Visual HyperScript User Manual

Ucamco Software Product Group
September 2016

Ucamco
Former Barco ETS

September 2016

Correspondence regarding this publication can be sent to:

Ucamco NV
Bijenstraat 19,
B-9051 Gent,
Belgium

For more information:

Our web site: http://www.ucamco.com

E-mail: info@ucamco.com

About Ucamco

Ucamco (formerly Barco ETS) is a market leader in PCB CAM software, photoplotting and direct imaging systems, with a global network of sales and support centers. Headquartered in Ghent, Belgium, Ucamco has over 25 years of ongoing experience in developing and supporting leading-edge photoplotters and front-end tooling solutions for the global PCB industry. Key to this success is the company's uncompromising pursuit of engineering excellence in all its products. Ucamco also owns the IP rights on the Gerber File Format through its acquisition of Gerber Systems Corp. (1998).

Helpdesk

Europe, Middle East, Africa, Latin Amerika	Asia Pacific	North America
Customer Support for Plotters, Software en ManiaBarco AOI Monday - Friday: 08.00 AM - 6.00 PM MET ✆ + 32 9 216 99 00 General support: support@ucamco.com License: license@ucamco.com Java™ HyperTool: hypertool@ucamco.com HyperScript: hyperscript@ucamco.com General information: info@ucamco.com Sales: sales@ucamco.com	Please contact our business partner for your country during support working hours see contacts page	Customer Support for Plotters, Software en ManiaBarco AOI Monday - Friday: 08.00 AM - 6.00 PM Pacific Time Saturday : 10 am to 4 pm Pacific Time +1 949 632 6895 General support: support@ucamco.us License: license@ucamco.us Java™ HyperTool: hypertool@ucamco.com HyperScript: hyperscript@ucamco.com General information: info@ucamco.us

Contents

1 Introduction

1.1 Document Description

Visual HyperScript (VHS) is a scripting environment for Ucam. It is designed to help UcamX (and Ucam v10) users to automate all kinds of operator actions. Today, over 1500 commands are available. Visual HyperScript covers almost the entire functionality of Ucam. The user can record simple repetitive tasks or he can build complete applications with Launch Pads and interactive Dialogs. Scripting is fun and can be done by people with any skill level.

The first part of the manual (Chapters 1 to 6) is designed to provide a basic Visual HyperScript training for absolute beginners where they can learn how to:

- Record UcamX actions into basic scripts
- Add scripts to the VHS Menu
- Run recorded scripts from the VHS menu

The second part (Chapters 7 to 17) is designed to provide a full Visual HyperScript training to UcamX operators where they learn how to:

- Enhance scripts with Comments, Variables, Decisions and Loops
- Add interactive & Startup Dialogs
- Create Launch Pads
- Add basic Java code

The third part (Chapters 18 to 19) is for operators with programming skills where they learn how to:

- Customize commands
- Create Standalone Scripts
- Add more complex java code

The fourth part (Chapters 20 to 21) contains a complete reference of the Visual HyperScript Editor and Menu Editor.

The fifth part (Chapter 22) contains a set of demo scripts. The source code of the scripts can be downloaded from the Ucamco website.

The last Chapter of the manual contains a Glossary where you can find more info on the terminology used in this manual.

Note: This manual does not contain a reference section of the Visual HyperScript scripting commands. The Visual HyperScript API Reference Guide is available as a separate document and can be accessed from the Visual HyperScript Editor Help menu

1.2 Who should use this Guide?

In order to use this Guide, you should have a fundamental understanding of PCB fabrication and the UcamX CAM software.

This Guide is to be used by:

- CAM operators
- Automation engineers
- Developers of software applications

1.3 About This Document

The following conventions are used in this document:

 **Note:**	Provides essential extra information.
 **Tip:**	Provides useful extra information.
 **Example:**	Contains examples of file syntax, commands, settings, etc.
 **Warning:**	Contains an important warning.

1.4 Other Available Documentation

Ucamco offers its customers a wide range of documentation on all its software, including installation requirements and procedures, release information, detailed task descriptions and info on customer support availability.

The following Ucamco documents are referenced in this manual:

- Visual HyperScript API Specification
- UcamX User Interface Guide

Both documents can be found on the download page of www.ucamco.com

If you need more info please mail us at info@ucamco.com

2 Situating Visual HyperScript

UcamX has 4 levels of automation, from basic level to advanced level.

2.1 Learns (Deprecated)

Recording of basic repetitive actions.

More information can be found in UcamX Help.

Tip: Learns are executed until stopped. In VHS you can simulate this behavior by assigning the LOOP property to a script (see 15.3.2, 21.3.1 and 21.4.1)

2.2 Checklists (Deprecated)

Follow up of pre-defined procedures with limited UcamX functionality (80 commands).

More information can be found in UcamX Help.

2.3 Visual HyperScript

Easy scripting tool for non programmers providing nearly full UcamX functionality.

2.4 HyperTool

Scripting tool for experienced Java programmers providing full UcamX functionality.

The HyperTool API Reference Guide can be found in the UcamX Help Menu.

3 Installation

This section describes how to set up your computer to start using Visual HyperScript.

3.1 Requirements

- Windows or Unix PC (see Ucamco Software Installation Requirements)
- Ucam v10.x or UcamX installed

3.2 Setting Up UcamX for using Visual HyperScript

Running Visual HyperScript requires the following ucam.db keys to be set:

vhs.home

Contains the path to the Visual HyperScript Home directory.

Default: HOME:/vhs

vhs.script.path

Contains a path to a VHS script directory used by import & run commands (see 12.3)

Default: HOME:/vhs/scripts

vhs.init.scripts

Contains a path to a VHS script directory containing custom commands (see 18.4)

Default: HOME:/vhs/init

All options set in the Option panel can also be set using ucam.db keys (see 20.2.4)

4 Getting Help

This section describes how to get help when using Visual HyperScript.

Visual HyperScript Help

Accessible via the **Help > On Application > Main Menu Bar > HyperScript menu option** in UcamX . You can also use the **Help > On Context > HyperScript** and select any menu option to get help on the selected menu option.

Visual HyperScript API Reference Specification

A link to the API is included in the Help page for the VHS Editor.

Visual HyperScript demo Jobs & Scripts

Some VHS demo jobs & scripts can be downloaded from our website.
More info can be found in chapter 22.

Ucamco Customer Support

Send all HyperScript issues to hyperscript@ucamco.com

See also Customer Care info on page iii

5 Getting Started

This chapter is for absolute beginners.
Here we learn how to record and run a basic script.

5.1 Recording a Basic VHS Script

1. Prepare UcamX for recording a script.
   e.g. to record a script that performs a specific action
   on the top copper layer in plane 1 you must:
   - Open a job
   - Set the top copper layer to plane 1
2. Select **Visual HyperScript Editor** from the **HyperScript** menu
3. Click  on the VHS Editor menu bar to create a new empty script
4. Click  to start recording
5. Perform the UcamX actions that need to be recorded
   Each UcamX action is recorded as a command line in the script Window
6. Click  to stop recording
7. Click  to save the script as **MyFirstScript**
8. Click  to add the script to the HyperScript menu
9. Close the Visual HyperScript Editor using the **Exit** option on the **File** menu

5.2 Running a Basic Recorded VHS Script

1. Prepare UcamX for running the recorded script.
   e.g. to run a script that performs a specific action
   on the top copper layer in plane 1 you must:
   - Open a job
   - Set the top copper layer to plane 1
2. Select **MyFirstScript** from the HyperScript menu
   and watch the script being executed

6 Managing Saved Scripts

In this chapter we learn how to use the VHS Editor to manage saved scripts.

6.1 Updating a Script

To add/edit commands, comments, variables, decisions and loops

1. Select **Visual HyperScript Editor** from the **HyperScript** menu
2. Click to open a saved script file
3. Place the cursor on line 1 in front the firtst command
4. Press the **Return** key
5. Type "// This is a recorded script" (without the quotes)
6. Click **File >Save** or to save the script
   To save the script using a new name see 6.3

Tip: During editing press **Ctrl+Space** to see a list of all possible commands. If you type the first characters of a command, **Ctrl+Space** will show all commands beginning with the characters you have typed. This auto complete function allows you to select any command out of the displayed list. It is up to you to replace the dummy parameters with suitable values.

Note: Commands that have a correct syntax are color coded. This gives you a visual feedback in case you have misspelled a command name.

6.2 Adding UcamX Actions to a Script

To add new UcamX actions to a saved script we (re)use the Record function:

1 Click to open a saved script file

1 Place the cursor in the script at the location where the new command(s) need to be inserted

2 Click to start recording

3 Perform the UcamX actions that need to be recorded

4 Click to stop recording

5 Click to save the script
To save the script using a new name see 6.3

6.3 Renaming a Script

To rename saved/updated scripts:

1 Click to open a saved script file

2 Click **File > Save As…** to save the script using a new name

3 Click if you want to add the script to the HyperScript menu

Note: To remove the old script name from the HyperScript menu you need to use the VHS Menu Editor (see 21.2)

6.4 Deleting a Script

A saved script cannot be deleted using the VHS Editor.
You need to use the file manager from your operating system to remove the file.

7 Enhancing Recorded Scripts

A basic recorded script can only replay the recorded actions. In the next chapters we look at how to extend the functionality of recorded VHS scripts by adding:

Commands & Java Code

VHS commands and Java code is manually added to provide more functionality

Chapter 8 explains how to add VHS commands and Java code

Comments

Comments are used inside a script to provide information about the script.

Chapter 9 explains how to enter comments.

Variables

Variables are used inside a script to store information. This information can be used during the execution of the script

Chapter 10 explains how to define and use Variables.

Interactive Dialogs

Interactive Dialogs are used to exchange information with the user during the execution of a script.

Chapter 11 explains all types of dialogs.

 **Note:** Startup Dialogs can use the same dialog types (see Chapter 17)

Flow Control

Flow Control instructions are standard java instructions used to break up the flow of execution of a script to allow conditional execution of certain blocks of code.

Chapter 12 explains how to use flow control to optimize your scripts

8 Enhancing Scripts – Commands & Code

8.1 Writing VHS commands

There are more than 1500 VHS commands available, most of them are recordable.

Some of the recordable VHS commands also have a variant with a different parameter set (e.g. one command with object coordinates and the same one with x,y coordinates). To record this variant, you can change a record mode parameter in the optionpanel of the VHS Editor. If the command with the required parameter set cannot be recorded by changing the record mode than you will need to enter the command by hand (see updating scripts 6.1)

Note: All VHS commands are described in the Visual HyperScript API Specification. This document is part of the VHS documentation set and is also accessible from the HyperScript help page.

8.2 Writing Java code

Visual HyperScript is a full option scripting language based on Java. So it is possible to write scripts that use the full functionality of the Java programming language including complex mathematics and user interfaces. Variables need to be typed.

It is beyond the scope of this document to provide detailed info about the Java language. Please consult the Oracle site for more information.

http://download.oracle.com/javase/tutorial/java/index.html

If the HyperScript environment has not enough power to satisfy your automation needs than you can always switch to HyperTool, Ucamco's high-end automation product. More info can be obtained by sending your inquiry to hypertool@ucamco.com

9 Enhancing Scripts – Comment

9.1 Intro

Comment is non executable descriptive text added by the script writer.
Comment is shown in green.

It is advisable to add lots of comment to your scripts:

- To make sure that anyone can understand your script
- To facilitate future maintenance on the script
- To help you find info (e.g. matching brackets if the nesting becomes complex)

9.2 Syntax

Single line comment

```
// This is a single line of comment
```

Comment can also be added at the end of a command line e.g.

```
apeCreateCircle(1,0.75); // Create a circle aperture
```

Multi line comment

```
/* This is an example of
   multi line comment */
```

This type of comment is used to document the parameters of a function

```
/**
 * Create Circle aperture
 *
 * @param apeNum Aperture Number
 * @param dia Diameter
 */
```

In-line comment

```
apeCreateCircle(1 /* ape number */ ,0.75 /* ape size */);
```

10 Enhancing Scripts - Variables

10.1 Intro

Variables are containers to store values during the execution of a script.

A Variable can be:

- Used for calculations
- Used for branching
- Printed to the console window
- Displayed in a Prompt Dialog

Visual HyperScript has 4 kinds of Variables:

- Program Variables
- Pre-defined Variables
- Environment Variables
- Ucam.db Keys

10.2 Program Variables

Program Variables are user defined Variables to be used in the script.

10.2.1 Typing Variables

Visual HyperScript is a loosely typed scripting langue. It means that in most cases there is no need to specify the type of a Variable. VHS uses the context of the command to decide which type to assign to a Variable.

Java: **double dAbc = 123.456; // dAbc must be typed as double**

HyperScript: **dAbc = 123.456; // dAbc will be typed as double because 123.456 has decimals**

10.2.1.1 Java types used in VHS

Type	Description & Assignment
Boolean	**true** or **false** **bVar = true;** Using a promptBoolean command (see 11.4.6)
String	Text characters **sVar = "This is a string";** Using a promptString command (see 11.4.4) Using a promptOption command (see11.4.6)

	Using a promptFileName command (see 11.4.10)
Option List	List of strings **oVar = [{ “option_1”, “option_2”}];**
Integer	Whole Number **iVar = 123;** Using a promptInteger command (see 11.4.7)
Double	Number with decimals **dVar = 123.456;** Using a promptDouble command (see 11.4.8)
Double Unit	Number with decimals and unit Use the prompt Unit command to set a unit value (see 11.4.9)

10.2.1.2 HyperScript types (classes)

Type	**Description & Assignment**
Arc	Using 3 points and sense **aVar = Arc (Point pFrom, Point pTo, Point pCenter, String sSense)** Using 6 coordinates and sense **aVar = Arc (double dFromX, double dFromY, double dToX, double dToY, double dCenterX, double dCenterY, String sSense)** Using 6 coordinates, sense and units **aVar = Arc (double dFromX, double dFromY, double dToX, double dToY, double dCenterX, double dCenterY, String sSense, String sUnits)** Using a copy of an existing arc (aMyArc) **aVar = Arc (Arc aMyArc)**
Point	Using X and Y coordinates with unit **pVar = Point(double dX, double dY, String sUnits) ;** Using X and Y coordinates without unit **pVar = Point(double dX, double dY) ;** Using a copy of an existing point (pMyPoint) **pVar = Point(Point pMyPoint) ;** Using a promptpoint command (see 11.4.11) Using a pickPoint command (see 11.5)
Line	Using X and Y coordinates of “from” and “to” point with unit **lVar = Line(double dFromX, double dFromY, double dToX, double dToY, String sUnits);** Using X and Y coordinates of “from” and “to” point without unit **lVar = Line(double dFromX, double dFromY, double dToX, double dToY);**

	Using “from” and “to” points without unit **lVar = Line(Point pFrom, Point pTo);** Using a copy of an existing line (lMyLine) **lVar = Line(Line lMyLine);** Using a promptLine command (see 11.4.12) Using a dragLine command (see 11.5)
Rectangle	Using left bottom and right top points **rVar= Rectangle(Point pPoint1, Point pPoint2) ;** Using rectangle boundaries and unit **rVar = Rectangle(double dXmin, double dYmin, double dXmax, double dYmax, String sUnits);** Using rectangle boundaries without unit **rVar = Rectangle(double dXmin, double dYmin, double dXmax, double dYmax) ;** Using a copy of an existing rectangle (rMyRect) **rVar = Rectangle(rMyRect)) ;** Using a promptRectangle command (see 11.4.13) Using a dragRectangle command (see 11.5)

10.2.2 Assigning a Value to a Variable

- Using a manual assignment

```
sJobName = "MyJob";
```

- Using a dialog function (see chapter 11)

```
sName = promptString("Name:", "Enter here your name");
```

- Using a VHS command
  e.g. to obtain the value of the current job:

```
sJobname = jobName();
```

Tip: VHS has lots of get/set commands:
- the command without parameter gets the value
- the command with parameter sets the value

10.2.3 Obtaining/Displaying the Value of a Variable

- Using a print to console command

```
print("Distance between pt1 and pt3: " + dDist);
```

- Using a dialog function (see chapter 11.3)

```
viewWarning(iReturnCode);
```

- Using single line print commands in the VHS Editor (see xxx)

```
print "Name: " + sVar; print "Number: " + iVar;
```

- Using the Variables Window of the VHS Editor during debugging (see 20.2.5)

10.3 Using “methods” on variables

Variables, used in scripts, often need some processing. For each variable type there are a number of functions (methods) available.

