(5 7 8))</i>	<i>nil</i>

⇒ Check your work against the solution on the next page.

Sample Solution

The `~/SKILL/FlowOfControl/Solutions/TrMakeBBox.il` file contains the solution.

```
;;; TrValidatePoint
;;; returns t/nil whether or not exp is a valid point
;;; test: non-nil list of two numeric arguments

procedure( TrValidatePoint( exp )
  cond(
    ( !exp      nil )
    ( !listp( exp ) nil )
    ( length( exp ) != 2  nil )
    ( !numberp( xCoord( exp ) )  nil )
    ( !numberp( yCoord( exp ) ) nil )
    ( t      exp )
  ) ; cond
) ; procedure
```

Did you use the following functions:

- The *listp* and *numberp* functions?
- The *length* function?
- The *cond* function or nested *if-then-else* expressions?

Lab Summary

In this lab, you wrote a function to validate a point.



Lab 7-3 More Flow of Control

Objective: To compare 2-dimensional points.

In this lab, you write a SKILL function that tests whether one point is lower and to the left of a second point, according to the following definition.

Term	Definition
lower-left	A point P is lower-left of another point Q only if (1) $xCoord(P) \leq xCoord(Q)$ (2) $yCoord(P) \leq yCoord(Q)$

Requirements

Make the *TrLLp* function do the following:

- Accept two arguments *pt1* and *pt2*.
- Return *t* if *pt1* is lower-left of *pt2*.
- Return *nil* otherwise.

Assume that *pt1* and *pt2* are valid points.

Use the following:

- The *xCoord* and *yCoord* functions to retrieve the coordinates of *pt1* and *pt2* as needed.
- The \leq operator to compare numeric values.
- The $\&\&$ operator to combine the results of numeric comparisons.

Continue with the lab on the next page.

Testing Your Solution

Use the following test cases in addition to other test cases of your own choice.

pt1	pt2	result
0:0	10:10	<i>t</i>
10:0	-1:-2	<i>nil</i>
-1:-2	10:0	<i>t</i>

Did you need to explicitly control the precedence of the \leq and $\&\&$ operators?

⇒ Check your work against the solution on the next page.

Solution

The `~/SKILL/FlowOfControl/Solutions/TrMakeBBox.il` file contains the solution.

```
;;; TrLLp
;;; returns t/nil indicating pt1 is to the lower-left of pt2

procedure( TrLLp( pt1 pt2 )
  xCoord( pt1 ) <= xCoord( pt2 ) && yCoord( pt1 ) <= yCoord( pt2 )
) ; procedure
```

Lab Summary

In this lab, you wrote a SKILL expression to test whether one point was lower and to the left of another point. You used the

- `<=` and `&&` operators.
- `xCoord` and `yCoord` functions.



Lab 7-4 Controlling Complex Flow

Objective: To build a bounding box.

In this lab, you write two SKILL functions to build a bounding box according to the following definition.

Term	Definition
bounding box	A list of two points. The first point in the list is below and to the left of the second point.

Requirements

Make the *TriMakeBBox* function do these tasks:

- Accept two valid points as arguments *pt1* and *pt2*.
- Return a bounding box derived from the two valid points.

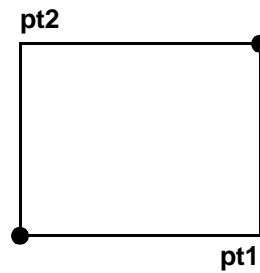
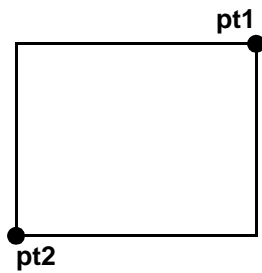
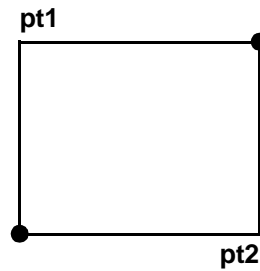
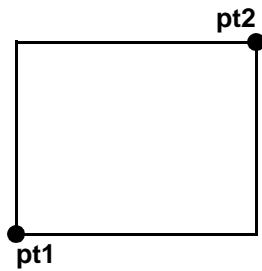
Note: An "i" occurring in a function's prefix usually indicates that the function is an internal or private function. The SKILL language does not allow you to hide internal or private functions so you need to rely on naming conventions. In this case, *TriMakeBBox* is an internal function because it does not do any argument validity checking.

