

Kid's Programming Language

Teacher's User Manual

From **Morrison Schwartz**

Web site: www.kidsprogramminglanguage.com

Kid's Programming Language Teachers' User Manual

Table of Contents

Introduction: Kid's Programming Language	4
What is Kid's Programming Language (KPL)?.....	4
Why should I learn and use KPL? Because KPL is:.....	4
Section One: The Integrated Development Environment ..	5
KPL's Menu System	5
KPL's Toolbar	11
KPL's Code Editor.....	13
Explorer Pane.....	18
Messages Pane	19
Section Two: KPL's Programming Language	20
Identifiers	20
Data Types	20
Variables	22
Key Words	22
Comments	23
Loops.....	23
Decisions and Branching	25
Functions and Methods	25
Section Three: Learning Program Examples	27
Learning Program 001 – Hello World!	27
Learning Program 002 – Introduction to Sprite Graphics	29
Learning Program 003 – Arrays, and Sprite Animation Timelines	30
Learning Program 004 – While Loops and If...Then...Else logic	32
Learning Program 005 – Controlling a game with Keyboard input	34
Learning Program 006 – Using Structures, and 2-player gaming	37
Section Four: Debugging KPL programs	41
Section Five: Resources for KPL programmers	43
Section Six: System Methods available in KPL.....	44
MATH FUNCTIONS	44
DATA FUNCTIONS	45
DATE FUNCTIONS	46
KEYBOARD FUNCTIONS	48
CONSOLE FUNCTIONS.....	48
CONSOLE METHODS.....	48
SCREEN FUNCTIONS.....	49
SCREEN METHODS	49
PEN METHODS.....	51

SPRITE FUNCTIONS.....	52
SPRITE METHODS.....	53
SOUND METHODS	55
TRACE AND RUNTIME METHODS	55

Introduction: Kid's Programming Language

What is Kid's Programming Language (KPL)?

- KPL is a simplified but modern programming language and programming environment
- KPL was created for teaching beginners how to program
- KPL offers lots of sample KPL programs and games as educational and fun examples
- KPL emphasizes graphics, sprites, sounds and game development, because these are the best ways to get and keep a beginner's interest long enough for them to learn to program
- KPL is available in 10 international languages, thanks to volunteer translators, and more translations are happening all the time
- KPL is designed to help a beginner 'graduate' to Visual Studio or Eclipse – the current state of the art computer programming environments

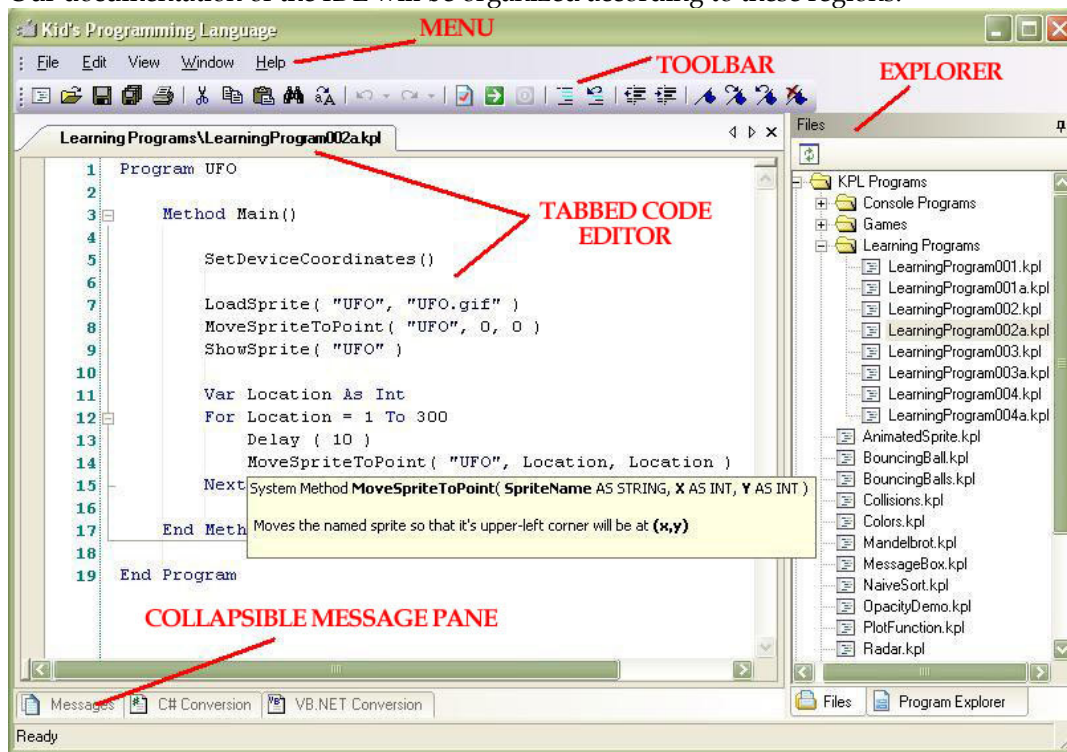
Why should I learn and use KPL? Because KPL is:

- **Fun** – And learning is best when learning is fun!
- **Accessible** – Computer programming can be a complicated science, but KPL is carefully designed to make it easy to get started
- **Engaging** – Kids know and enjoy computer graphics and games, and there is no better way to capture and keep their interest in programming than by making these easy to do
- **Simple** – Computer Science has a natural tendency toward increasing power and complexity. KPL pushes very purposefully against that tendency, because as complex as it might get later, CS has to *start simply*, or else beginners will never get started
- **Rewarding** – Quick results reward beginners for their programming
- **Highly leveraged** – A small amount of KPL code delivers a large amount of functionality
- **Progressive** – Beginners must be able to advance steadily from the very beginnings of programming, to creating fully functional and distributable programs
- **Preparatory** – Beginners who are interested enough in computer science to graduate beyond KPL are well prepared to use the current state of the art programming languages and environments
- **Modern** – Unlike older beginner languages, KPL's technology and design are modern. Older languages look and feel dated to users, and their usability is reduced because they are inconsistent with current software design standards and conventions
- **Publishable** – One of the most rewarding experiences for me as a computer scientist is to know that other people are happily using programs I wrote. The ability for beginners to "publish" their KPL programs for others to read and run will offer beginners a taste of that most rewarding experience, in KPL version 2, which will be available within a few months
- **Technically state-of-the-art** – KPL is based on .NET, which itself was inspired by Java and the Java Runtime. Besides the preparatory advantages mentioned above, this means that KPL's life cycle of technical relevance is starting from a pretty modern position, so your investment in KPL will remain relevant for far longer than older languages
- **International** – We are working hard to make KPL available around the world, using local language, character set and cultural conventions. Ten international languages are already available! One secondary result of this is that the educational content and KPL community will help expand horizons and imaginations

Section One: The Integrated Development Environment

KPL offers a simple but usable Integrated Development Environment which is designed to help a beginner 'graduate' to Visual Studio.NET, Eclipse, or another commercial development environment.

Our documentation of the IDE will be organized according to these regions:



KPL's Menu System

File Menu	Edit Menu	View Menu	Window Menu	Help Menu
New Program	Cut	Messages	1 First open program	About KPL
Open Program	Copy	Files Explorer	2 Second open program	Technical Support
Save Program	Paste	Program Explorer	3 Third open program	
Save As...	Undo	C# Conversion	Etc...	
Save All	Redo	VB.NET Conversion		
Close Program	Select All			
Print Program	Find			
Exit KPL	Replace			

File Menu

The File menu can be accessed by holding the Alt key down and pressing the F key (Alt+F). Once the File menu is displayed, items on it can be selected by pressing the Key which is underlined in the menu selection. For example, pressing Alt+F then N will open a new KPL program as described below. This is Windows-standard behavior.

New Program (Ctrl+N) – Creates a new template KPL program, which is properly structured, compiles and runs – though it does not do anything functional until further code is provided. **Ctrl-N** indicates this function can also be invoked by holding the Ctrl key and typing N.

Open Program (Ctrl+O) – Launches a dialog allowing the user to browse to and open a KPL program from disk – either local disk or networked, of course. **Ctrl-O** indicates this function can also be invoked by holding the Ctrl key and typing O.

Save Program (Ctrl+S) – Saves any code changes which have been made to the currently selected KPL program.

Save As... – Opens a dialog which allows the currently selected KPL program to be saved with a different name or in a different location on disk.

Save All (Ctrl+Shift+S) – Saves any code changes which have been made to **all** open KPL programs.

Close Program (Ctrl+F4) – Closes the currently selected KPL program file. If changes have been made to that file, a dialog will allow you to save those changes before the file is closed, if you wish to save them.

Print Program (Ctrl+P) – Launches a Windows-standard Print dialog allowing you to print the currently selected KPL program.

Exit KPL (Alt+F4) – Exits and closes the KPL development environment. If changes have been made to any open KPL program files, warning dialogs will allow you to save those changes before exiting KPL, if you wish to save them.