10.3.1 For Java types

e.g to find the location of character “a” inside a string sMyString you can use the command: `sMystring.indexOf(”a”);`

All methods for Java types can be found in the Java API

http://www.oracle.com/technetwork/java/javase/documentation/api-jsp-136079.html

10.3.2 For VHS types

All methods for HyperScript types are found in the Classes section of the VHS API.

Examples can be found in the Demo Scripts (See Chapter 23)

10.4 Pre-defined Variables

10.4.1 Visual HyperScript file info Variables

Name	Type	Value
FILE_TYPE	int	0
FILE_PARENT	int	1
FILE_ATTRIBUTES	int	2
FILE_SIZE	int	3
FILE_MODIFICATION_DATE	int	4
FILE_NAME	int	5

The fole info Variables have a fixed value. They are used to index the objectlist returned by the getLayers() command. Each index retrieves its corresponding value.

Examples

To retrieve layer info for all layers:

```
list = osGetFileList("C:\\Windows\\Web\\Wallpaper\\Windows", true);

int counter = 1;
path = forEachItem(list) {
     print("---- File " + (counter++) + " ----");
     info = osFileInfo(path);
     print("  type " + info[FILE_TYPE]);
     print("  parent " + info[FILE_PARENT]);
     print("  attibutes " + info[FILE_ATTRIBUTES]);
```

```
print("  size " + info[FILE_SIZE]);
print("  modification date " + info[FILE_MODIFICATION_DATE]);
print("  name " + info[FILE_NAME]);
     }
```

To get the file name of the 3th file in the list:

```
osFileInfo(list[2])[FILE_NAME];
```

10.4.2 Visual HyperScript layer info Variables

Name	Type	Value
LAYER_NAME	int	0
LAYER_CLASS	int	1
LAYER_SUBCLASS	int	2
LAYER_ATTACH	int	3
LAYER_INDEX	int	4
LAYER_ACTIVITY	int	5
LAYER_APERTURES	int	6

The layer info Variables have a fixed value. They are used to index the objectlist returned by the getLayers() command. Each index retrieves its corresponding value.

Examples

To retrieve layer info for all layers:

```
int counter = 1;
 info = forEachItem(getLayers()) {
     print("---- Layer " + (counter++) + " ----");
     print("  name " + info[LAYER_NAME]);
     print("  class " + info[LAYER_CLASS]);
     print("  subclass " + info[LAYER_SUBCLASS]);
     print("  attach " + info[LAYER_ATTACH]);
     print("  index " + info[LAYER_INDEX]);
     print("  active " + info[LAYER_ACTIVITY]);
     print("  active " + info[LAYER_APERTURES]);
 }
```

To get the activity status of the second layer:

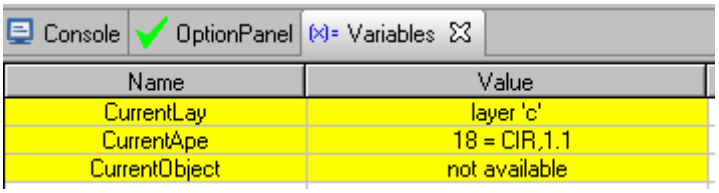
```
getLayers()[1][LAYER_ACTIVITY];
```

To get the number of apertures of the first layer:

```
getLayers()[0][LAYER_APERTURES];
```

Note: The index count starts with 0.

10.4.3 VHS Debugger Variables.



These Variables provide essential feedback during debugging of a script avoiding the need for extra print instructions in the script.

10.5 Ucam.db Keys

Visual HyperScript can get/set Ucam.db key values.

The db keys can hold values of 6 different types:
String, Boolean, Integer, Double, Unit Value and Path

Note: The **Path** type is processed in VHS as a String

There is a command for each Variable type.

To get the value of a key

- sVar = dbString(String dbKey);
- bVar = dbBoolean(String dbKey);
- iVar = dbInteger(String dbKey);
- dVar = dbDouble(String dbKey);
- uVar = dbUnitValue(String dbKey);
- sVar = dbPath(String dbKey);

To set the value of a key

- dbString(String dbKey, String sVar);
- dbBoolean(String dbKey, Boolean bVar);
- dbInteger(String dbKey, Integer iVar);
- dbDouble(String dbKey, Double dVar);
- dbUnitValue(String dbKey, Double uVar);
- dbPath(String dbKey, String sVar);

Example: sVHSHome = **dbPath**("vhs.home");// get ucam.db key

10.6 Environment Variables

Visual HyperScript can retrieve the value of operating system environment Variables.

Example

To load & print the location of your temporary storage location into a Variable:

```
myTemp = envString("TEMP");
print(myTemp);
```

The console window shows the path:

C: \Temp

Note: There are also **osCreateTmpDir** and **osMarkAsTmp** commands available to create & mark temporary storage locations

11 Enhancing Scripts - Dialogs

11.1 Intro

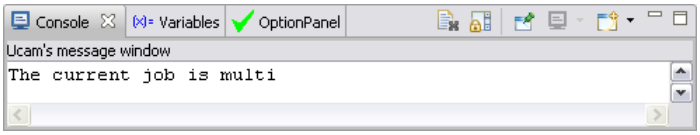
Scripts have several ways of exchanging information with the user during the execution of a script:

- Print – Print a message to the Console Window
- Warning – Display a simple message with a OK button
- Prompt – Display & request any type of Variable information.
- Pick & Drag – Request for Point, Line or Rectangle coordinates
- Run – Branch to another script

11.2 Print

The print command can be used during the execution of a script to write feedback to the Console Window. The print command is in fact a BeanShell command. That is why it is not in the VHS command list and why it does not use color coding. The “+” character can be used to join text and Variables.

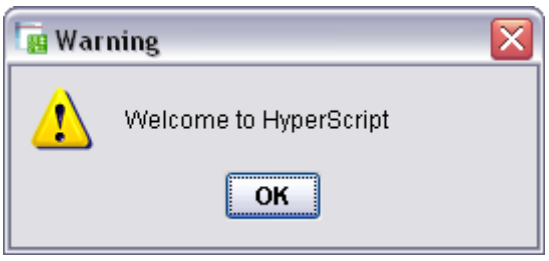
```
print("The current job is " + jobName());
```



11.3 Warning

The warning command is used to display a text message on screen until the OK button is pressed. Meanwhile the script continues to execute. As long as the OK button is not pressed, all consecutive warnings will be added to the same dialog.

```
viewWarning("Welcome to HyperScript");
```



Note: viewWarning can not be used in Standalone Scripts (see 19.8), use the promptLabel command instead.

11.4 Prompt

A prompt sequence halts the execution of the script and displays a dialog on screen. The user can perform any UcamX action, including entering/selecting values in the dialog, before clicking

- The **Continue** button to continue the execution of the script
- The **Abort** button to stop the execution of the script.

 **Tip:** The **[Esc]** key can also be used to abort a script

A prompt sequence consists of a **promptStart** command, a **promptEnd** command and, in between, any number of the following prompt commands:

- promptLabel
- promptString
- promptOption
- promptBoolean
- promptInteger
- promptDouble
- promptUnit
- promptFileName
- promptPoint
- promptLine
- promptRectangle

The following sections describe all elementary prompt commands. Each prompt sequence contains 1 single prompt command. The Demo Chapter contains an example of a Prompt Dialog which uses all possible prompt commands (see 22.9).

11.4.1 PromptStart

The prompStart commands initiates a prompt dialog.

It is also possible to specify :

- a parameter set name
  parameter sets can save a set of parameters for later reuse
  for an example of a dialog with a parameter set see 22.9
- a dialog name
  for an example of a dialog with a dialog name see 11.4.3

```
promptStart("MyParSet","Welcome");
```

 Note: The parameter set and dialog name are optional

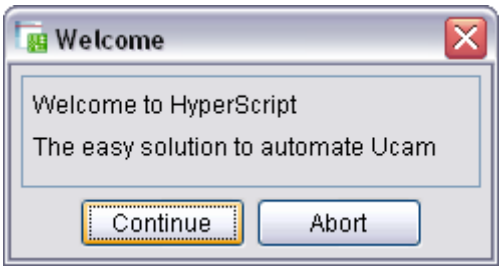
11.4.2 PromptEnd

The prompEnd commands closes a prompt dialog.

11.4.3 PromptLabel

A promptLabel command adds one or more text lines to a Prompt Dialog box.

```
promptStart("","Welcome"); // Dialog name "Welcome"
  promptLabel("Welcome to HyperScript");
  promptLabel("The easy solution to automate Ucam");
promptEnd();
```



11.4.4 PromptString

A promptString command adds a text label and text entry field to a Prompt Dialog box. The entered value is assigned to a string Variable (sVar).

```
promptStart();
  sVar = promptString("Name:", "Enter here your name");
promptEnd();
```

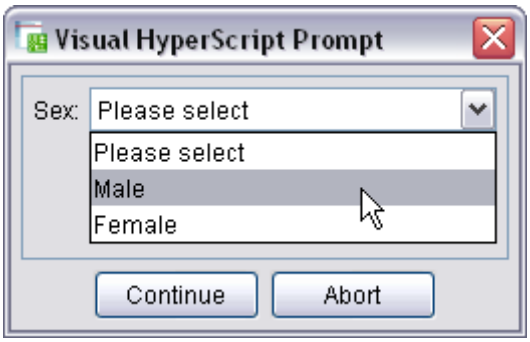
11.4.5 PromptOption

A promptOption command adds a text label and a selectable drop down field to a Prompt Dialog box. The selected value is assigned to a string Variable (sVar).

Tip: Your code is more easy to read if you use a Variable (e.g. oDef) to define the option values at the start of your script

```
oDef = [{"Please select","Male","Female"}];
promptStart();
```

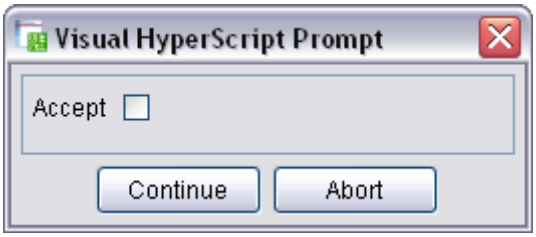
```
  sVar = promptOption("Sex:", oDef, "Please Select");
promptEnd();
```



11.4.6 PromptBoolean

A promptBoolean command adds a text label and a check box to a Prompt Dialog box. The state of the check box is assigned to a boolean Variable (bVar).

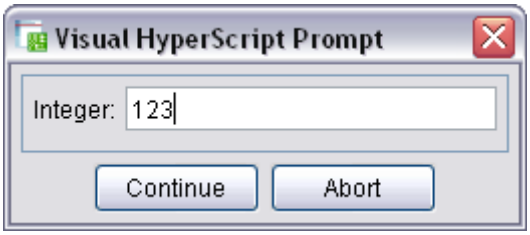
```
promptStart();
  bVar = promptBoolean("Accept", false);
promptEnd();
```



11.4.7 PromptInteger

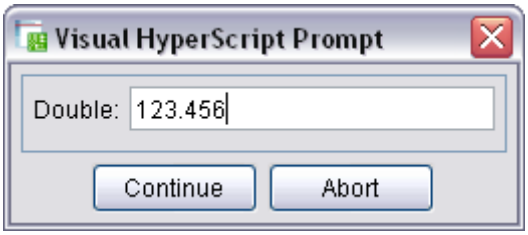
A promptInteger command adds a text label and an integer number entry field to a Prompt Dialog box. The entered value is assigned to an integer Variable (iVar).

```
promptStart();
  iVar = promptInteger("Integer:", 123);
promptEnd();
```



11.4.8 PromptDouble

A promptDouble command adds a text label and a decimal number entry field to a Prompt Dialog box. The entered value is assigned to a double Variable (dVar).



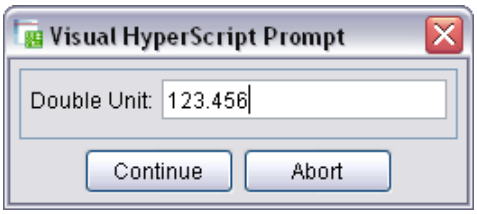

```
promptStart();
  dVar = promptDouble("Double:", 123.456);
promptEnd();
```

11.4.9 PromptUnit

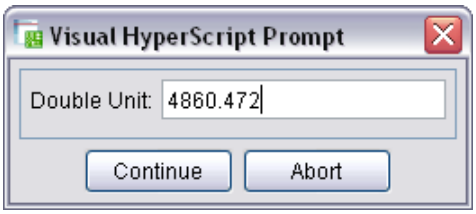
A promptUnit command adds a text label and a decimal number entry field to a Prompt Dialog box. The entered value is assigned to a unit Variable (uVar), a double Variable using the unit supplied in the prompt command. If you change the current unit (e.g from **mm** to **mil**) the Variable will be converted to the changed unit.

Note: The unit is not entered or displayed in the entry field.

```
promptStart();
  uVar = promptUnit("Double Unit:", 123.456, "mm");
promptEnd();
```



promptUnit with current unit “mm”

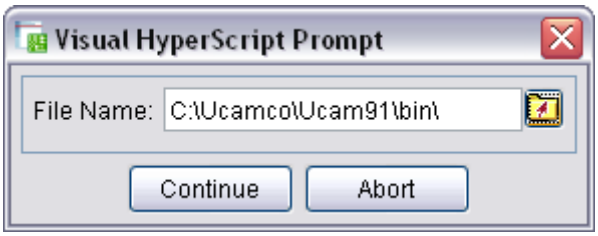


promptUnit with current unit “mil”

11.4.10 PromptFileName

A promptFileName command adds a text label, a text entry field and a file selector icon to a Prompt Dialog box. The entered value is assigned to a string Variable (fVar).

```
promptStart();
  fVar = promptFileName("File Name:", "");
promptEnd();
```



11.4.11 PromptPoint

A promptPoint command adds point coordinate (X,Y) entry fields and a Pick button to a Prompt Dialog box. The entered values are assigned to a Point Variable (pVar).

The user has 3 ways to enter the coordinates:

- Accept the default coordinates and click Continue
- Enter new (double) coordinates and click Continue
- Click the Pick button, click a location in the main window and click Continue

```
promptStart();
  pVar = promptPoint("Point", Point(100, 150));
promptEnd();
```

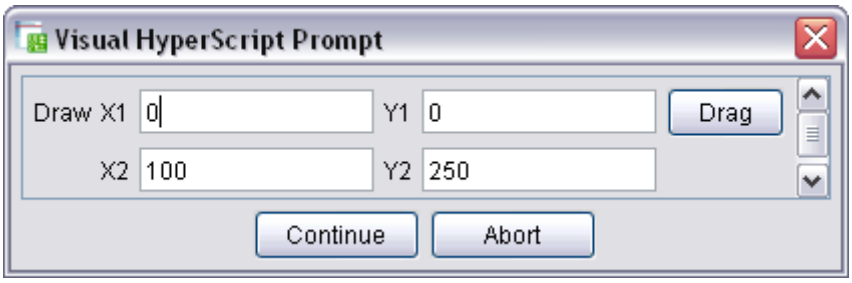
11.4.12 PromptLine

A promptLine command adds start point (X1,Y1) end point (X2,Y2) entry fields and a Drag button to a Prompt Dialog box. The entered values are assigned to a Line Variable (lVar).