Make the *TrMakeBBox* function do these tasks:

- Accept one argument, *exp*.
- Use your *TrValidatePoint* function to return *nil* if *exp* is not a non-*nil* list of two valid points.
- If *exp* is a list of two valid points, use your *TriMakeBBox* function to return the bounding box.

Suggestions

Notice that *pt1* and *pt2* can occupy the following corners of the bounding box. In each of the four cases, you need to compute the coordinates of the indicated lower-left and upper-right points. Consider using the *min* and *max* functions to treat the four cases together.



Testing Your Solution

Test your code on these test cases.

Function Call	Expected Result
<i>TrMakeBBox('(5 6))</i>	<i>nil</i>
<i>TrMakeBBox(list(5:0 6))</i>	<i>nil</i>
<i>TrMakeBBox(list(6 5:0))</i>	<i>nil</i>
<i>TrMakeBBox(list(list("z") 5:6))</i>	<i>nil</i>
<i>TrMakeBBox(list(0:0 5:6))</i>	<i>((0 0) (5 6))</i>
<i>TrMakeBBox(list(5:6 0:0))</i>	<i>((0 0) (5 6))</i>
<i>TrMakeBBox(list(-5:-6 0:0))</i>	<i>((-5 -6) (0 0))</i>
<i>TrMakeBBox(list(0:6 5:0))</i>	<i>((0 0) (5 6))</i>
<i>TrMakeBBox(list(5:0 0:6))</i>	<i>((0 0) (5 6))</i>

⇒ Check your work against the solution on the next page.

Sample Solution

The `~/SKILL/FlowOfControl/Solutions/TrMakeBBox.il` file contains the full solution for the three labs.

```

procedure( TriMakeBBox( pt1 pt2 )
  let( ( llx lly urx ury pt1x pt2x pt1y pt2y )
    pt1x = xCoord( pt1 )
    pt2x = xCoord( pt2 )
    pt1y = yCoord( pt1 )
    pt2y = yCoord( pt2 )
    llx  = min( pt1x pt2x )
    lly  = min( pt1y pt2y )
    urx  = max( pt1x pt2x )
    ury  = max( pt1y pt2y )
    list( llx:lly urx:ury )
  ) ; let
) ; procedure

procedure( TrMakeBBox( exp )
  let( ( p r )
    cond(
      ( !exp nil )
      ( !listp( exp ) nil )
      ( length( exp ) != 2 nil )
      ( !TrValidatePoint( p = car( exp ) ) nil )
      ( !TrValidatePoint( r = cadr( exp ) ) nil )
      ( t TriMakeBBox( p r ) )
    ) ; cond
  ) ; let
) ; procedure

```

Did you use the cond function or nested if-then-else expressions?

Lab Summary

In this lab, you wrote two SKILL functions to build a bounding box.



Labs for Module 8

List Construction

Lab 8-1 Revising the Layer Shape Report

Objective: Use the *setof* and *length* functions to count shapes in a design.

In this lab, you use the *setof* and *length* functions to revise the *TrShapeReport* function that you developed earlier.

If you prefer, use the sample solution:

```
~/FlowOfControl/Solutions/TrShapeReport.il
```

Requirements

- Maintain the source code for your revised *TrShapeReport* function in the *~/SKILL/ListConstruction/TrShapeReport.il* file.
- To count the number of rectangles, use the following lines of code:

```
rectCount = length(  
    setof( shape cv~>shapes  
        shape~>objType == "rect" )  
)
```
- To compute *miscCount*, sum the other counts and subtract them from the number of shapes in the design.