Edit Menu

The Edit menu can be accessed by holding the Alt key down and pressing the E key (Alt+E). Once the Edit menu is displayed, items on it can be selected by pressing the Key which is underlined in the menu selection. Each of the functions on this menu applies to the currently selected KPL program file in the tabbed code editor. The more usual way to access these functions, of course, is to use the keyboard shortcuts associated with each, as shown below. This is Windows-standard behavior.

Cut (Ctrl+X) – Cuts the currently selected text from the currently selected KPL program file. Cut text is placed on the Windows Clipboard.

Copy (Ctrl+C) – Copies the currently selected text from the currently selected KPL program file, without deleting it. Copied text is placed on the Windows Clipboard.

Paste (Ctrl+V) – Pastes the Windows Clipboard contents into the currently selected KPL program file at the cursor location. Pasted text is still available on the Windows Clipboard.

Undo (Ctrl+Z) – Reverts the last change made to the currently selected KPL program file. This command can be repeated multiple times, each time reverting the previous change, all the way back to when the file was opened in KPL.

Redo – Reverts the last Undo made in the currently selected KPL program file. This command can be repeated multiple times, each time reverting the previous Undo.

Select All (Ctrl+A) – Selects and highlights all text in the currently selected KPL program file. Regions which are “collapsed” in the code editor (see the Code Editor section below) will still be selected in full with this function.

Find (Ctrl+F) – See image below. Displays a Find/Replace dialog as shown which allows for text searches within the current KPL program file. The Find command does not allow for changing found text – see the Replace command below for that functionality.

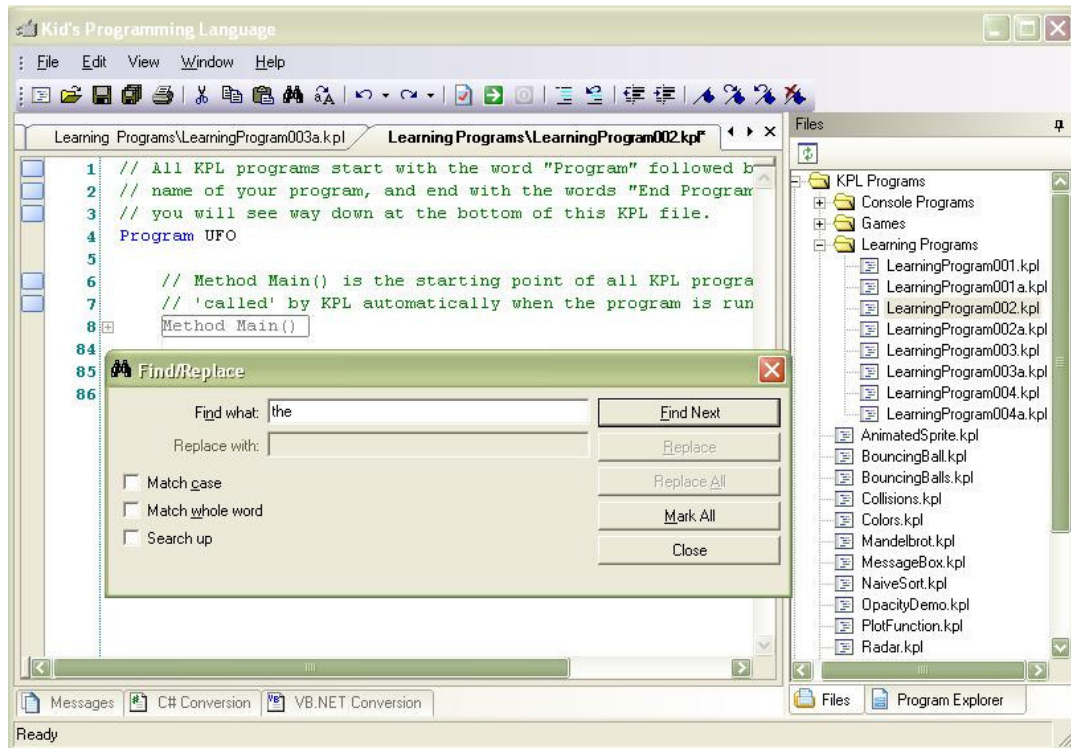
To use Find, type the text you want to find in the “**Find what:**” textbox and hit Enter or click the “**Find Next**” button. The next instance of the word you are finding will be selected in the Code Editor, and the editor will scroll if necessary to display the selected text. If the next instance of that word happens to be in a collapsed region, that region will also be expanded as the text is found and selected.