The user has 3 ways to enter the coordinates:

- Accept the default coordinates and click Continue
- Enter new (double) coordinates and click Continue
- Click the Drag button, drag a line in the main window and click Continue

```
promptStart();
  lVar = promptLine("Draw", Line(0, 0, 100, 250));
promptEnd();
```



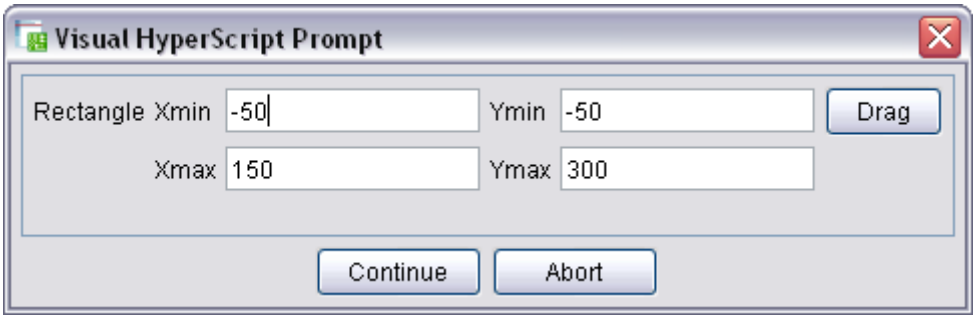
11.4.13 PromptRectangle

A promptRectangle command adds left bottom point (X1,Y1) end right top (X2,Y2) entry fields and a Drag button to a Prompt Dialog box. The entered values are assigned to a Rectangle Variable (rVar).

The user has 3 ways to enter the coordinates:

- Accept the default coordinates and click Continue
- Enter new (double) coordinates and click Continue
- Click the Drag button, drag a rectangle in the main window and click Continue

```
promptStart();
  rVar = promptRectangle("Rectangle", Rectangle(-50, 150, -50, 300));
promptEnd();
```



11.4.14 Combining Prompt Commands into 1 Dialog

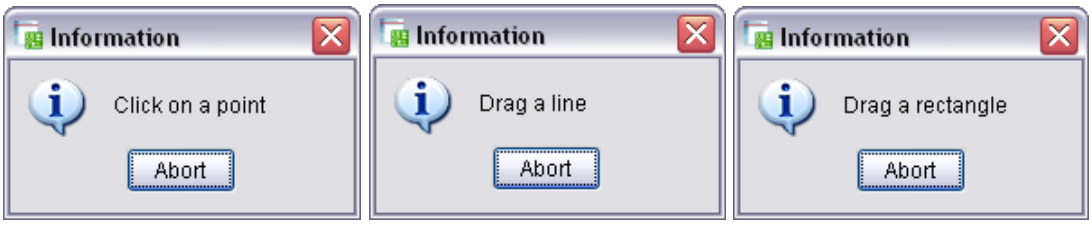
Any number of prompt commands can be combined in a Prompt Dialog.

To find out how, please have a look at the **Showing all "Promptable" Variable Types** demo script in chapter 22.9

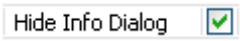
11.5 Pick & Drag

The **promptPoint**, **promptLine** and **promptRectangle** commands are not the only way to get Point, Line and rectangle coordinates.

During a script execution it is also possible to ask a user for Point, Line or Rectangle coordinates without displaying the coordinate values. This can be done using the **pickPoint, dragLine** or **dragRectangle** command. Each function halts script execution, shows an Information window* and waits for user interaction.



*The display of this info Dialog can be set/reset in the Option Panel (see 20.2.4)



Example 1: Use pick & drag commands to load coordinates into Variables

```
pVar = pickPoint("Click on a point");  //Get Point coordinates
lVar = dragLine("Drag a line"); //Get Line coordinates
rVar = dragRectangle("Drag a rectangle"); //Get rectangle coordinates
```

Example 2: Use pick & drag commands to insert objects

```
// Insert flash with current aperture
// on picked X,Y coordinates
insertFlash(pickPoint("Click on a point"));
// Insert draw with current aperture
// using dragged X1,Y1 and X2,Y2 coordinates
insertDraw(dragLine("Drag a line"));
// Insert rectangular polydraw with current aperture
// using dragged Xmin,Ymin and Xmax,Ymax coordinates
insertPolydrawRect(dragRectangle("Drag a rectangle"));
```

12 Enhancing Scripts - Flow Control

12.1 Intro

Flow Control instructions are standard java instructions used to break up the flow of execution of a script to allow conditional execution of certain blocks of code.

We have 3 types of flow control statements:

- Decision making statements
- Branching statements
- Loop commands

12.2 Decision making statements

Java

- if-then
- if-then-else
- switch

12.3 Branching statements

Java

- break
- continue
- return

Detailed information about java flow control statements can be found on the site of Oracle, owner of the java language

http://download.oracle.com/javase/tutorial/java/nutsandbolts/flow.html

HyperScript

- **importFile**
  Import an external script file and continue when it is finished
  for an example: see 0
- **importScript**
  Import and run a list of semicolon separated VHS commands

Note: The code is executed inside the current environment, so you can access Variables and functions defined in the external script after the import command has been completed.

Warning: Imported code cannot be debugged using HyperScript Debugger.

- **runFile**
  Run an external script and continue when it is finished
  for an example: see 22.3.3
- **runScript**
  Run a number of HyperScript commands separated by a semicolon

Note: The code is executed outside the current environment (Variables will not be affected), however the script gets a copy of all current Hyper-Script Variables, so it is possible to pass arguments.
Jobs, layers etc. may be changed by the external script, so using the runFile or runScript command in a forEachLayer loop may result in unpredictable results.

12.4 Loops

12.4.1 Intro

Loops are used to repeat a block of code or even entire scripts.

Visual HyperScript has 3 types of loops:

- Standard Java loops in scripts
- VHS Loops in scripts
- LOOP flag in dialogs (see 21.3.1, 21.4.1 and 15.3.2)

12.4.2 Standard Java Loops

- for
- while
- do while

Detailed information about java loops can be found on the site of Oracle, owner of the java language

http://download.oracle.com/javase/tutorial/java/nutsandbolts/flow.html

For an example of a **while** loop: see 22.12

12.4.3 VHS Loops

Visual HyperScripts adds a very specific loop, the forEach loop.

The forEach loop is used to access each successive item in a collection of items. These items can be layers, apertures, objects, objectlists, nets, ….

HyperScript has 16 custom forEach loop commands

- forEachJobNet
- forEachLayerNet
- forEachLayer

- forEachDrill
- forEachExtra
- forEachSignal
- forEachApe
- forEachObject
- forEachItem
- forEachNet
- forEachInRectangle
- forEachRegion
- forEachFlash
- forEachDraw
- forEachArc
- forEachVtxt

forEach commands for Integr8tor and Panel Editor

- forEachI8job
- forEachPEInputJob
- forEachPEPanelJob
- forEachPESolution

Notes: forEach commands are not recordable

Tip: Allways check if there is a variant of a specific forEach command before using the generic command without parameters. e.g. the forEachDrill command has a generic form without parameters and another with parameter subclass. Using the subclass parameter could save you the effort of having to test for the subclass in the script.

12.4.3.1 Nesting of forEach Loops

Use the following simplified nested forEach loops as a guideline for making your own nested structures. The "|" character means that you can choose between multiple valid options.

```
forEachLayer | forEachDrill | forEachExtra | forEachSignal {
  // vhs commands
  forEachApe {
    // vhs commands
    forEachObject | forEachInRectangle {
      // vhs commands
      forEachRegion | forEachFlash | forEachDraw | forEachArc | forEachVtxt {
        // vhs commands
      }
    }
  }
}
```

12.4.3.2 Examples

Example 1: forEachApe and forEachDraw

```
// print the start and end coordinates of all selected draws
forEachApe() {
  forEachDraw() {
    if (objSelect()) {
      print(objFromPoint() + " " + objToPoint());
    }
 }
}
```

Note: The enclosing forEachApe loop is necessary to make sure that the aperture used for the selected draw is marked as current.

Example 2: forEachNet

```
// Select all objects of net iNet on layer in plane 1
iObjCount = 0;
iNet = 20;
forEachNet(iNet) {
  objSelect("+");
  objCount++;
}
print(iObjCount + " object(s) of net " + iNet + " selected.");
```

Examples 3 & 4: forEachItem

```
// Print all items of an objectlist
iCounter = 1;
sItem = forEachItem([{"CIR", "REC", "BOX", "COM", "THE"}]) {
  print("Item " + (iCounter++) + " is " + sItem);
}
```

```
// Get and print all layer names
 int counter = 1;
 item = forEachItem(getLayerNames()) {
     print("Layer name " + (counter++) + " is " + item);
 }
```

Notes: More examples can be found in the Demo Scripts (see Chapter 22)

12.4.4 LOOP flag in Dialogs

When adding a script to a Launch Pad, it is possible to set the loop flag in the Item Properties of the script. This way the script will keep on looping until the user clicks the abort button (or presses the Esc key) on any script dialog (see 21.3.1, 21.4.1 and 15.3.2)

13 Best practices

13.1 Naming conventions

For Variables

Variables do not need to be typed but you can make life easy by prefixing each Variable name with its type. This way you always know the type of a Variable.

Variable name prefixes

- String: **s** – e.g. sVar, sName …
- Option: **o** – e.g. oVar, oState
- Boolean: **b** – e.g. bVar, bAccept
- Integer: **i** – e.g. iVar, iAge
- Double: **d** – e.g. dVar, dAmount
- Unit: **u** – e.g. uVar, uLength
- Point **p** – e.g. pVar, pFlash
- Line **l** – e.g. lVar, lDraw
- Rectangle **r** – e.g. rVar, rArea
- Arc **a**– e.g. aVar, aMyArc
- FileName: **f** – e.g. fVar, fReadme

Note: A filename is a string with a special format.
Therefore you can also use **s**

For Scripts, Custom Commands and Packaged Launch Pads

Try to use a consequent naming scheme.

13.2 Feedback

Use print commands to display information about the script during execution. Never give the operator the impression that the script is hanging while in fact it is performing a time consuming forEach loop. Write some feedback to the console window so the Operator knows that everything is still OK.

Print commands used for debugging can be left in the script as comment.
This can be useful in case of future updates.

Note: You do not always need to put a print command inside your script. During debugging you can use the VariablesView window in the VHS Editor to examine the value of a Variable (see 20.2.5) or you can use single line print commands (see 14.3).

Tip: Write lots of comments!

14 Debugging VHS Scripts

The VHS Editor has a build-in script debugger.

14.1 Buttons

Icon	Description
	Start debugger
	Stop debugger
	Step into
	Run next step
	Run to line with cursor
	Hold execution
	Terminate script

14.2 The Variables Window

The Variables window in the VHS Editor displays the value of customer defined Variables and pre-defined Variables (CurrentLay, CurrentApe and CurrentOpbject) during debugging of a script. The window can be made visible using the **Windows > Variables** option or the keyboard shortcut Ctrl+Alt+V

14.3 Single Line Scripts

The button bar has a drop down list containing single line scripts (i.e VHS commands separated by a semicolon)

Each single line script can be executed by selecting it from the drop down list

- Before a script is started
- During a debug stop
- After execution of a script

This can be used e.g. to print info to the console window instead of putting the print commands hardcoded inside the script.

 **Note:** Single line scripts can also be executed using the importScript and runScript command (see12.3)

15 Editable Launch Pads

An Editable Launch Pad is a dialog window from which you can launch personal scripts.

15.1 Create an Editable Launch Pad

1 Open the VHS Menu Editor
2 Add a menu item
3 Set Type to Editable Launch Pad
4 Enter a menu item label
5 Click OK to accept menu item properties
6 Click OK to accept the menu item

At this point we have created an empty Launch Pad.
The next step is to open this Launch Pad and to add Sections & Buttons.

15.2 Open an Editable Launch Pad

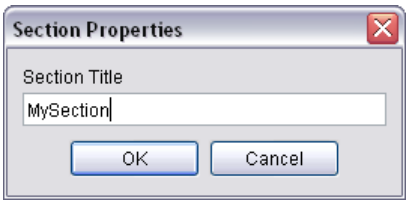
You can open Editable Launch Pads from the HyperScript menu.

15.3 Add items to an Editable Launch Pad

The right mouse button can be used to add:

- Sections
- Buttons

15.3.1 Add a Section



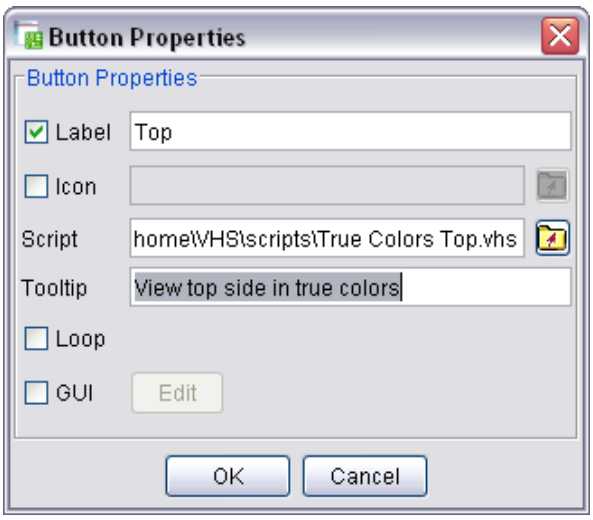
Section Title	Name of the inserted section
OK	Accept current Section Properties
Cancel	Cancel the Section Properties Dialog

To add a section, right click inside a Launch Pad and select **Add Section…**

To change a section title, right click inside a section and select **Section Properties…**

To delete a section, right click inside a section and select **Delete Section**

15.3.2 Add a Button



Label	Name of the button
Icon	Path to icon
Script	Path to the script (.vhs) file
Tooltip	Info text when hovering over this button
Loop	Activate if script must be looped until Abort button clicked
GUI	Activate if script needs initial dialog
Edit	Create/Edit initial dialog for current script (see 17.2)
OK	Accept current button properties
Cancel	Cancel the Button Properties Dialog

- To add a button, right click inside a Launch Pad section and select **Add Button…**
- To change button properties, right click the button and select **Button Properties...**
- To delete a button, right click the button and select **Delete Button**

15.4 Convert an Editable to a Packaged Launch Pad

Editable Launch Pads are only for personal use. If there is a need to distribute them to other users, the Editable Launch Pad needs to be converted into a Packaged Launch Pad (.hst). A Packaged Launch Pad is in fact a Launch Pad archive with all its referenced scripts & Icons inside.

The conversion can be done as follows:

1 Right click inside an Editable Launch Pad but outside a section
2 Select **Save as Packaged Launch Pad…**

To run the Packaged Launch Pad on another UcamX station:

1 Move the .hst file to the $HOME\VHS\LaunchPads directory
2 Add the Packaged Launch Pad to the HyperScript menu (see 21.3.3)

16 Packaged Launch Pads

Packaged Launch Pads are Launch Pad archives containing all referenced scripts & icons. They provide an easy way to transfer Launch Pads (including scripts and buttons) to other computers running Ucam. Packaged Launch Pads can also be added to the HyperScript menu using the VHS Menu editor (see xxx).

Packaged Launch Pads cannot be edited. If you need to change script or button properties than you will need to convert the Packaged Launch Pad to an Editable version.

16.1 Create a Packaged Launch Pad

There is no way to build a Packaged Launch Pad from scratch. You need to create an Editable Launch Pad and convert it into a Packaged one (see 15.4).