Testing Your Solution

1. Using the same designs as in Lab 7-1, run the old version of the *TrShapeReport* function.

The CIW displays the report.

2. Redefine the *TrShapeReport* function and run against the same designs.

The CIW displays the report.

Are the two reports identical?

⇒ Check your work against the solution on the next page.

Sample Solution

The *./ListConstruction/Solutions/TrShapeReport.il* file contains the solution shown here.

```
procedure( TrShapeReport( wid )
  let( ( rectCount polygonCount pathCount miscCount cv )
    cv = geGetWindowCellView( wid )
    printf(
      "%s %s %s contains:"
      cv~>libName cv~>cellName cv~>viewName )

    rectCount    = length(
      setof( shape cv~>shapes shape~>objType == "rect" ))
    polygonCount = length(
      setof( shape cv~>shapes shape~>objType == "polygon" ))
    pathCount    = length(
      setof( shape cv~>shapes shape~>objType == "path" ))

    miscCount =
      length( cv~>shapes ) -
      ( rectCount + polygonCount + pathCount )

    printf( "\n%-10s %-10d" "Rectangles" rectCount )
    printf( "\n%-10s %-10d" "Polygons" polygonCount )
    printf( "\n%-10s %-10d" "Paths" pathCount )
    printf( "\n%-10s %-10d" "Misc" miscCount )
  ) ; let
) ; procedure
```

Lab Summary

In this lab, you used the *length* and *setof* functions in a new version of the *TrShapeReport* function.

Which version do you prefer? Why?



Lab 8-2 Describing the Shapes in a Design

Objective: Use the *foreach mapcar* function to build a list of shape descriptions.

In this lab, you write a function that returns a list of descriptions of the shapes in a design.

For the purposes of this lab, a shape description is a three element list that identifies the object type, the layer name, and the layer purpose of a shape as in this example:

```
( "rect" "metal1" "drawing" )
```

Requirements

Make the *TrShapeList* function do the following tasks:

- Accept the window ID of the design window. Name the argument *wid*.
- Return a list of shape descriptions for all the shapes in a design.

Recommendations

Use the following:

- *geGetWindowCellView* function to determine the *cellView* database object for the design.
- *~>* operator to retrieve the *shapes* attribute of this database object.
- *foreach mapcar* function to traverse the list of shapes in the design.

For each shape, do the following:

- Retrieve the *lpp* attribute of the shape.
- Use the *cons* function to add the *objType* attribute onto the front of the *lpp* list.

Testing Your Solution

Test your function on the same designs you used in previous labs.

⇒ Check your work against the solution on the next page.

Sample Solution

```
procedure( TrShapeList( wid )
  let( ( cv )
    cv = geGetWindowCellView( wid )
    foreach( mapcar shape cv~>shapes
      cons( shape~>objType shape~>lpp )
    ) ; foreach
  ) ; let
) ; procedure
```

The *./ListConstruction/Solutions/TrShapeList.il* file contains the solution.

Lab Summary

In this lab, you used the *foreach mapcar* function to process the list of shapes in a design.



Labs for Appendix A

Menus

Lab A-1 Exploring Menus

Objective: Correlate SKILL source code with pop-up menus and pull-down menus.

In this lab, you examine SKILL® source code to determine what the code does. You validate your understanding by loading the source code and interacting with menus.

Examining the Code for a Pop-Up Menu

Use the *view* command to examine the file *~/SKILL/Menus/opiPopUp.il*.

1. In the CIW, enter

```
view( "~/SKILL/Menus/PopUp.il" )
```

Can you describe the pop-up menu that this code builds?

When is the pop-up menu displayed?

⇒ Check your work against the solution on the next page.

Answer

The source code does the following:

- Creates a pop-up menu named *"Example"* and stores a reference to its data structure in the *TrPopUpMenu* variable.
- Defines a bindkey for the Layout application to display a pop-up menu.