Searches begin at the top of the KPL program file by default, and proceed downward as you click **Find Next**. To search from the bottom of the file up, you can check the **Search Up** checkbox and click **Find Next**.

Text searches, by default, will match the text no matter whether it appears in upper or lower case. To perform a case-specific search, click the **Match Case** checkbox.

Text searches, by default, will match text which is embedded within longer words. To match only whole words, you can check the **Match whole word** checkbox.

The Mark All button will place Bookmarks on each line in the KPL program file which contains the word you are searching for. For more information on Bookmarks, see the Code Editor section below.



Replace (Ctrl+H) – Displays the Find/Replace dialog described above, with the **Replace** and **Replace All** buttons enabled to allow text replacement as well as searching. To perform a replace, type the new text you would like inserted into the **Replace with:** textbox. Clicking the **Replace** button will cause only the currently selected instance of the found word to be replaced with your new text. Clicking the **Replace All** button will cause all instances of the found word in the file to be replaced, and will tell you how many instances were found and replaced. A single **Undo** action will revert a **Replace All** action.

View Menu

The View menu can be accessed by holding the Alt key down and pressing the V key (Alt+V). Once the View menu is displayed, items on it can be selected by pressing the key which is underlined in the menu selection. For example, pressing Alt+V then M will cause the Message pane to be displayed.

Messages (Ctrl+M) – This command displays the Messages panel in the Collapsible Message Pane (see that section below). If the Collapsible Message Pane is not already open, this command will extend it.

Files Explorer (Ctrl+Shift+F) – This command displays the Files panel in the Explorer Pane (see that section below). If the Explorer Pane is not already open, this command will extend it.

Program Explorer (Ctrl+E) – This command displays the Program Explorer panel in the Explorer Pane (see that section below). If the Explorer Pane is not already open, this command will extend it.

C# Conversion – This command displays the C# Conversion panel in the Collapsible Message Pane (see that section below). If the Collapsible Message Pane is not already open, this command will extend it.

VB.NET Conversion – This command displays the VB.NET Conversion panel in the Collapsible Message Pane (see that section below). If the Collapsible Message Pane is not already open, this command will extend it.

Window Menu

Each open KPL program file will be added to this menu as it is open and removed as it is closed. It simply provides a menu and keyboard-based shortcut to those open files. For instance typing **Alt+W** then **1** will select the first open KPL program file; **Alt+W** then **3** will select the third open KPL program file, etc...

Help Menu

About KPL – This menu function displays the KPL about box as shown below. The about box is important because it provides version information about the version of KPL which is installed on the computer, and because it offers a convenient link to the [Kids Programming Language](http://www.kidsprogramminglanguage.com) website.
























Technical Support – This menu function displays the dialog as shown below, offering information about and a link to KPL technical support online.





KPL's Toolbar

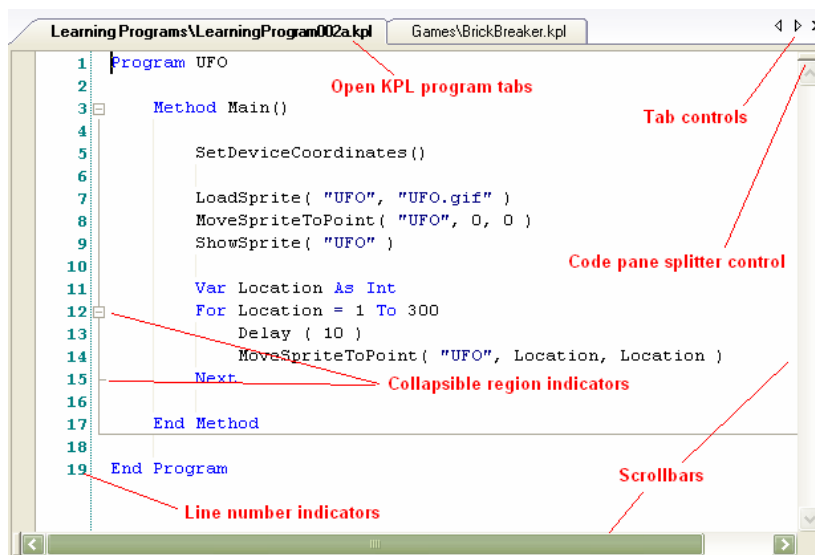
KPL's toolbar is Windows-standard, including support for mouse-over tooltips, which match the functions in the table below. Note that toolbar buttons will enable or disable as appropriate. For instance, if no KPL program files are open in the Code Editor, nearly all the toolbar buttons are grayed out and disabled – since they are only relevant to a selected KPL program file.