16.2 Convert a Packaged to an Editable Launch Pad

To run the Packaged Launch Pad on another UcamX station:

1 Move the .hst file to a $HOME\VHS\LaunchPads directory
2 Add the Packaged Launch Pad to the HyperScript menu (see 21.3.3)
3 Select the Packaged Launch Pad from the VHS Menu

To convert a Launch Pad from Packaged to Editable:

1 Right click inside a Packaged Launch Pad but outside a section
2 Select **Convert to Editable Launch Pad…**

17 Script Startup Dialogs

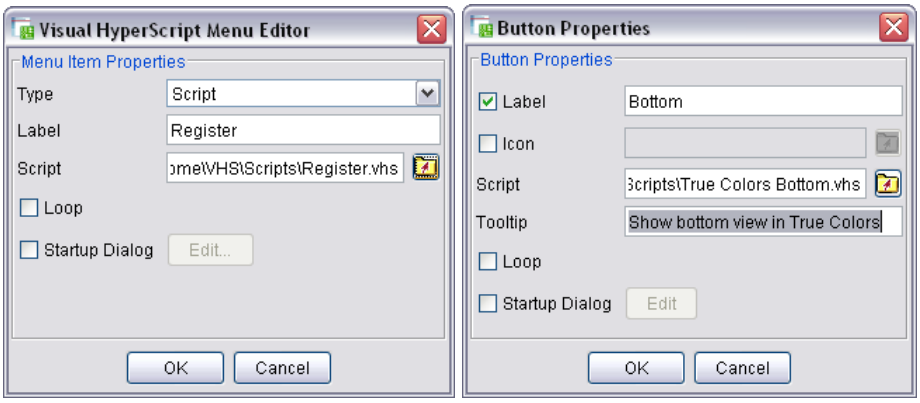
When launching a script directly from the HyperScript menu or from an Editable Launch Pad, it is also possible to start the script with an initial dialog. This dialog can be used to retrieve all parameters needed to run the script. This way the script can run unattended and does not have to be interrupted to request user input. The dialog with the run parameters stays visible on the screen during the execution of the script.

Note: For an example of a Startup Dialog: see 22.3.2

17.1 Creating a Startup Dialog

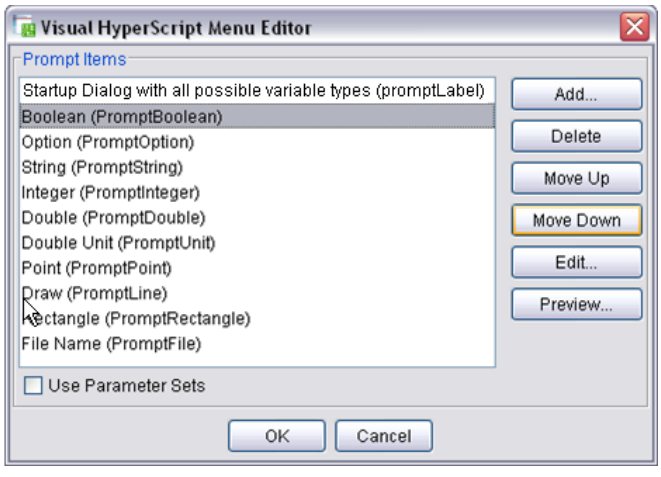
A Startup Dialog can be defined on 2 locations:

- When Adding or Editing a menu item of type script in the VHS Menu Editor (see 21.4.1)
- When Adding a Button or editing Button Properties in an Editable Launch Pad (see 15.3.2)



17.2 Editing a Startup Dialog

Click the Edit button to see the Startup Dialog editor



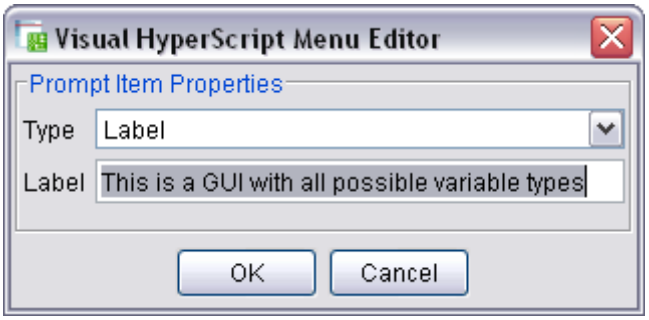
17.2.1 Buttons

Label	Description
Add	Add a new Startup Dialog item and specify its properties
Delete	Delete the selected Startup Dialog item
Move Up	Move the selected Startup Dialog item up
Move Down	Move the selected Startup Dialog item down
Edit	Edit the properties of the selected Startup Dialog item
Preview	Preview the Startup Dialog
OK	Accept the current Startup Dialog setup
Cancel	Cancel the Startup Dialog Editor

Note: Options are grayed out if the corresponding action is not applicable

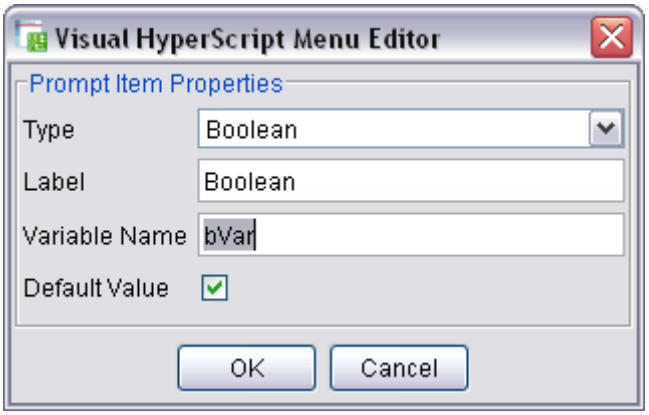
17.2.2 Add a Label

Click **Add** and select Type = Label



17.2.3 Add a Boolean

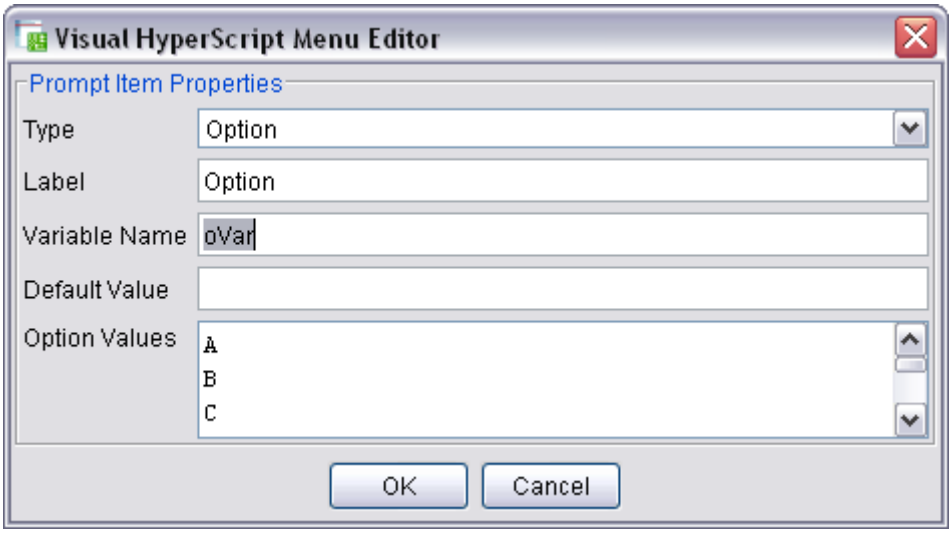
Click **Add** and select Type = Boolean



Check Default Value box if default value is “true”