Key Description	Callback
Ctrl Shift<Btn2Down>(2)	<i>hiDisplayMenu(TrPopUpMenu)</i>

Running the Code

Load the code in *~/SKILL/Menus/PopUp.il*.

1. In the CIW, enter


```
load( "~/SKILL/Menus/PopUp.il" )
```
2. Open a *Layout* application window on a design of your choice.
3. Display the pop-up menu with the **Ctrl Shift<Btn2Down>(2)** bindkey. Use the middle mouse button to choose a menu item.

Is everything as you expected it to be?

Continue with the lab.

Examining the Code for a Pull-Down Menu

Use the *view* command to examine the *~/SKILL/Menus/Pulldown.il* file.

1. In the CIW, enter


```
view( "~/SKILL/Menus/Pulldown.il" )
```

What does the code do?

What is the name of the pull-down menu?

⇒ Check your work against the solution on the next page.

Answer

The source code creates a pull-down menu named "*Navigation*" and installs it in the leftmost position of the CIW menu banner.

Running the Code

Load the code in *~/SKILL/Menus/Pulldown.il*.

1. In the CIW, enter

```
load( "~/SKILL/Menus/Pulldown.il" )
```

The Navigation menu appears in the leftmost position in the CIW.

2. Display the *Navigation* pull-down menu.

Is everything as you expected it to be?

Lab Summary

In this lab, you

- Examined two SKILL source code files to determine what the code does.
- Loaded each file to verify the actual behavior of the SKILL source code.



Labs for Appendix B

Customization

Lab B-1 Defining Bindkeys in the .cdsinit File

Objective: Call the *hiSetBindKey* function in your .cdsinit file.

In this lab, you define three bindkeys in your .cdsinit file. These three bindkeys are immediately available the next time you access the Virtuoso® Design Environment.

In Lab 4-3, you defined three related bindkeys.

- Two bindkeys raise the CIW to the top of the screen. One bindkey applies to the *Schematics* application windows and the other applies to *Layout* application windows.
- A third bindkey raises the current window to the top of the screen. This bindkey applies to the *Command Interpreter* window.

1. Review your solution to Lab 1-3.

In this lab, you include that solution in your .cdsinit file.

2. Review the tests you performed to verify that the bindkeys behaved appropriately.

In this lab, you perform these tests again.

Continue with the lab.

Editing Your .cdsinit File

1. Use the *edit* function to study the .cdsinit file for this course. In the CIW, enter

```
edit( "~/SKILL/.cdsinit" )
```

An editor window displays the ~/SKILL/.cdsinit file.

2. Locate the section of the file that defines bindkeys.

```
;;;;;;;;;; Load Bindkey definitions ;;;;;;;;;;  
  
when( isFile( "leBindKeys.il" )  
      TrLoad( "leBindKeys.il" )  
      ) ; when  
  
when( isFile( "schBindKeys.il" )  
      TrLoad( "schBindKeys.il" )  
      ) ; when
```

3. Use the editor to add your *hiSetBindKey* function calls after two calls to the *TrLoad* function.

☞ Check your work against the solution on the next page.

Answers

Verify that the portion of your *.cdsinit* resembles the following:

```
;;;;;;;;;;;;;; Load Bindkey defintions ;;;;;;;;;;;;;;;

when( isFile( "leBindKeys.il" )
  TrLoad( "leBindKeys.il" )
) ; when

when( isFile( "schBindKeys.il" )
  TrLoad( "schBindKeys.il" )
) ; when

hiSetBindKey( "Command Interpreter"
  "<Key>F8" "hiRaiseWindow( hiGetCurrentWindow() )" )

hiSetBindKey( "Schematics"
  "<Key>F8" "hiRaiseWindow( window(1))" )

hiSetBindKey( "Layout"
  "<Key>F8" "hiRaiseWindow( window(1))" )
```

Continue with the lab.