Function	Icon	Explanation
New Program		Creates a new template KPL program, which is properly structured, compiles and runs – though it does not do anything functional until further code is provided. This function is also available from the Ctrl+N hotkey.
Open Program		Launches a dialog allowing the user to browse to and open a KPL program from disk. This function is also available from the Ctrl+O hotkey.
Save Program		Saves any code changes which have been made to the currently selected KPL program. This function is also available from the Ctrl+S hotkey.
Save All		Saves any code changes which have been made to all open KPL program files. This function is also available from the Ctrl+Shift+S hotkey.
Print Program		Launches a Windows-standard Print dialog allowing you to print the currently selected KPL program. This function is also available from the Ctrl+P hotkey.
Cut		Cuts the currently selected text from the currently selected KPL program file. Cut text is placed on the Windows Clipboard. This function is also available from the Ctrl+X hotkey.
Copy		Copies the currently selected text from the currently selected KPL program file, without deleting it. Copied text is placed on the Windows Clipboard. This function is also available from the Ctrl+C hotkey.
Paste		Pastes the Windows Clipboard contents into the currently selected KPL program file at the cursor location. Pasted text is still available on the Windows Clipboard. This function is also available from the Ctrl+V hotkey.
Find		See the Find dialog described under the View menu above. This function displays the Find dialog. This function is also available from the Ctrl+F hotkey.
Replace		See the Replace dialog described under the View menu above. This function displays the Replace dialog. This function is also available from the Ctrl+H hotkey.
Undo		Reverts the last change made to the currently selected KPL program file. This button is grayed if no changes are available to Undo, blue if there are changes to Undo. This command can be repeated multiple times,

		each time reverting the previous change, all the way back to when the file was opened in KPL. This function is also available from the Ctrl+Z hotkey.
Redo		Reverts the last Undo made in the currently selected KPL program file. This command can be repeated multiple times, each time reverting the previous Undo. This button is grayed if no changes are available to Redo, blue if there are changes to Redo.
Check program for errors		This function causes KPL to check the currently select program file for errors. The Messages pane will display the results of this check. If the Messages pane is not displayed when this function is invoked, it will be displayed.
Run the program		This function causes KPL to check the currently select program file for errors, and if no errors are found, to launch the run dialog to run the program. The Messages pane will display the results of the check. If the Messages pane is not displayed when this function is invoked, it will be displayed.
Stop the running program		This function stops the currently running KPL program. If no program is running, the button is disabled and grayed as shown. When a program is running, the button is enabled and red in color.
Comment selected code		This function causes one or more selected lines of KPL code to be “commented out” so that they will no longer be part of the compiled and running KPL program. This is done by placing the // comment indicator at the beginning of the line.
Uncomment selected code		This function causes one or more selected lines of KPL code to be “uncommented,” so that it will again be part of the compiled and running KPL code for the program. If the // comment indicator is located at the beginning of the line of code. This function has no effect if the line of code is not already commented out, or if the comment indicator is not at the beginning of the line of code.
Decrease indent		This function causes one or more selected lines of KPL code to be shifted to the left by one indented tab stop. If the code on the selected line already begins at the left margin, it will not be shifted.
Increase indent		This function causes one or more selected lines of KPL code to be shifted to the right by one indented tab stop.
Toggle bookmark		Toggles a bookmark on or off on the currently selected line of KPL code. Bookmarks can be used to quickly return to a particular place in your KPL program. For more information on bookmarks, see the Code Editor section below.
Go to next bookmark		This function moves the cursor within the KPL program to the next line of code which is bookmarked. This function will wrap to the first bookmark from the

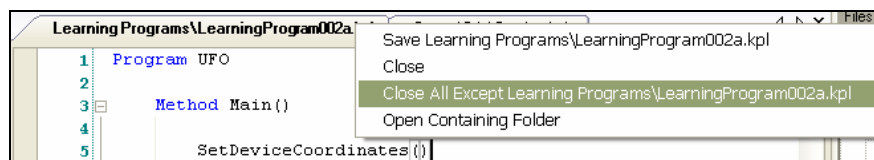
		top of the KPL program, if there is a bookmark above the current cursor but not one below.
Go to previous bookmark		This function moves the cursor within the KPL program to the first previous line of code which is bookmarked. This function will wrap to the first bookmark from the bottom of the KPL program, if there is a bookmark below the current cursor but not one above.
Clear all bookmarks		This function clears all bookmarks in the currently selected KPL program file.