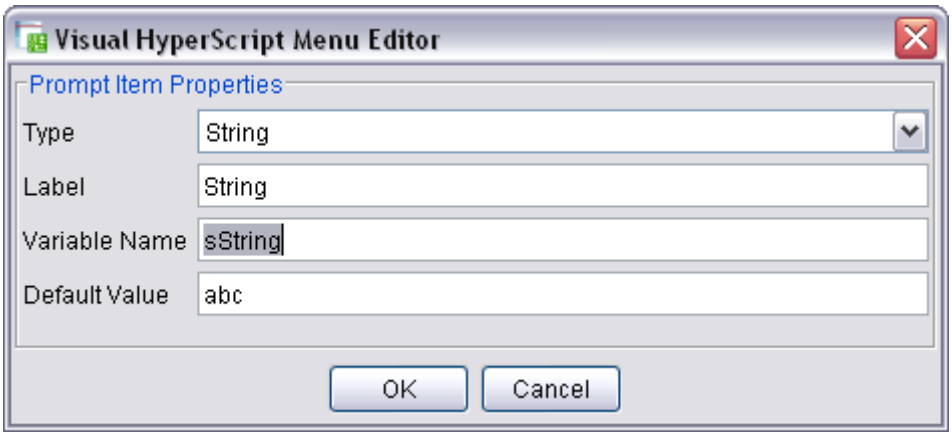
17.2.4 Add an Option List

Click **Add** and select Type = Option



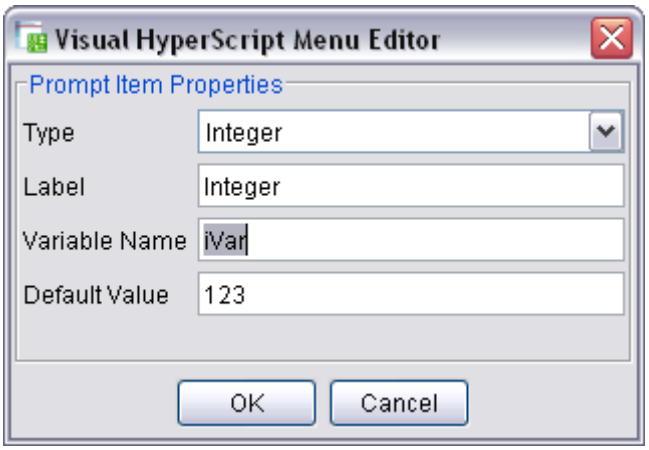
17.2.5 Add a String

Click **Add** and select Type = String



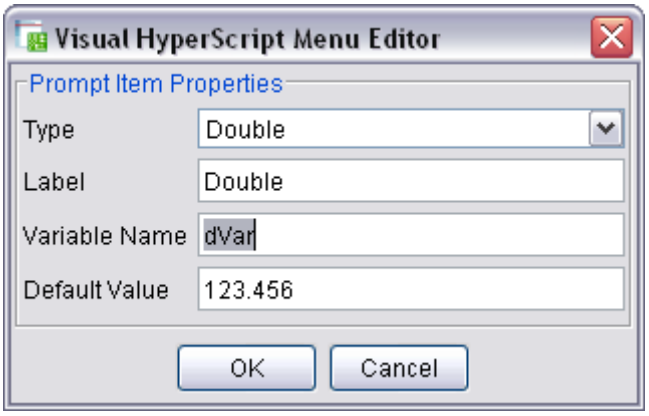
17.2.6 Add an Integer

Click **Add** and select Type = Integer



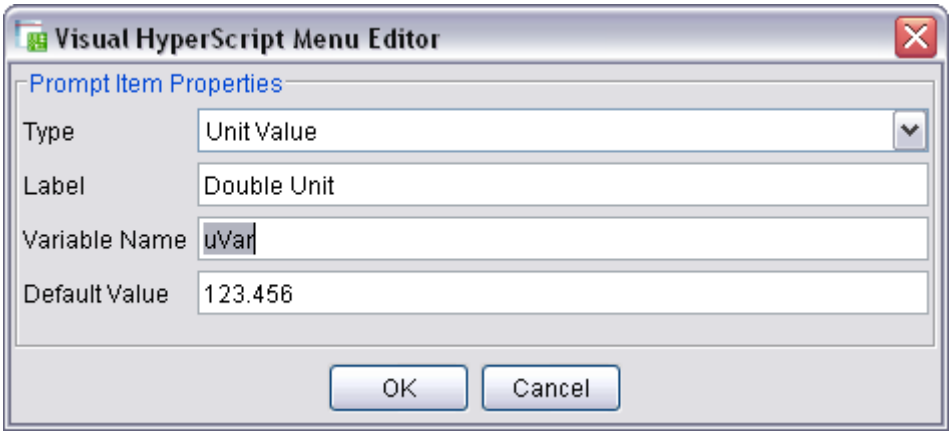
17.2.7 Add a Double

Click **Add** and select Type = Double



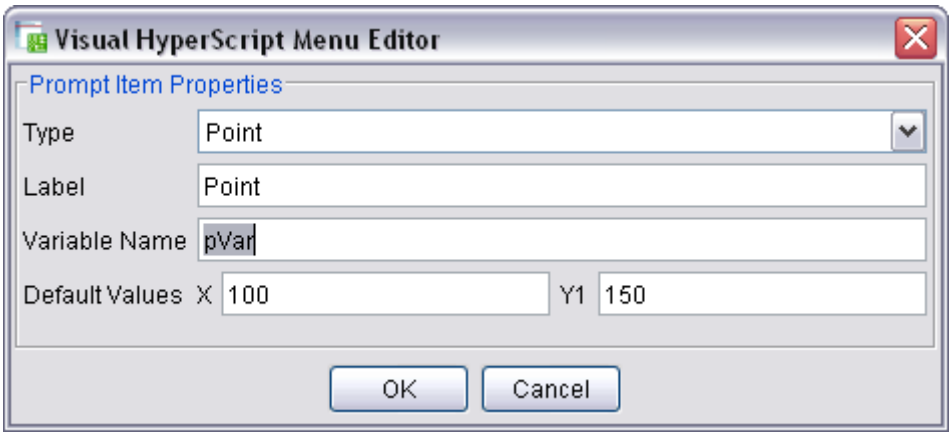
17.2.8 Add a Unit Value

Click **Add** and select Type = Unit Value



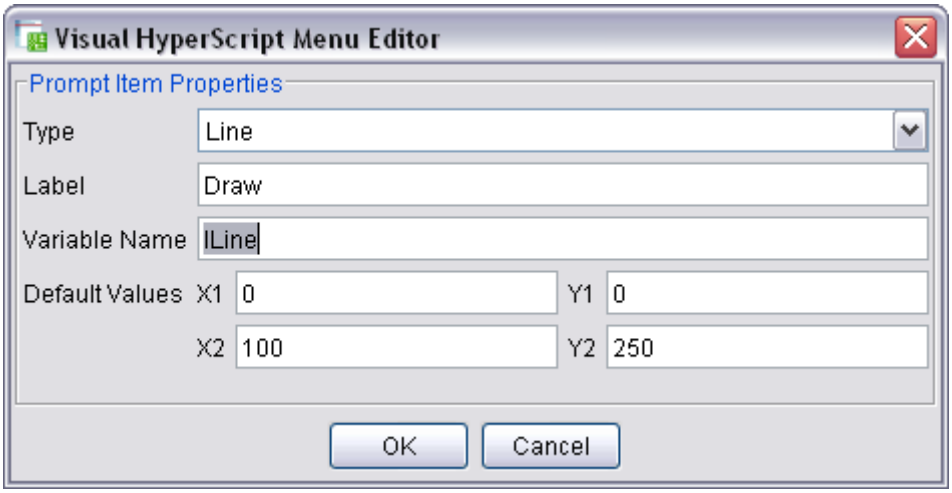
17.2.9 Add a Point

Click **Add** and select Type = Point



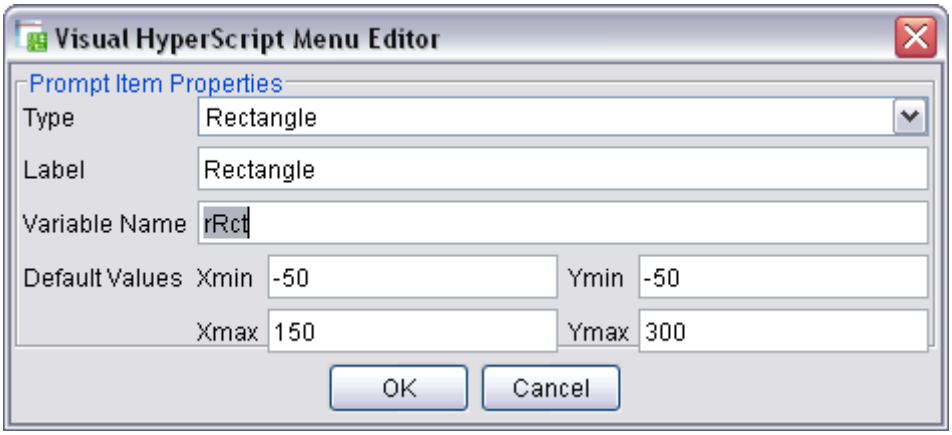
17.2.10 Add a Line

Click **Add** and select Type = Line



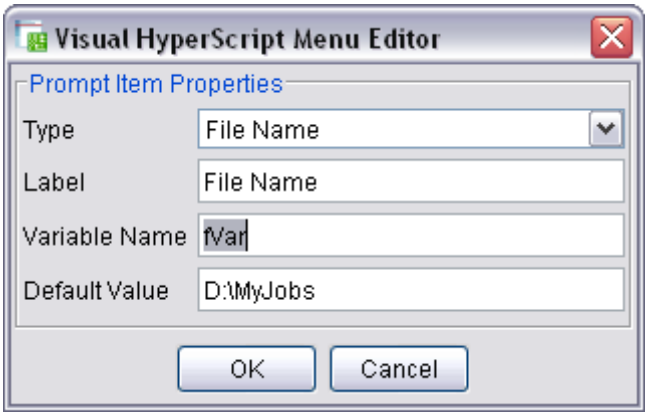
17.2.11 Add a Rectangle

Click **Add** and select Type = Rectangle



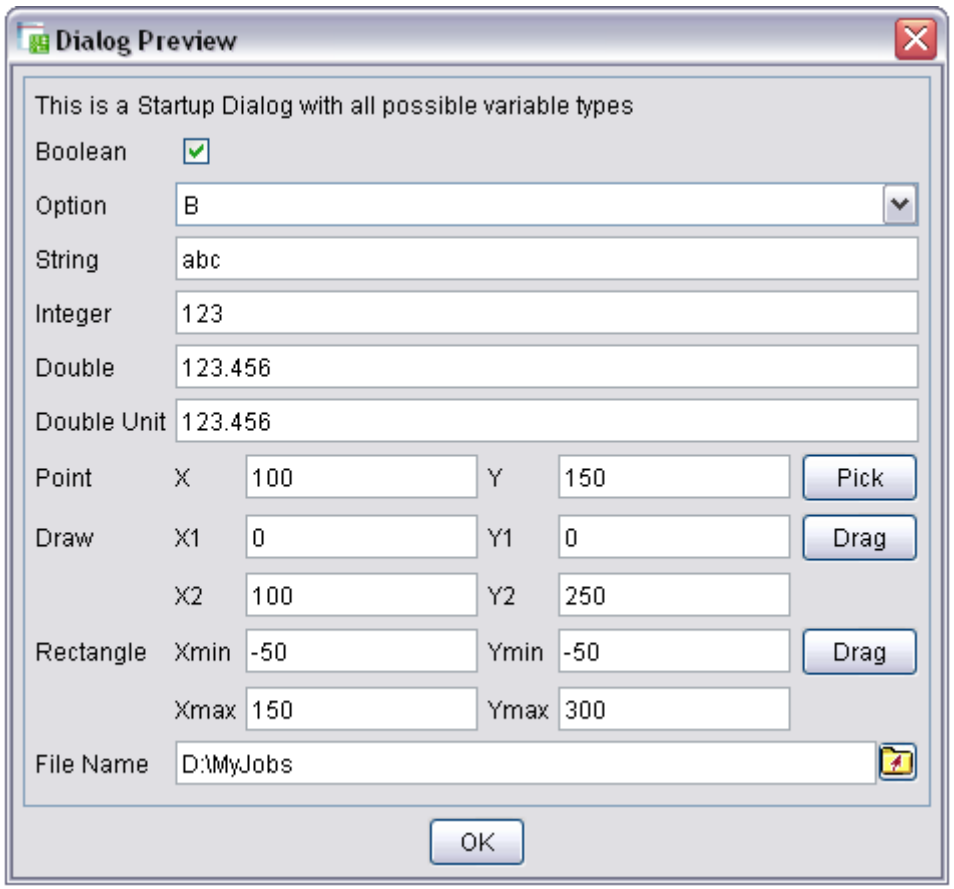
17.2.12 Add a File Name

Click **Add** and select Type = File Name



17.2.13 Preview

Click **Preview** to see the completed Startup Dialog



18 Customizing VHS Functions

The Visual HyperScript API Specification lists all available VHS commands. Most of the UcamX functionality is already included. If you need more functionality, than you will need to do some customization.

18.1 Write custom commands

Why?

- Functionality – to add custom functions
- Localization – to translate commands into your own language

Example: for a new command that reads a text file and prints each line to the console we could use the following code:

```
// This program reads a text file and prints each line to the
console.
// It uses FileOutputStream to read the file.
public void fileInput(String sFileName) {
   File file = new File(sFileName);
   FileInputStream fis = null;
   BufferedInputStream bis = null;
   DataInputStream dis = null;
     try {
       fis = new FileInputStream(file);
       // BufferedInputStream is added for fast reading
       bis = new BufferedInputStream(fis);
       dis = new DataInputStream(bis);
       // dis.available() returns 0 if the file has no more lines
       while (dis.available() != 0) {
         // reads the line from the file and prints it to the console
         System.out.println(dis.readLine());
       }
       // dispose all the resources after using them
       fis.close();
       bis.close();
       dis.close();

     } catch (FileNotFoundException e) {
       e.printStackTrace();
     } catch (IOException e) {
       e.printStackTrace();
     }
}
```

Save this code as **fileInput.vhs** in your init directory and restart UcamX.

Run the new command as **fileInput(<path to a file>).**

Tip: VHS contains a number of Operating System commands that can be used to handle files. All Operating System commands start with “os”.

18.2 Modify existing commands

Why?

- Functionality – to create modified versions of functions
- Add/remove Parameters – to add custom or remove obsolete parameters
- Customize parameters – to preset parameters with custom values

18.3 Include and/or combine existing commands

Commands can be included and/or combined into new functions which are not yet available,

e.g. to implement a percentual scale command we could combine

- **scale(val)** - scales object dimensions and coordinates
- **distort(Xval,Yval)** - scales coordinates without changing object dimensions

into a **usr_scale_pct(n)** command that would apply a percentual scale of n% to (selected) objects without affecting the coordinates.

```
/**
* Percentual scale of selected apertures
* @param Ipct Percentual scale value
*/
void usr_scale_pct(double Ipct){
  double Dpct = Ipct / 100.0;
  scale(Dpct, false); // This is a VHS command
  double Ddistort = 1.0 / Dpct;
  distort(Ddistort, Ddistort); // This is a VHS command
}
```

Save this code as **usr_scale_pct.vhs** in your init scripts directory.

Note: The demo chapter shows how to do a percentual scale using prompt commands (see 22.6.1) or using a custom function (see 22.6.2).

18.4 Load custom commands into UcamX

Custom commands are loaded at startup of UcamX. The **vhs.init.scripts** ucam.db key is used to indicate the location of your custom commands. e.g set **vhs.init.scripts** to **HOME:/VHS/init** to have all command files (.vhs) in the scripts folder available in UcamX.

19 Integration of VHS Scripts in UcamX

A VHS script can be executed in 7 ways:

Easy

- From the VHS Editor
- From the HyperScript Menu
- From an Editable VHS Launch Pad
- From a packaged VHS Launch Pad

Advanced

- From any other UcamX Menu
- From a Toolbar
- As a Startup Script
- As a Standalone Script

19.1 Executing a Script from the VHS Editor

As long as a script is under development it will be launched from the VHS Editor.
As soon as a script is ready, it will be launched using 1 of the 7 other methods.

1 Open the VHS Editor
2 Open a script (.vhs)
3 Click the “Run” button

19.2 Executing a Script from the HyperScript Menu

This is the preferred way of launching your scripts.

1 Open the VHS Editor
2 Open a script (.vhs)
3 Click the ‘’Add HyperScript Menu Item’’ button
4 Enter the Menu Item name
5 Click OK

The script can now be launched from the HyperScript menu.

19.3 Executing a Script from an Editable Launch Pad

A Launch Pad is a dockable dialog containing user defined buttons.
Each button invokes a vhs script.

1 Add an Editable Launch Pad to the HyperScript menu (see 21.3.2)
2 Add a button, linking to a script to the Editable Launch Pad (see 15.3.2)

The Launchpad is called using its menu option in the HyperScript menu

The script is launched using the button on the Launch Pad.

19.4 Executing a Script from a Packaged Launch Pad

1 Add a Packaged Launch Pad (.hst) to the HyperScript menu (see 21.3.3)

All scripts included in the Pakaged Launch Pad can now be launched from the buttons on the packaged Launch Pad in the HyperScript menu.

Note: If you want to edit any of the Packaged Launch Pad objects than you need to convert the Packaged Launch Pad to an Editable Launch Pad. You can do this by right clicking the Launch Pad and selecting the **Convert to Editable Launch Pad** option. The **<launch pad name>.hst** file will be replaced by a folder **<launch pad name>** containing all editable scripts (.vhs)

19.5 Executing a Script from any other UcamX Menu

Use the Toolbar Manager to:

1 Add the script as a new source command

2 Assign the new source command to an existing menu

A detailed procedure can be found in the UcamX User Interface Guide (chapters 8.8 to 8.11)

19.6 Executing a Script from a Toolbar

Use the Toolbar Manager to:

1 Add the script as a new source command

2 Add the new source command to an existing toolbar

A detailed procedure can be found in the UcamX User Interface Guide (chapters 8.8 and 8.2)

19.7 Executing a Script as a Startup Script

A script can be used to perform some initialization tasks. A fast way to implement this is to change the target field of the UcamX startup icon:

1 Right-click the UcamX startup icon

2 Select Properties from the context menu

The target field contains something like:

`C:\Ucamco\UcamX2016\bin\ucamXLauncher.bat`

3 Add –vhs ‘quoted script path’ to the Target field
e.g. `C:\Ucamco\UcamX2016\bin\ucamXLauncher.bat –vhs: "D:\HyperScript\Scripts\MyScript.vhs"`

4 Click OK

5 Double click the UcamX startup icon to see how UcamX is started and MyScript.vhs is executed

19.8 Executing a Script as a Standalone Application

Scripts can also be used as a Standalone applications. They can make use of all UcamX functionality without even starting the UcamX main window.

A special batch file **VHSExecuter.bat** takes care of this.
By default it is located in the …\UcamX2016\bin directory.
Make sure that your **Path** environment Variable includes a path to this executable.

1 Click **Start > Run**

2 Enter ‘cmd’ and press enter
A DOS window with command prompt appears

3 Enter VHSExecuter’ ‘quoted script path‘ ‘parameters’ and press enter
e.g. VHSExecuter.bat “D:\scripts\JobInfo.vhs” \“Multi\”


Note: string parameters must be enclosed by \”


Tip: The Demo chapter contains an example of a Standalone script (see 22.11)

20 Visual HyperScript Editor

20.1 Introduction

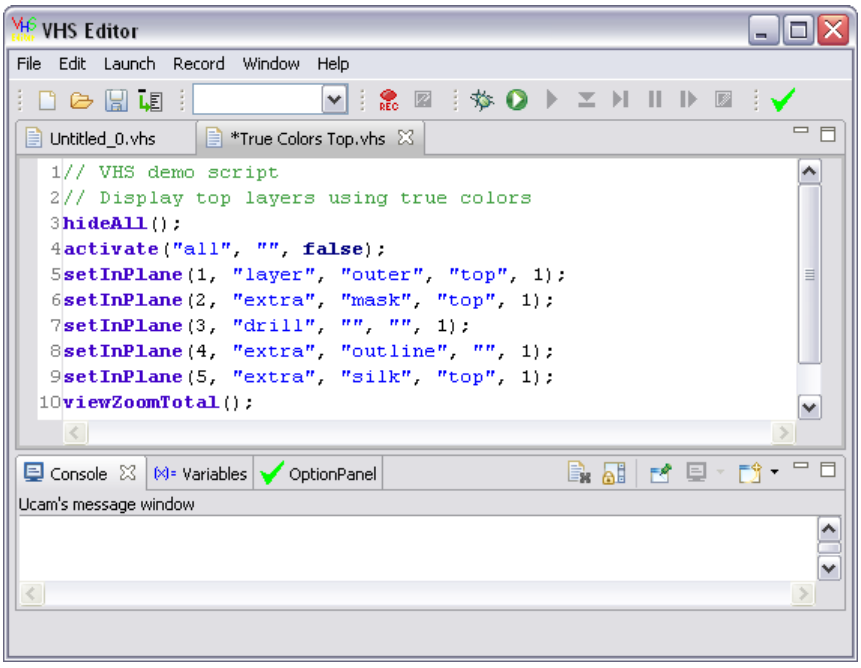
The Visual HyperScript Editor is a user friendly environment for recording, editing and debugging VHS scripts.

```
promptStart("","Welcome"); // Dialog name "Welcome"
  promptLabel("Welcome to HyperScript");
  promptLabel("The easy solution to automate UcamX");
promptEnd();
```

Example script displaying a Welcome message

20.2 VHS Editor Dialog

The VHS Editor is launched from the HyperScript menu.



The VHS Editor Dialog consists of:

- Menu Bar
- Button Bar
- Editor Area with Tabbed Panels
- Option Panel

20.2.1 Menu Bar

File Menu

Menu Option	Description	Icon on Button Bar or shortcut
New File	Create a new script file (.VHS) A new tab page is added	
Open File	Open an existing script file (.VHS)	
Close	Close the current tab page	CTRL+W
Close All	Close all tab pages	CTRL+SHIFT+W
Save	Save script in default script directory	
Save As …	Save script using File Selection Dialog	
Save All	Save all scripts in current location	
Add HyperScript Menu Item	Adds a link to the current script to the HyperScript Menu	
Exit	Exit VHS Editor	

Edit Menu

Menu Option	Description	Icon on Button Bar
Undo	Undo last action	
Redo	Redo last action	
Find/Replace	Find/Replace string in script	

Launch Menu

Menu Option	Description	Icon on Button Bar
Debug	Start debugger	
Run	Run script	
Step	Run next step	
Step Into	Step into	
Run to Line	Run to line with cursor	
Suspend	Hold execution	
Resume	Stop debugger	
Terminate	Stop execution	

Record Menu

Menu Option	Description	Icon on Button Bar
Start Recording	Start script recording	
Stop Recording	Stop script recording	

Window Menu

Icon	Menu Option	Description	Shortcut
	Console	Displays script execution feedback Output of print() commands is shown here	Ctrl+Alt+C
	Variables	Displays the value of customer defined Variables and pre-defined Variables (CurrentLay, CurrentApe and CurrentOpbject) during debugging of a script	Ctrl+Alt+V
	Option-Panel	Displays selectable record options	Ctrl+Alt+O

Help Menu

Icon	Menu Option	Description	Shortcut
	VHS Editor Help	Shows Help page with direct link to VHS User Manual	
	VHS API Reference	Shows Help page with direct link to VHS API Specification	
	About VHS Editor	Shows VHS Editor version information and installation details	

20.2.2 Button Bar

Buttons

Icon	Description
	Open new empty script (.vhs)
	Open existing script
	Save current script
	Add HyperScript Menu Item
	Drop down list of single line commands (see 14.3)
	Start Recording
	Stop Recording
	Start debugger
	Run script
	Run next step
	Step into
	Run to line with cursor
	Hold execution
	Stop debugger
	Stop execution
	Show Option Panel

20.2.3 Editor Area with Tab Pages

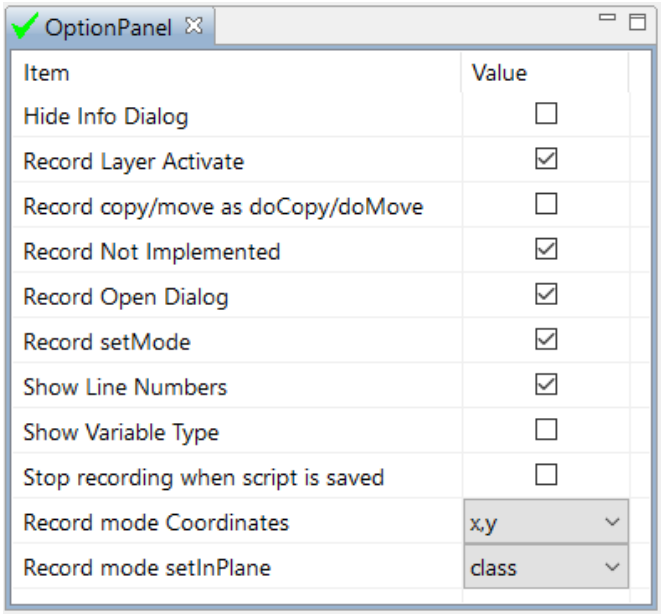
The Editor Area holds one tab page for each open script. Scripts are made current by clicking on the title in the tab.