Testing the *.cdsinit* File

1. Exit the Virtuoso Design Environment session.
2. Start the Virtuoso Design Environment.
3. Perform the tests for the lab.

Lab Summary

In this exercise, you edited the course *.cdsinit* file to make several bindkeys immediately available to the user.



Labs for Appendix C

File I/O

Lab C-1 Writing Data to a File

Objective: Use the *infile*, *fprintf* and *close* functions to write data to a file.

In this lab, you write data to a new file *./FileIO/MyFile.text*.

Obtaining an Output Port on a File

1. In the CIW, enter

```
myPort = outfile( "~/FileIO/MyFile.text" )
myPort
```

The system displays the print representation of the port:

```
port:./FileIO/MyFile.text"
```

2. Verify that the value of *myPort* is an output port by invoking the *outportp* function. In the CIW, enter

```
outportp( myPort )
```

What is the return result?

The return result indicates that the value is an output port.

Writing Data to the File

1. In the CIW, enter

```
lineCount = 0
fprintf( myPort "This is line # %d\n" lineCount++ )
fprintf( myPort "This is line # %d\n" lineCount++ )
fprintf( myPort "This is line # %d\n" lineCount++ )
```

2. Use the *fprintf* function to write the following line to the port:

```
No more lines
```

⇒ Check your work against the solution on the next page.

Answers

1. In the CIW, enter

```
fprintf( myPort "No more lines" )
```

Continue with the lab.

Closing the File

1. In the CIW, enter

```
close( myPort )
```

The port is no longer open.

2. Verify that the port is no longer open by attempting to write data to the port. In the CIW, enter

```
fprintf( myPort "This is line # %d\n" lineCount++ )
```

The CIW displays the following error:

```
*Error* printf/fprintf: cannot send output to a closed port -  
port:"~/FileIO/MyFile.text"
```

3. Since *myPort* has been closed, it is a good idea to set *myPort* to *nil*. In the CIW, enter

```
myPort = nil  
outportp( myPort )
```

Why is it a good idea to set myPort to nil?

In general, once you return a resource to the system, such as a file port, you should make sure no SKILL® variable contains a reference to the resource. Otherwise, your SKILL application might generate warnings or even errors if it attempts to access the resource.

Viewing the File

1. Verify the contents of the file by starting the *view* function. In the CIW, enter

```
view( "~/FileIO/MyFile.text" )
```

A view file window appears.

Lab Summary

In this lab, you used the

- *outfile* function to obtain an output port.
- *fprintf* function to write data to the file.
- *close* function to close the file.
- *view* function to verify the contents of the file.



Lab C-2 Reading Data from a Text File

Objective: Use various functions to read data from a text file.

In this lab, you read data from the file you created in the preceding lab. You can use the solution file.

Obtaining an Input Port on a File

Before you read data from a file, you need to obtain an input port.

1. In the CIW, enter

```
myPort = infile( "./Solutions/MyFile.text" )  
myPort
```

The system displays the print representation of the port:

```
port: "~./FileIO/MyFile.text"
```

Continue with the lab on the next page.

Reading the File a Line at a Time

Read a line from the file. Be aware of the order of the arguments to the *gets* function.

1. In the CIW, enter

```
nextLine = nil
gets( nextLine myPort )
nextLine
```

The *gets* function returns the next line as a SKILL string and also updates the value of *nextLine*. Notice that the string contains a "\n".

2. Read the next three lines from the file.

You are positioned at the end of the file.

3. Attempt to read the next line from the file.

What does the gets function return in this case?

Is the value of nextLine changed?

⇒ Check your work against the solution on the next page.

Answers

1. To read the next three lines from the file, enter the following in the CIW:

```
gets( nextLine myPort )  
gets( nextLine myPort )  
gets( nextLine myPort )
```

← You can paste the previous line from the log.

2. The *gets* function returns *nil* when positioned at the end of the file. The variable *nextLine* is not automatically updated.