KPL's Code Editor



Open KPL program tabs

Each KPL program file which is open appears on its own tab in the code editor. Clicking on the tab for any open file will cause the code for that file to be shown in the code editor pane. The selected tab clearly visually indicates which file is being displayed. A right-click on a tab shows the pop-up menu below:



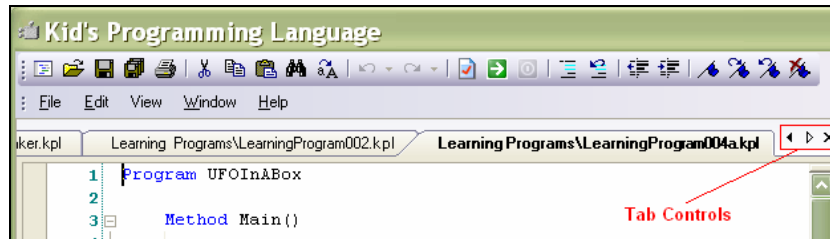
Save and **Close** are self-evident, and refer to the file whose tab was right-clicked.

Close All Except... is a very handy feature. Often, multiple files can be open in the editor, and using this function closes all of the open files *except for* the file which is right-clicked. It's a convenient way to "clean up" by closing program files which are no longer being used.

Both **Close** functions will, of course, prompt the user to save a file before closing, if unsaved changes have been made to the file.

Open Containing Folder is also a convenient shortcut. It will launch a standard File Explorer dialog for the Windows folder which contains the right-clicked KPL program. This allows for all the usual Windows file functions – for instance, zipping the file, mailing it to someone, copying it to a different location, etc...

Tab Controls

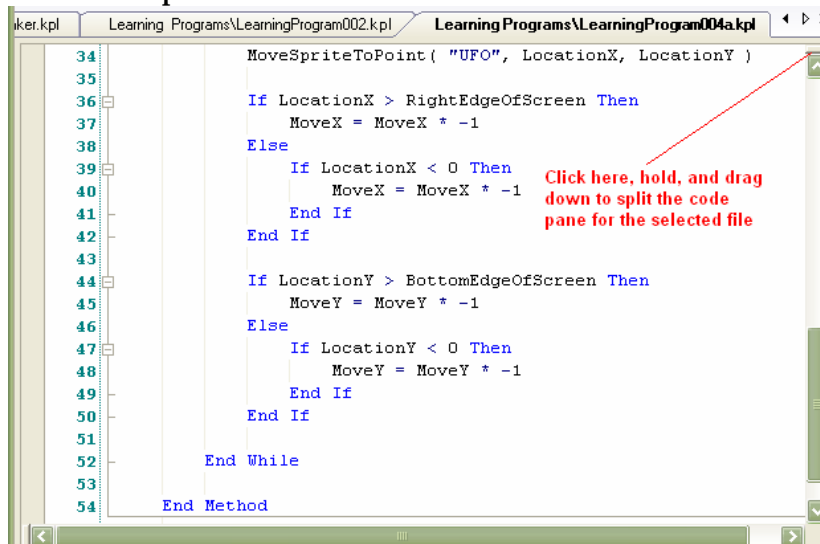


Three small controls at the far right of the strip of tabs are the **Tab Controls**, which allow additional control of those tabs.

The **left and right arrow indicators** allow scrolling of the displayed tabs to the right or the left, in case there are so many tabs displayed that they cannot all fit in the width of the Code Editor Pane. Each indicator will be filled in with black if there are tabs in the indicated direction which are not fully displayed.

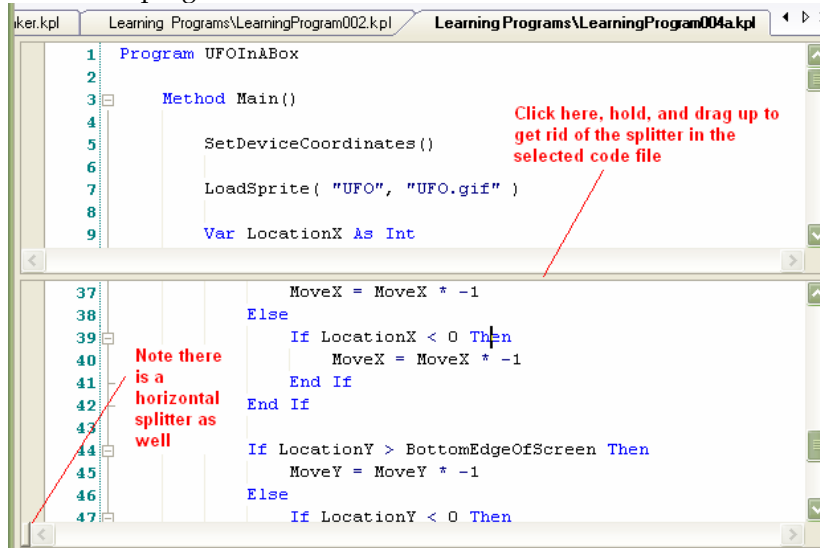
The **X** indicator, as usual, is a shortcut to close the selected KPL program file. Again as usual, if changes have been made to the selected file and not yet saved, you will be asked if you wish to save them before this function closes the file.