Buttons

Icon	Description
	Show list of other (2) tab pages
	Minimize VHS Editor area
	Maximize VHS Editor area
	Restore previous VHS Editor view (when minimized)
	Show Editor area (when minimized)

20.2.4 Option Panel

The Option Panel allows the user to select/deselect recording options.
All options can also be set using ucam.db keys



Recording options - with corresponding ucam.db keys

Hide Info Dialog - vhs.options.hide_info_Dialog

Hide Information Dialog for pick & drag commands

Default: false

Record Layer Activate - vhs.options.record_layer_activate

Set active to record individual activation of layers via checkbox

Default: true

Note: The Activation using “Activate” menu is always recorded

Record copy/move as doCopy/doMove - vhs.options.record_copymove_as_doCopydoMove

Set active to record copy and move actions as doCopy or doMove

Default: false

Record Not Implemented - vhs.options.record_not_implemented

Activate to mark non implemented commands

Default: true

Record Open Dialog - vhs.options.record_open_Dialog

Set active to record opening and closing of UcamX Dialogs

Default: true

Record setMode - vhs.options_record_setmode

Always start recording with a setMode command

Default: true

Show Line Numbers - vhs.options.show_line_numbers

Show line numbers in front of code (needed for debugging)

Default: true

Show Variable Type – vhs.options.show_Variable_type

Show type of Variables in Variables View during debugging

Default: false

Record mode Coordinates - vhs.options._record_mode_ coordinates

Choose how coordinates are recorded

- **x,y** - using coordinates
- **object** - using point, line and rectangle commands
- **pick** - using dragPoint, dragLine and dragRectangle commands

Default: object

Record mode setInPlane – vhs.options.record_mode_ setinplane

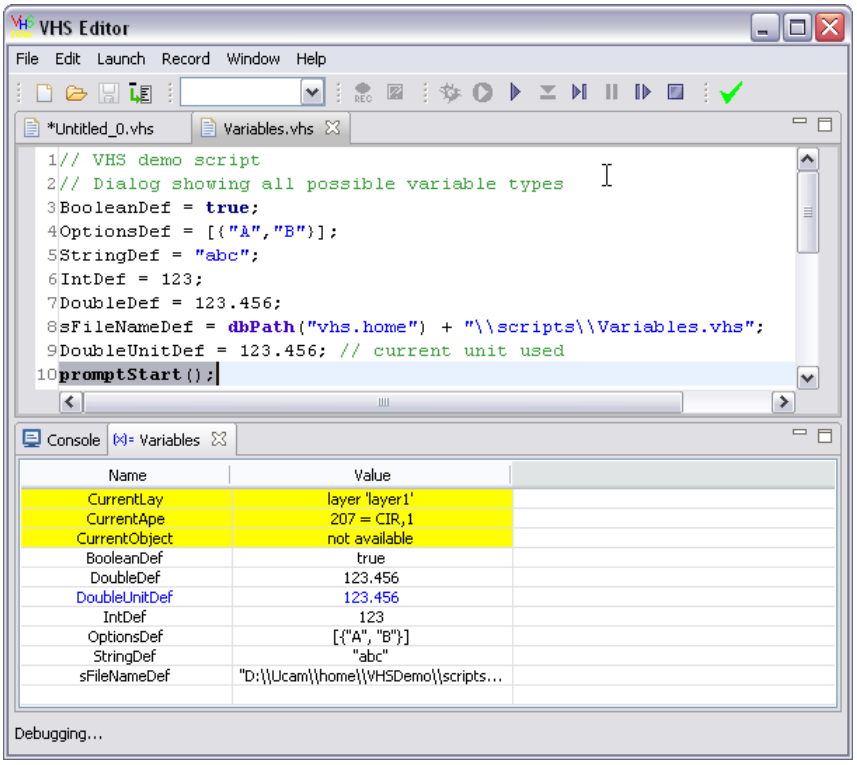
Choose how setInPlane is recorded

- **class** - plane, class, attach and index
- **subclass** - plane, class, subclass, attach and index
- **index** - plane and index
- **name** – plane and name

Default: class

20.2.5 Variables

The Variables window shows the current value of all pre-defined and customer defined Variables during debugging



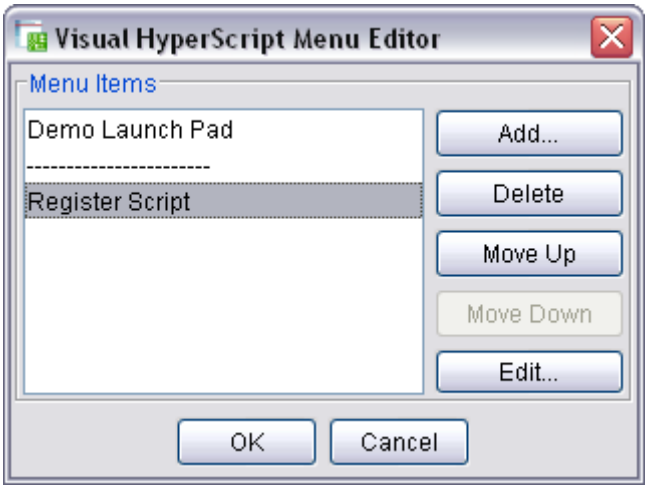
20.2.6 Console

The console window shows all feedback of the running script,
The results of your Print commands are also displayed in this window.

21 Visual HyperScript Menu Editor

21.1 Introduction

The Visual HyperScript Menu Editor is launched from the Hyperscript menu and allows you to add VHS Scripts and Launch Pads to the HyperScript menu.



21.2 Buttons

Add	Add a new menu item and specify its properties (see 21.3)
Delete	Delete the selected menu item
Move Up	Move the selected menu item up
Move Down	Move the selected menu item down
Edit	Edit the properties of the selected menu item see (see 21.4)
OK	Accept the current menu setup
Cancel	Cancel the Menu Editor

Note: Options are grayed out if the corresponding action is not applicable

21.3 Add menu Item

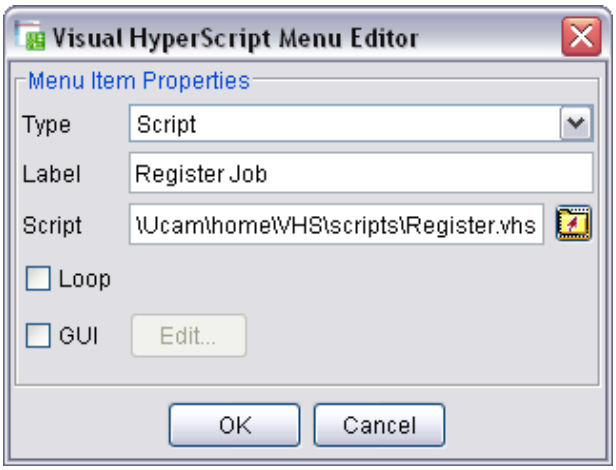
The Add button of the VHS Menu Editor can add:

- Scripts
- Editable Launch Pads

- Packaged Launch Pads (.hst)
- Separators

21.3.1 Add a Script to the HyperScript Menu

Click **ADD…** and set Type to **Script**

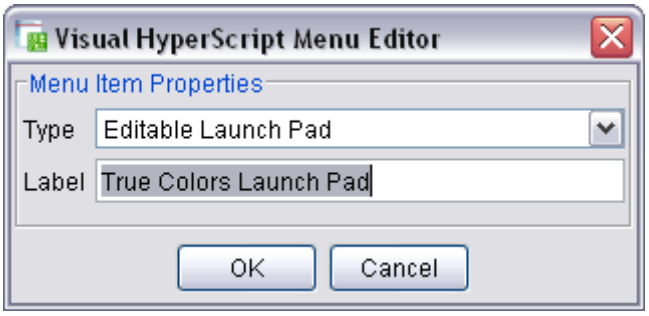


Label	Menu item text
Script	Path to the script (.vhs) file
Loop	Activate if script must be looped until Abort button clicked
GUI	Activate if script needs initial dialog
Edit	Create/Edit initial dialog for current script (see 17.2)
OK	Accept current menu item properties
Cancel	Cancel the Menu Editor

Note: Scripts can also be added to the HyperScript menu using the **addHyperScriptMenuItem** command

21.3.2 Add an Editable Launch Pad to the HyperScript Menu

Click **ADD…** and set Type to **Editable Launch Pad**

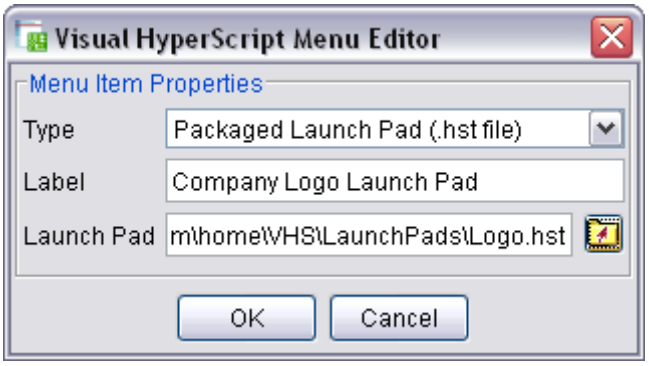


Label	Menu item text
OK	Accept current menu item properties
Cancel	Cancel the Menu Editor

Note: The resulting Editable Launch Pad is empty. Chapter 15.3 explains how to add Pages, Sections and Buttons to this Launch Pad

21.3.3 Add a Packaged Launch Pad to the HyperScript Menu

Click **ADD…** and set Type to **Packaged Launch Pad**

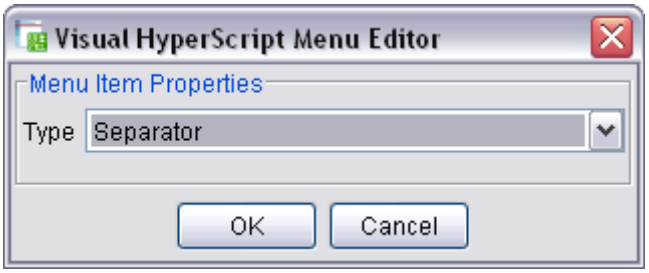


Label	Menu item text
Launch Pad	Path to the Packaged Launch Pad (.hst) file
OK	Accept current menu item properties
Cancel	Cancel the Menu Editor

Note: A Packaged Launch Pad (.hst) is an archive containing a dialog with all its scripts.This concept makes it possible to exchange Launch Pads with other UcamX users.

21.3.4 Add a Separator to the HyperScript Menu

Click **ADD…** and set Type to **Separator**



OK	Accept current menu item properties
Cancel	Cancel the Menu Editor

Note: Separators are used to divide the HyperScript menu into multiple sections e.g. scripts and launch pads.

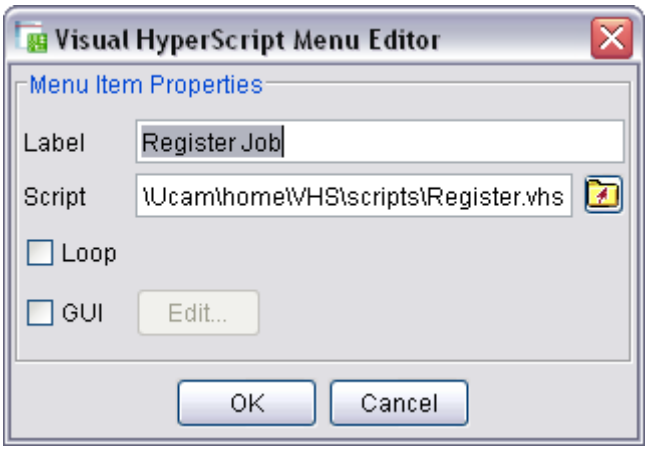
21.4 Edit HyperScript menu Items

The Edit button of the VHS Menu Editor edits the properties of:

- Scripts
- Editable Launch Pads
- Packaged Launch Pads (.hst)

21.4.1 Edit Script Properties

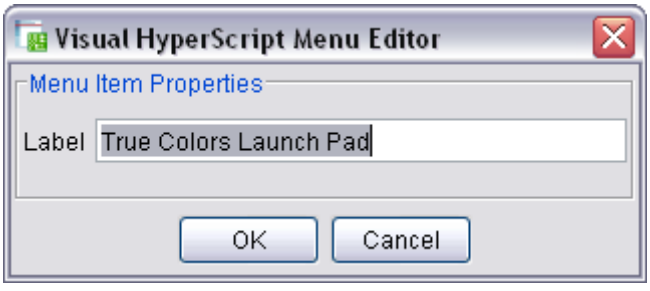
Select a script in the Menu Items list and click **Edit…**



Label	Name of the selected script
Script	Path to the script (.vhs) file
Loop	Activate if script must be looped until Abort button clicked
GUI	Activate if script needs Startup Dialog
Edit	Create/Edit initial dialog for current script (see 17.2)
OK	Accept current menu item properties
Cancel	Cancel the Menu Editor

21.4.2 Edit Editable Launch Pad Properties

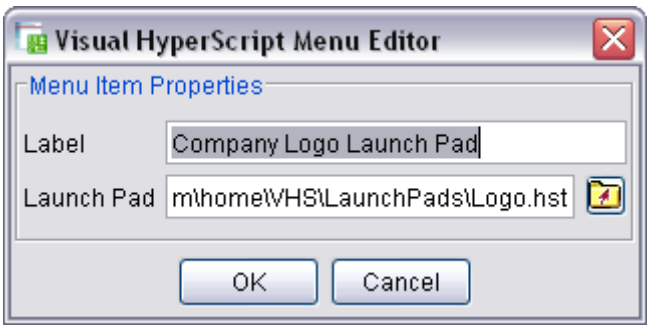
Select an Editable Launch Pad in the Menu Item list and click **Edit…**



Label	Name of the Editable Launch Pad
OK	Accept current menu item properties
Cancel	Cancel the Menu Editor

21.4.3 Edit Packaged Launch Pad Properties

Click **Edit…** and select a Packaged Launch Pad in the Menu Item list



Label	Name of the Packaged Launch Pad
Launch Pad	Path to the Packaged Launch Pad (.hst) file
OK	Accept current menu item properties
Cancel	Cancel the Menu Editor

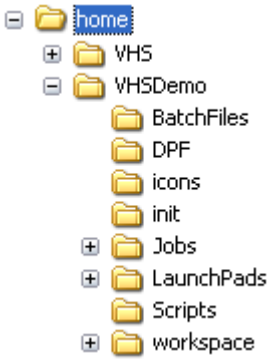
22 Running the Demo scripts

You can download a number of basic and advanced demo scripts that can be started from the Demo Launch Pad in the HyperScript menu.