Continue with the lab.

Closing the File

1. Close the port on the file.

Reading the Data From a Text File

Read words from a file and add each word to a list. The following steps guide you.

1. Obtain an input port on the *~/SKILL/FileIO/Solutions/MyFile.text* file.
2. To read the next word from the file, enter these two lines in the CIW:

```
fscanf( myPort "%s" word )  
word
```

The *fscanf* function returns the number of items read according to the conversion specifications.

3. Read the remaining words of the file. Initialize *wordList* to *nil*. Use the *cons* function to add each word to a list, *wordList*.

⇒ Check your work against the solution on the next page.

Answers

1. To read the next word from the file and add it to the list in *wordList*, enter these two lines in the CIW:

```
fscanf( myPort "%s" word )  
wordList = cons( word wordList )
```

Continue with the lab.

Closing the File

1. Close the port on the file.

Reading Numeric Data from a Text File

1. Obtain an input port on the *~/SKILL/FileIO/Solutions/MyFile.text* file.
2. In the CIW, enter

```
fscanf( myPort "%[^0-9]" phrase )  
phrase
```

The first SKILL expression reads the characters up to the next occurrence of *0-9* and stores the SKILL string in the variable *phrase*.

3. In the CIW, enter

```
fscanf( myPort "%d" number )  
number
```

The first SKILL expression reads the next numeric data item and stores its value in the variable *number*.

4. Read the remaining numeric items.
5. Close your input port.

⇒ Check your work against the solution on the next page.

Answers

1. The following code reads a single line in the file:

```
fscanf( myPort "[%0-9]" phrase )  
phrase  
fscanf( myPort "%d" number )  
number
```

Lab Summary

In this module, you used the

- *infile* function to obtain an input port on a file.
- *gets* function to read the file a line at a time.
- *fscanf* function to read data items from the file.
- *close* function to close the file.



Lab C-3 Writing Output to a File

Objective: To enhance the *TrShapeReport* function to write its output to a file.

In Lab 7-1, you wrote the *TrShapeReport* function. This function displays a summary report in the CIW. In this lab, you enhance this function to send the report to a temporary file and to display this file for the user in a Virtuoso® Design Environment text window.

Requirements

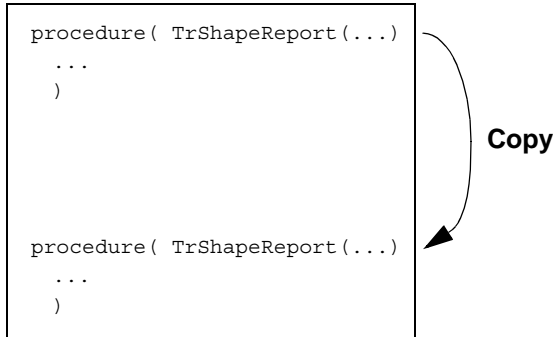
Enhance the *TrShapeReport* function to

- Send its output to a file instead of the CIW.
Send the program output to the */tmp/ShapeReport.txt* file.
- Display the file in a Show File window.
Use the *view* function to display the file in the window titled *"Shape Report"*.

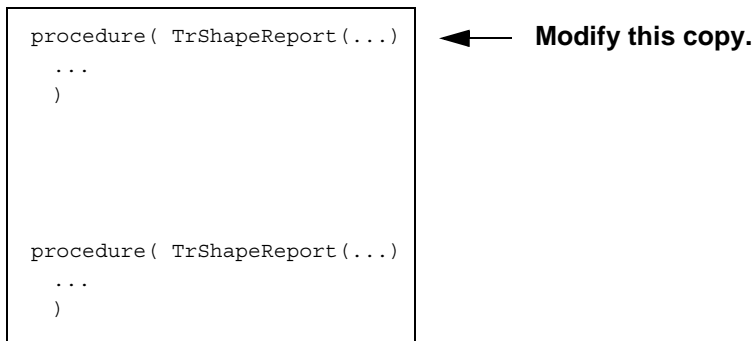
Continue with the lab on the next page.