Code Pane Splitter Control



As the image shows, click the little rectangle at the top of the code pane, hold, and drag down to create a splitter allowing independent parts of the code file to be displayed and viewed. This can be convenient when comparing or working with code in different parts of a single KPL file.

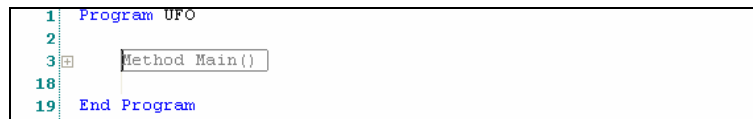
You can then click on the splitter bar, hold, and drag back up to the top to remove the split view in the KPL program file, as shown here:



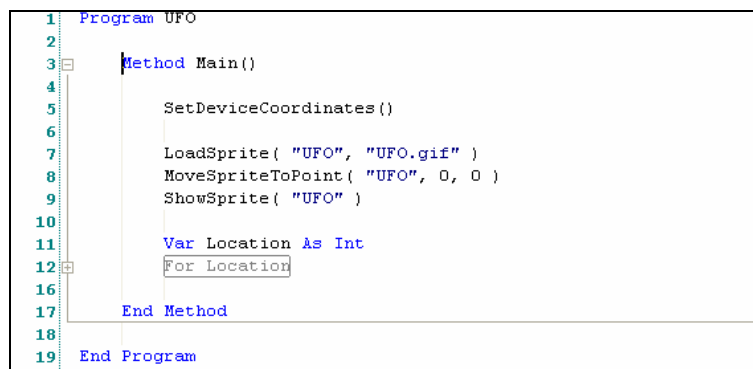
Note that there is a similar horizontal splitter control at the left end of the horizontal scroll bar.

Collapsible Region Indicators

It can often be convenient when programming to “collapse” code which you are *not* currently working on, in order to concentrate better on code which you *are* currently working on.



As you can see by noting the line numbers above, this does not remove the collapsed code from the KPL program, it simply hides it until you “expand” the collapsed region, as shown here:



There are several points to note about regions. First, a convenient vertical line extends down the left edge of the code region, and then a horizontal line extends across to show the bottom of the region, making it very easy to see how much code is included within the region.

Second, it is possible to have collapsible regions within other regions in the code, as shown with the **FOR** loop above. KPL automatically creates a collapsible region for every function, method, loop, structure and if...then block.

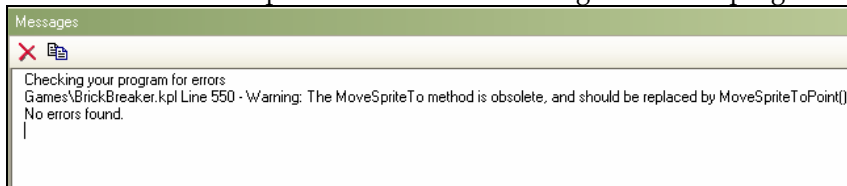
Third, try collapsing a region and then placing the mouse cursor over that collapsed region. KPL will automatically display a tooltip (potentially a very large one!) which shows the code in that region, without the need to open the region in the code pane. Here's how that looks, with the mouse over the collapsed Method Main():



Line Number Indicators

KPL is not a line-number driven sequential language, like BASIC used to be, and you cannot reference line numbers in code, like the old **GOTO 200** programming commands. In KPL, line numbers are simply informational.

The most common use for line numbers is in the **Messages Pane** (more details about that below), when KPL needs to report an error or a warning in the KPL program. Here's an example:



When KPL refers to a line of code by number this way in the Messages pane, you can double-click on the line in the Messages pane and the code editor will automatically move to and display that line of code.