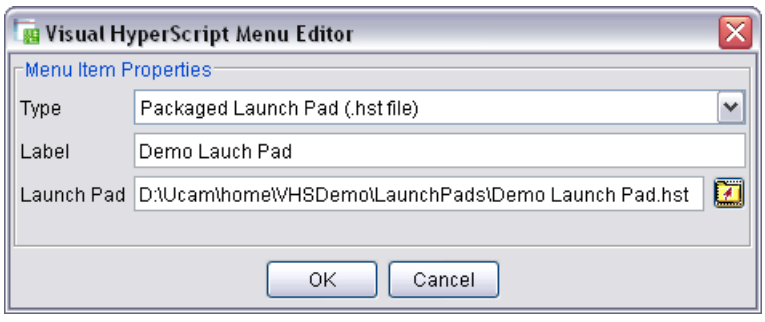
Run the scripts to get a first impression of Visual HyperScript. Open each script in the VHS Editor and examine the code to find out how a simple recorded script can be enhanced to fit your personal needs.

22.1 Initial Actions

1 Make a backup of your current scripts and Launch Pads.
2 Download the **VHSDemo.zip** archive from the Ucamco download page
3 Extract the VHSDemo.zip archive inside your HOME directory
Your HOME directory should look like this:



4 Set the ucam.db key **vhs.home** to the new HOME:/VHSDemo folder
5 Restart UcamX
6 Use the VHS Menu Editor to add the Demo Packaged Launch Pad to the HyperScript menu



7 Start the Demo Launch Pad from the HyperScript menu
8 Scale the Demo Launch Pad to full size
9 Drag and dock in to a location on the screen where is can stay
10 Save the Workspave as “VHS Demo”

Note: Some interactive scripts from the Demo Launch Pad use a dialog window to show/enter info. The Registration Script (see 22.3 steps 3 to 5) explains how to fix the location of this dialog on your screen.

22.2 Welcome

This script creates a job in your temporary storage directory (set your TEMP environment Variable if it is not yet available), creates a layer and flashes a welcome message using a text aperture.

Actions

1 Load the “VHS Demo” Workspace

2 Click the **Show** button in the Welcome section of the Demo Launch Pad

Script name: “Welcome.vhs”

```
// VHS demo script
// Creates a job and uses a text aperture to flash a welcome text
//
// Create a job in your temp directory
Stemp = envString("TEMP"); // get location of temp storage
newJob(Stemp + "VHSDemo\\Jobs\\Welcome", "Welcome");
//
// Create a layer
createExtra("top");
//
// Create a text aperture with the welcome message
apeCreateText(3, 5.0, "Welcome to Visual HyperScript");
//
// Insert one Flash using the text aperture
insertFlash(Point(0.0, 0.0), 0, "select");
//
// Zoom
viewZoom("total");
```

22.3 Registration

22.3.1 Recorded

The Register script is a simple recorded script that registers all copper layers of a job to the drill layer

Actions

1 Open the Register job from HOME:/VHSDemo/Jobs/Register
Note how top mask, drill and silk screen are not registered to the copper layers

2 Load the “VHS Demo” Workspace

3 Click the **Recorded** button in the Registration section of the Demo Launch Pad
The layers will be registered

Note: The top silk layer can not register because it does not contain flashes

Script name: “Register.vhs”

```
// VHS demo script
// This ia a recorded script
//
setMode("selall", "mm", "no");
// Register Top Copper to Drill
hideAll();
activate("all", "", false);
setInPlane(1, "layer", null, "", 1);
setInPlane(3, "drill", null, "", 1);
activate("drill", null, 1, true);
registerLayers();
viewZoomTotal();
// Register Top Mask to Top Copper
hideAll();
setInPlane(1, "extra", null, "", 2);
setInPlane(3, "layer", null, "", 1);
activate("layer", null, 1, true);
registerLayers();
// Register Bottom Copper and Inners to Drill
hideAll();
activate("all", "", false);
setInPlane(1, "layer", null, "", 6);
setInPlane(4, "layer", null, "", 5);
setInPlane(4, "layer", null, "", 4);
setInPlane(4, "layer", null, "", 3);
setInPlane(4, "layer", null, "", 2);
activate("layer", null, 2, true);
activate("layer", null, 3, true);
activate("layer", null, 4, true);
activate("layer", null, 5, true);
setInPlane(3, "drill", null, "", 1);
activate("drill", null, 1, true);
registerLayers();
// Register Bottom Mask to Bottom Copper
hideAll();
activate("all", "", false);
setInPlane(1, "extra", null, "", 4);
setInPlane(3, "layer", null, "", 6);
activate("layer", null, 6, true);
registerLayers();
// Show result
hideAll();
activate("all", "", false);
setInPlane(1, "layer", null, "", 1);
setInPlane(2, "extra", null, "", 2);
setInPlane(3, "drill", null, "", 1);
viewZoomTotal();
```

22.3.2 With Startup Dialog

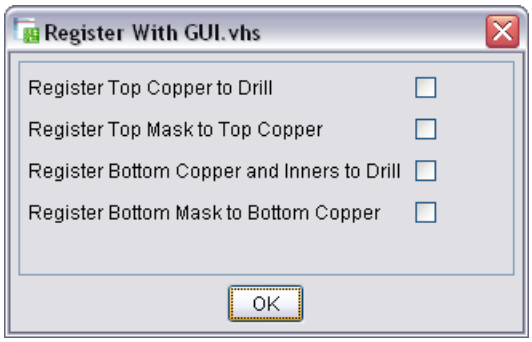
The “Register with Startup Dialog” script adds a Startup Dialog to allow the operator to select the registration steps.

Actions

1 Open the Multi job from HOME:/VHSDemo/Jobs/Register

2 Load the “VHS Demo” Workspace

3 Click the **With Startup Dialog** button in the Registration section of the Demo Launch Pad



Note: This startup dialog was created when the **With Startup Dialog** button was added to the Launch Pad. The GUI toggle was activated and 4 Booleans (bSelect 1 to 4) were added (see 15.3.3 and 17.2.3)

4 Move the dialog to a location on the screen where is can stay

5 Use **Workspaces > Save** to store the location and size of the registration dialog in the VHS Demo workspace

6 Activate one or more registration entries

7 Click **OK**
Repeat steps 6 and 7 until the registration process is completed.

Note: As long as the dialog stays on the screen you could open other jobs with a similar layer structure and perform the registration

Script name: “Register with Startup Dialog.vhs”

```
// VHS demo script
// Recorded Register script, completed with
// comments, Variables, decisions and print commands
// Parameters (bSelect 1 to 4) are set in the Startup Dialog
//
setMode("selall", "mm", "no");
hideAll();
activate("all", "", false);
//
if (bSelect1 == true) {
  print(" 1. Register top copper to drill");
  setInPlane(1, "layer", null, "", 1);
  setInPlane(3, "drill", null, "", 1);
  activate("drill", null, 1, true);
  registerLayers();
}
if (bSelect2 == true) {
  print(" 2. Register top mask to top copper");
  setInPlane(1, "extra", null, "", 2);
  setInPlane(3, "layer", null, "", 1);
  activate("layer", null, 1, true);
  registerLayers();
}
```

```
if (bSelect3 == true) {
  print(" 3. Register bottom copper and inners to drill");
  setInPlane(1, "layer", null, "", 6);
  setInPlane(4, "layer", null, "", 5);
  setInPlane(4, "layer", null, "", 4);
  setInPlane(4, "layer", null, "", 3);
  setInPlane(4, "layer", null, "", 2);
  activate("layer", null, 2, true);
  activate("layer", null, 3, true);
  activate("layer", null, 4, true);
  activate("layer", null, 5, true);
  setInPlane(3, "drill", null, "", 1);
  activate("drill", null, 1, true);
  registerLayers();
}
if (bSelect4 == true) {
  print(" 4. Register bottom mask to bottom copper");
  hideAll();
  activate("all", "", false);
  setInPlane(1, "extra", null, "", 4);
  setInPlane(3, "layer", null, "", 6);
  activate("layer", null, 6, true);
  registerLayers();
}
// Display registered layers
setInPlane(1, "layer", null, "", 1);
setInPlane(2, "extra", null, "", 2);
setInPlane(3, "drill", null, "", 1);
setInPlane(5, "extra", null, "", 1);
viewZoomTotal();
//
```

Note: The “print” command is a BeanShell command and therefore has no color coding

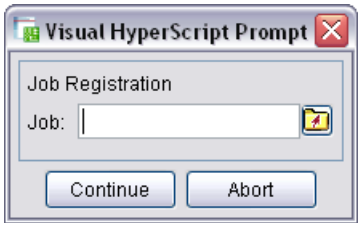
22.3.3 Using external script

The “Register with runFile” script uses a simple dialog to request the jobname. The registration is done by including a call to the Register.vhs script

Actions

1 Load the “VHS Demo” Workspace

2 Click the **With runFile** button in the Demo Launch Pad



3 Select the Multi job in your HOME/VHSDemo/Jobs/Register directory

4 Click **Continue**

Script name: “Register with runFile.vhs”

```
// VHS Demo script
// Interactive registration script using the Register.vhs script
promptStart();
  promptLabel("Job Registration");
  SjobName = promptFileName("Job: ", "");
promptEnd();
openJob(SjobName);
sVHSHome = dbPath("vhs.home");
runFile(sVHSHome + "\\Scripts\\Register.vhs"); // call external script
print("Job "+ SjobName + " registered");
```

22.4 Insert Ucamco Logo

The insert logo script loads a pre defined Ucamco logo in dpf format into the aperture list of the layer in plane 1 (red) and flashes it at coordinates 0,0.
It is up to the operator to move the logo to a suitable place on the layer.
There is a positive and a reverse version of the logo.

Actions

1 Open a job (e.g. HOME:/VHSDemo/Jobs/Multi/Multi.job)
2 Load the “VHS Demo” Workspace
3 Display only the top layer in plane 1 (red)
4 Click the **Positive** button in the Insert Logo section of the Demo Launch Pad

Script name: “Insert Ucamco Logo.vhs”

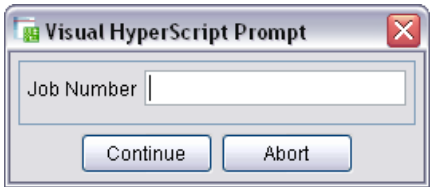
```
// VHS demo script
// This script loads a logo and flashes it at the clicked location
sVHSHome = dbPath("vhs.home");// get ucam.db key
loadApertures(sVHSHome + "\\DPF\\Ucamco.dpf");
pVar = pickPoint("Click on a point");  //Get Point coordinates
insertFlash(pVar, 0, "select");
viewZoom("total");
```

22.5 Rename Layers

This script renames all layer names of an imported customer job to company standard names (e.g. jobnumber_layerinfo). The console window shows a log of all renamed layers.

Actions

1 Open the multi job from HOME:/VHSDemo/Jobs/Rename
2 Load the “VHS Demo” Workspace
3 Click the **Do It** button in the **Rename Layers** section of the Demo Launch Pad



4 Enter a job number (can be a mix of numbers and characters)
5 Click **Continue**

Script name: “Rename Layers.vhs”

```
  // VHS demo script
  // Rename layers to company standard naming
  print("Rename Layers");
  promptStart();
    JobNumber = promptString("Job Number", "");
  promptEnd();
  // Dedicated loop to count the number of signal layers
  iSignalCount = 0;
  forEachSignal();{
    iSignalCount++;
  }
  // Rename signal layers
  forEachSignal() {
    if (layIndex() == 1) {
        print("Rename " + layName() + " to " + JobNumber +
"_Copper");
        setInPlane(1, "layer", null, "", layIndex(), true);
        modifyLayer(JobNumber +
"_Copper","","layer",laySubClass(),layIndex(),false,0.0,layReadable(),
"",0.0);
    }
    else if (layIndex() == iSignalCount){
        print("Rename " + layName() + " to " + JobNumber + "_Solder");
        setInPlane(1, "layer", null, "", layIndex(), true);
        modifyLayer(JobNumber +
"_Solder","","layer",laySubClass(),layIndex(),false,0.0,layReadable(),
"",0.0);
    }
    else {
        print("Rename " + layName() + " to " + JobNumber + "_L" +
layIndex());
        setInPlane(1, "layer", null, "", layIndex(), true);
        modifyLayer(JobNumber + "_L" +
layIndex(),"","layer",laySubClass(),layIndex(),false,0.0,layReadable()
,"",0.0);
    }
  }
  // Rename extra layers
  forEachExtra() {
        print("Rename " + layName() + " to " + JobNumber + "_" +
laySubClass() + "_" + layAttach());
        setInPlane(1, "extra", null, "", layIndex(), true);
        modifyExtra(JobNumber + "_" + laySubClass() + "_" +
layAttach(), "", "extra", laySubClass(), layAttach(), layIndex(),
false, "");
  }
```

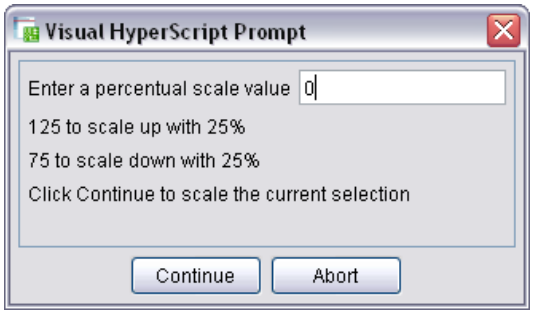
```
  // Rename drill layers
  forEachDrill() {
        print("Rename " + layName() + " to " + JobNumber + "_" +
laySubClass());
        setInPlane(1, "drill", null, "", layIndex(), true);
        modifyDrill(JobNumber + "_" + laySubClass(), "", "drill",
laySubClass(), layFrom(), layTo(), 0.0);
  }
```

22.6 Percentual Scaling of Objects

22.6.1 Using an Interactive Script

The percentual scale function combines the UcamX scale and distort functions into a new interactive function.

1 Actions
2 Open any job
3 Load the “VHS Demo” Workspace
4 Select a flashed object
5 Click the **Interactive** button in the Percentual Scale section of the Demo Launch Pad



6 Enter a scale value
7 Click **Continue**

Script name: “scale_pct.vhs”

```
// VHS demo script
// Percentual scale script using scale and distort functions
promptStart();
  Ipct = promptInteger("Enter a percentual scale value", 0);
  promptLabel("125 to scale up with 25%");
  promptLabel("75 to scale down with 25%");
  promptLabel("Click Continue to scale the current selection");
promptEnd();
double Dpct = Ipct / 100.0;  // java calculation
scale(Dpct, false);
double Ddistort = 1.0 / Dpct;  // java calculation
distort(Ddistort, Ddistort);
```

Note: This script uses pure java instructions to perform the calculations. Variables in pure java instructions need to be typed.

22.6.2 Using a Custom Function

If we always want to scale the selected apertures with a fixed value, than we could replace the script by a custom command. Custom commands are placed in a directory set by the **vhs.init.scripts** ucam.db key. For this demo we are setting its value to HOME/VHSDemo/init

The init directory already has the following **usr_scale_pct.vhs** script

```
/**
* Percentual scale of selected apertures
*
* @param iPct Percentual scale value
*/

void usr_scale_pct(double iPct){
  double dPct = iPct / 100.0;
  System.out.println("Scale = " + dPct);
  scale(dPct, false);
  double dDistort = 1.0 / dPct;
  System.out.println("Distort = " + dDistort);
  distort(dDistort, dDistort);

}
```

Now we can use the use the usr_scale_pct command in our scripts.

Actions

1 Open any job
2 Select a flashed object
3 Load the “VHS Demo” Workspace
4 Click the **Custom** button in the Percentual Scale section of the Demo Launch Pad

Script name: “scale_pct Custom.vhs”

```
// VHS demo script
// Percentual scale script using custom function
usr_scale_pct(125);
print("All selected apertures are scaled by 125%");
```

Note: Custom commands have no color coding

22.7 Set Top/Bottom View to True Colors

This script shows the top and bottom side of a printed circuit in “True Colors”.