Recommendations

1. Using the text editor, copy the declaration of the *TrShapeReport* function to a different place in the file.



2. Modify the original *TrShapeReport* function to call a new function named *TrShapeReportToPort*.



Specifically, modify the source code for the *TrShapeReport* function so that it does the following:

- Call the *outfile* function to obtain a port on the file.
`thePort = outfile("/tmp/ShapeReport.txt")`
- Call the *TrShapeReportToPort* function you write in the next step.
`TrShapeReportToPort(wid thePort)`
- Call the *close* function to close the port.
- Call the *view* function as follows. The second parameter, *nil*, asks for the default bounding box.

```
view( "/tmp/ShapeReport.txt" nil "Shape Report" )
```

3. Modify the copy of the *TrShapeReport* function that you made previously to create the *TrShapeReportToPort* function.

```

procedure( TrShapeReport (...)
...
)

procedure( TrShapeReport (...)
...
)

```

← **Modify this copy.**

Specifically, you need to do the following:

- Change the name from *TrShapeReport* to *TrShapeReportToPort* and add an *outport* parameter in addition to the *wid* parameter.

```

procedure( TrShapeReportToPort( wid outport )
...
) ; procedure

```

- Replace each call to the *printf* function with a call to the *fprintf* function. Be sure you include *outport* as the first argument.

Testing Your Solution

First Test

1. In the CIW, enter

```
nand2wid = geOpen( ?lib "master" ?cell "nand2" ?view "layout" ?mode "r" )
```

2. In the CIW, enter

```
TrShapeReport( nand2wid )
```

The CIW displays the following:

```
master nand2 layout contains:
Rectangles 10
Polygons   0
Paths      3
Misc       0
```

Second Test

1. In the CIW, enter

```
mux2wid = geOpen( ?lib "master" ?cell "mux2" ?view "layout" ?mode "r" )
```

2. In the CIW, enter

```
TrShapeReport( mux2wid )
```

The CIW displays the following:

```
master mux2 layout contains:
Rectangles 8
Polygons   0
Paths      7
Misc       6
```

⇒ Check your work against the solution on the next page.

Solutions

The `~/SKILL/FileIO/Solutions/TrShapeReport.il` file contains the following solution:

```

procedure( TrShapeReportToPort( wid output )
  let( ( rectCount polygonCount pathCount miscCount )
    rectCount = polygonCount = pathCount = miscCount = 0
    cv = geGetWindowCellView( wid )
    fprintf(
      output
      "%s %s %s contains:"
      cv~>libName cv~>cellName cv~>viewName
    )
    foreach( shape cv~>shapes
      case( shape~>objType
        ( "rect" rectCount++ )
        ( "polygon" polygonCount++ )
        ( "path" pathCount++ )
        ( t miscCount++ )
      ) ; case
    ) ; foreach
    fprintf( output "\n%-10s %-10d" "Rectangles" rectCount )
    fprintf( output "\n%-10s %-10d" "Polygons" polygonCount )
    fprintf( output "\n%-10s %-10d" "Paths" pathCount )
    fprintf( output "\n%-10s %-10d" "Misc" miscCount )
    list( cv~>libName cv~>cellName cv~>viewName )
  ) ; let
) ; procedure

TrReportBBox = ' ((120 729) (372 876))

procedure( TrShapeReport( wid )
  thePort = outfile( "/tmp/ShapeReport.txt" )
  when( thePort
    TrShapeReportToPort( wid thePort )
    close( thePort )
  ) ; when
  view( "/tmp/ShapeReport.txt" TrReportBBox "Shape Report" )
) ; procedure

```

Lab Summary

In this lab, you extended your *TrShapeReport* function to write its output to a file and to display the file in a Show File window. You used these functions:

- *fprintf* function
- *outfile* function
- *close* function
- *view* function