To show true colors, you will need:

- A job with an outline layer using a contour aperture
- A special ucamcolor.db file

Actions

1 Make a backup of the ucamcolor.db file in the HOME directory (if existing)
2 Copy the True Colors ucamcolor.db file to your HOME directory
3 Open the multi job from HOME:/VHSDemo/Jobs/Multi final

4 Load the “VHS Demo” Workspace

5 Click the **Top View** button in the True Colors section of the Demo Launch Pad to see a top view of the printed circuit board in true colors

6 Click the **Bottom View** button in the True Colors section of the Demo Launch Pad to see the bottom view of the printed circuit board in true colors

Script name: “True Colors Top.vhs”

```
// VHS demo script
// Displays the top side of a pcb job in realistic colors
hideAll();
activate("all", "", false);
setInPlane(1, "layer", "outer", "top", 1);
setInPlane(2, "extra", "mask", "top", 1);
setInPlane(3, "drill", "", "", 1);
setInPlane(4, "extra", "outline", "", 1);
setInPlane(5, "extra", "silk", "top", 1);
viewZoomTotal();
```

Note: The recording of this script was done with Record mode setInPlane = subclass

Script name: “True Colors Bottom.vhs”

```
// VHS demo script
// Displays the bottom side of a pcb job in realistic colors
hideAll();
activate("all", "", false);
iSignalCount = 0;
forEachSignal();{
  iSignalCount++;
}
setInPlane(1, "layer", "outer", "bottom", iSignalCount);
setInPlane(2, "extra", "mask", "bottom", 1);
setInPlane(3, "drill", "", "", 1);
setInPlane(4, "extra", "outline", "", 1);
setInPlane(5, "extra", "silk", "bottom", 1);
viewZoomTotal();
```

Note: a loop was added to count the number of copper layers

22.8 Drill Map with Dimensions

This script creates a new drill map with outline and adds dimensioning info.

Actions

1 Open the multi job from HOME:/VHSDemo/Jobs/Multi final

2 Load the “VHS Demo” Workspace

3 Click the **Create** button in the **Drill Map with Dimensions** section of the Demo Launch Pad

Script name: “Drill Map with Dimensions.vhs”

```
// VHS demo script
// This script creates a new drill map with outline and dimensions
setMode("selall", "mm", "no");
hideAll();
activate("all", "", false);

// Copy outline to new MyDrillMap layer
setInPlane(1, "OUTLINE");
copyToClipboard();
createExtra("MyDrillMap", "help", "none");
pasteFromClipboard();
// convert contour to circle if needed
setCurrentAperture(1);
editAperture(apeNumber(), "", "CIR,0.1");
viewZoomTotal();

//Create drillmap and copy to MyDrillMap
setInPlane(1, "drill", "", "", 1);
//Get path to your symbol file
sSymFile = dbPath("drilltoolmanager.symbol.file");
//This script only replaces the first 10 tools
drillMapReplace(sSymFile, [{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}]);
setInPlane(1, "drill_drimap");
copyToClipboard();
setInPlane(1, "MyDrillMap");
pasteFromClipboard();
viewZoomTotal();

//Add dimensioning
setCurrentAperture(1);
selectAperture();
R = apeEnclosingBox();
apeCreateCircle(999, 0.1);
dimensioning("distance", 1.0, [{Point(R.xmin, R.ymax), Point(R.xmax,
R.ymax), Point(R.xmax-R.xmin, R.ymax+10)}], true, 2.0, 3.0, 5.0, 5.0,
5.0, 5.0, 5.0, 0.0, 0.0, 0.0, 3, false, false, null,0 ,0 ,"");
dimensioning("distance", 1.0, [{Point(R.xmin, R.ymin), Point(R.xmin,
R.ymax), Point(R.xmin-10, R.ymax-R.ymin)}], true, 2.0, 3.0, 5.0, 5.0,
5.0, 5.0, 5.0, 0.0, 0.0, 0.0, 3, false, false, null,0 ,0 ,"");
activate("all", "", false);
viewZoomTotal();
```

22.9 Showing All “Promptable” Variable Types

This script uses prompt commands to show a dialog box with all possible Variable types. Examine the script to find out how to initialize and display Variables in HyperScript.

Note: This dialog allows you to load/save/delete a parameter set (see 11.4.1)

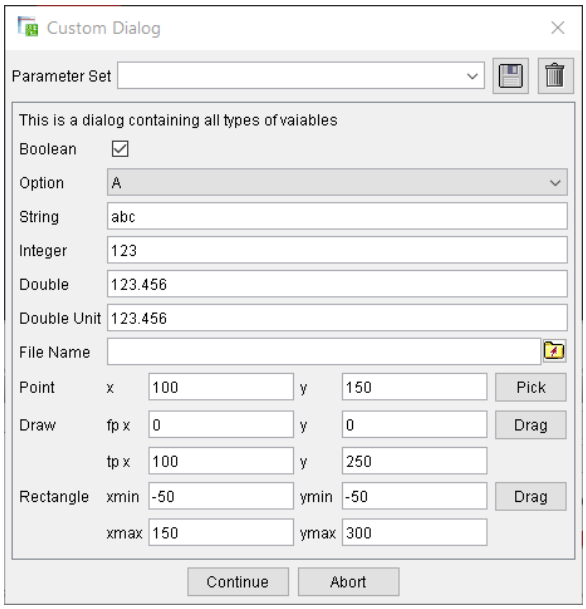
Actions

1 Copy the saved parameterset VHSDemo\ParameterSet\ParameterSet.cfg to your HOME directory

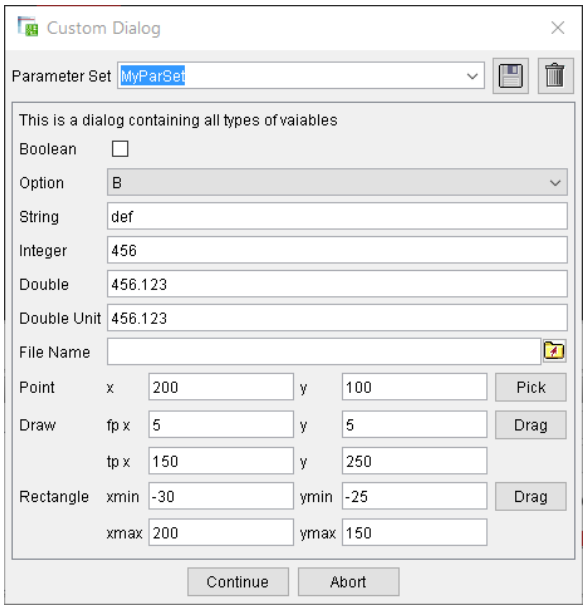
2 Restart UcamX

3 Load the “VHS Demo” Workspace

4 Click the **Show** button in the Variables section of the Demo Launch Pad



5 Click on the select arrow and select the saved parameter set “MyParSet” See how the variables receive the values of the saved parameter set



Script name: “Variables with Parameter Set.vhs”

```
// VHS demo script
// This script shows a dialog containing all variable types
// The dialog uses a saved parameter set
print("Dialog using pre-defined variables");
sParSet = "MyParSet";
boolean BooleanDef = true;
```

```
String[] OptionsDef = {"A","B"};
String StringDef = "abc";
int IntDef = 123;
double DoubleDef = 123.456;
double DoubleUnitDef = 123.456; //unit is mm
promptStart(sParSet, "Custom Dialog");
promptLabel("This is a dialog with all types of variables");
  Boolean BooleanVar = prompt("Boolean", BooleanDef);
  String OptVar = prompt("Option", OptionsDef, "A");
  String StringVar = prompt("String", StringDef);
  int IntVar = prompt("Integer", IntDef);
  double DoubleVar = prompt("Double", DoubleDef);
  double DoubleUnitVar = prompt("Double Unit", DoubleUnitDef, "mm");
  Point flashPt = promptPoint("Point", Point(100, 150)) ;
  Line line = promptLine("Draw", Line(0, 0, 100, 250));
  Rectangle rect = promptRectangle("Rectangle", Rectangle(-50, 150,
  -50, 300));
promptEnd();
```

22.10 Looping a Script

This script allows you to replace all occurrences of a selected painted pad by a standard aperture or by a complex.

This script was originally implemented as a Learn.

Actions

1 Open the Painted job from HOME:/VHSDemo/Jobs/Models

2 Load the “VHS Demo” Workspace

3 Click the  button in the **Replace Painted Pads** section

4 Drag a rectangle around a painted pad. When the mouse is released, the object inside the rectangular area (and al similar objects) are replaced by a standard or a complex aperture.
As this script is looping, the dialog box stays on the screen and you are ready to repeat this action until all painted pads are replaced

5 Click Abort button (or **[Esc]** key) to terminate the script

Script name: “Models Loop.vhs”

```
// VHS demo script
// This script replaces all ocurrences of a selected
// painted pad by a standard or complex aperture
// This script needs the Loop flag to be set
setMode("selall", "mm", "no");
selectWindow("+", dragRectangle("Select a painted pad"), "i",
null, null);
modelsDefineSelections();
// Use return code of modelsCreateStandard (true or false) to
// choose conversion to standard or complex aperture
if(!modelsCreateStandard(0.01)) {
  modelsCreateComplex();
  }
modelsSelect(0.01, 0.01);
modelsReplace(0.01, 0.01);
```

```
print(apeInfo());
selectAll("-");
```

Note: You cannot use the zoom Window function in a loop. Use the mouse wheel and buttons to zoom in and out on the cursor location

22.11 Running a Standalone Script

This script will convert all copper layers (dpf) of a UcamX job to extended Gerber (RS-274X) without opening the UcamX application.

Script name “exp274x.vhs

```
// This script converts copper layers (dpf) of a UcamX job to
// extended Gerber (RS-274X) without opening the UcamX application.

openJob(param1);
hideAll();
activate("all", "", false);
activate("lay", "", true);
output274x("C:\\ucamco\\UcamX2016\\env\\dat\\Cad");
```

Note: Change the path if UcamX is installed in a different directory

Batch file name “dpfTo274x.bat”

```
echo on
set str= %1
echo.%str%
set str=%str:\=\\%
echo.%str%
for /f "useback tokens=*" %%a in ('%str%') do set str=%%~a
echo.%str%
VHSExecuter.bat %HOME%\VHSDemo\scripts\exp274x.vhs \""%str%"\"
```

Note: Make sure that your **Path** environment Variable includes a path to the VHSExecuter batch file.

Actions:

1 Close UcamX

2 Use Windows Explorer to browse to a folder containing a UcamX job (job file and dpf layers)

3 Right click on the job file and select “open with”

4 Browse to **…\Home\VHSDemo\batchfiles\pdfTo274x.bat** and select it

All layers of the selected job will be converted to RS-274X.

22.12 Calculate & Export BoardSnapshot data

This script uses standard UcamX functionality to calculate & export BoardSnapshot data to a text file. It also creates (at the same location) a VHS file with all BoardSnapshot data formatted as VHS Variables. This file can be imported into VHS using the BoardSnapImport script (see 22.13).

Tip: Check this script out to learn the java commands for handling files.

Actions

1 Set the ucam.db key bsnap.txt to %p\%n_bsnap.txt

2 To enable Boardsnapshot Product Information output edit your UcamX2016\env\dat\ucam.properties file, look for uibsnap parameters and add uibsnap_text_tb.set: true

3 Restart UcamX

4 Open a job

5 Load the “VHS Demo” Workspace

6 Click the **Calculate** & **Export** button in the BoardSnapshot section of the VHSDemo Launch Pad

The script will show all parameters that are added to your export file.

Script name “BoardSnapExport.vhs”

```
// VHS demo script
// This script
// - Calculates new Boardsnap data using a filter
// - Prints calculated BS parameters
// - Exports BS parameters to an importable vhs file
// Initial actions:
// Set Ucam.db parameter bsnap.text to %p\%n_bsnap.txt
// To set text output default edit UcamX2016\env\dat\ucam.properties
// look for uibsnap parameters and add "uibsnap_text_tb.set: true"
// (do not enter the quotes)
// Restart UcamX
// Open a job
// Launch Board Snapshot and specify Setup values

setMode("selall", "mm", "no");
// Set source and export file
sSource = jobPath() + jobName() + "\\" + jobName() + "_bsnap.txt";
sTarget = jobPath() + jobName() + "\\" + jobName() +
"_bsnap.txt.vhs";
activate("all", "", true);
netlistBuild("job");
// Calculate Board Snapshot Data
boardSnapshot(false, "", true, sSource);
// Convert Board Snapshot Data
sQuote = "\"";
File file = new File(sSource);
FileInputStream fis = null;
DataInputStream dis = null;
try {
  fis = new FileInputStream(file);
  // BufferedInputStream is added for fast reading
  bis = new BufferedInputStream(fis);
  dis = new DataInputStream(bis);
  FileWriter fstream = new FileWriter(sTarget);
```

```
  BufferedWriter out = new BufferedWriter(fstream);
  // dis.available() returns 0 if the file has no more lines
  while (dis.available() != 0) {
   sLine = dis.readLine();
   iSplit = sLine.indexOf(":");
   sVariable = sLine.substring(0, iSplit);
   sValue = sLine.substring(iSplit+2,sLine.length());
   sValue = sValue.replace("\\","/");
   sVariable = sVariable.replace(".","_");
   // print variable to console
   print(sVariable + " = " + sValue);
   // export variable to file
   out.write(sVariable + " = " + sQuote + sValue + sQuote + ";\n");
  }
  print("------------------");
  print("Print/Export ready");
  // dispose all the resources after using them
  out.close();
  fis.close();
  bis.close();
  dis.close();
}
catch (FileNotFoundException e) {
  e.printStackTrace();
}
catch (IOException e) {
  e.printStackTrace();
}
```

22.13 Import BoardSnapshot data

This script allows you to import the BoardSnapshot parameter file from the previous demo into UcamX.

Actions

1 Load the “VHS Demo” Workspace

2 Click the **Import** button in the BoardSnapshot section

Note: This script contains a dummy print instruction to print 1 variable.
After the import all parameters (300+) are available as VHS Variables

Script name “BoardSnapImport.vhs”

```
// VHS demo script
//This script imports Boardsnap parameters in vhs format
//created with the BoardSnapExport script

//Get default location for BSvhs file
sBSvhs = jobPath() + jobName() + "\\" + jobName() + "_bsnap.txt.vhs";

//Select BSvhs file
promptStart();
  sBSvhs = promptFileName("Select Converted BoardSnap Export file:",
sBSvhs);
promptEnd();
```

```
//Import BS parameters in UcamX as variables
importFile(sBSvhs);

//Dummy script to show BS variable(s)
print("job_copper_top: " + job_copper_top);
```

23 Glossary

.bsh	Beanshell Script file
.vhs	Visual HyperScript Script file
.hst	Packaged Launch Pad file (zipped dialogs & scripts)
Visual HyperScript	Scripting language for UcamX
Dialog Box	Window used for interaction with the user
Launch Pad	Dialog Box containing tabbed windows, sections & buttons
Editable Launch Pad	Launch Pad for personal use
Packaged Launch Pad	Launch Pad to be used for export to other UcamX user(s) (.hst)
Page	Tabbed Window inside a Launch Pad
Section	Outlined part of a Page on a Launch Pad
Startup Dialog	Optional initial dialog displayed when launching a script
Button	Clickable object on a Launch Pad Section or on an interactive dialog
ETSCAM_DAT	System Variable referring to folder with UcamX system settings
ETSCAM_CFG	System Variable referring to folder with company wide settings
HOME	System Variable referring to folder with personal settings
Editor Area	Part of VHS Script Editor where scripts are entered/displayed
Tab Page	Method of displaying multiple open files in the Editor Area