Working with KPL Code in the Code Editor

It is traditional in coding to “indent” code to help indicate its hierarchical organization. This makes it easier for you and for others to understand the flow of your program. Indenting is normally done as you move deeper “into” the KPL code – such as into **Method Main()** below, and then again into a FOR loop within the method:

```
1 Program UFO
2
3 Method Main()
4
5     SetDeviceCoordinates()
6
7     LoadSprite( "UFO", "UFO.gif" )
8     MoveSpriteToPoint( "UFO", 0, 0 )
9     ShowSprite( "UFO" )
10
11     Var Location As Int
12     For Location = 1 To 300
13         Delay ( 10 )
14         MoveSpriteToPoint( "UFO", Location, Location )
15     Next
16
17 End Method
18
19 End Program
```

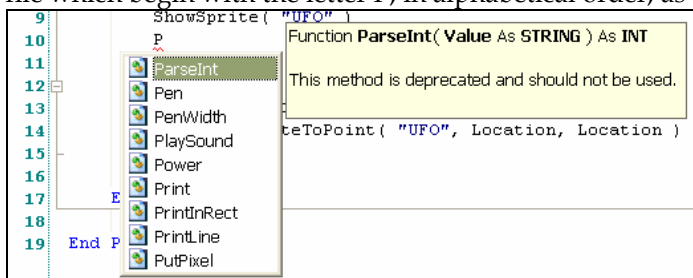
The usual way to indent is simply using the <Tab> key. Each tab stop is 4 characters wide. Note that faint vertical lines actually show the location of each tab stop, to make it easy to keep code aligned.

Code which is neatly aligned, indented, and organized using white space is far easier to read, understand and work with – whether you wrote it yourself, or someone else did. So it is important to develop good discipline and practices such as these.

The KPL Code Editor supports the usual Cut, Copy and Paste functions (see above). It does not at the moment support dragging and dropping of selected blocks of code.

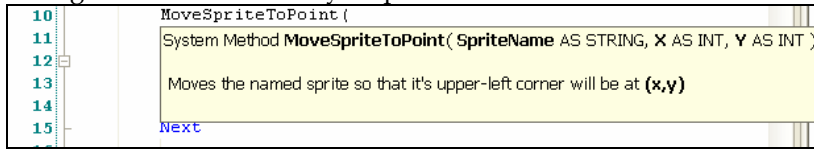
Intellisense/Autocomplete in the Code Editor

With the cursor blinking on a blank line in the Code Editor, type a **P**, then hold the **CTRL** key and type **J** (**CTRL+J**). This is the Microsoft-standard keystroke for invoking “Intellisense.” Intellisense presents you with a list of possible methods to complete the word you began typing. For instance, if you typed **P**, you will see a popup which presents you with all known system methods or functions which begin with the letter **P**, plus any methods or functions in your KPL file which begin with the letter **P**, in alphabetical order, as shown below:



When the Intellisense popup is displayed, you can use the up/down arrows or the mouse to select a function in the list. When a function is selected, a tooltip will be displayed showing its definition and description. If you hit <Enter> or double-click on a function in the list, it will be placed into the code file at the cursor location. If you hit <Esc> while the list is displayed, the list will go away and your code will not be changed.

Another example of Intellisense is what happens when you type a known method or function name in the KPL code editor. KPL recognizes it, and as soon as you type the (after the method name, KPL displays a tooltip below your line of code defining the function or method you are calling. This is an extremely helpful and convenient reminder.



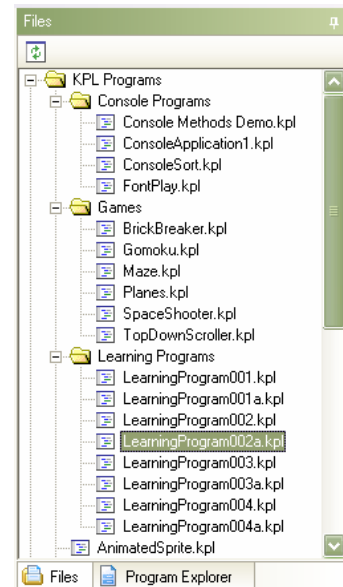
An interesting and useful side effect of Intellisense is that if you hit **CTRL+J** on a **blank** line in the code editor, the popup list which appears will present **ALL** available methods and functions – those available from KPL as well as those available from your own code. It's sometimes more convenient to look things up in the IDE than in this document. :)

Explorer Pane

The Explorer Pane is one of the optional panes in the KPL user interface. By default, it is located to the right of the Code Pane, and is "pinned" open. There are two tabs within the explorer pane, **Files** and **Program Explorer**. In this case the tabs are displayed and selectable at the bottom of the pane. Horizontal and vertical scrollbars are automatically displayed if needed.

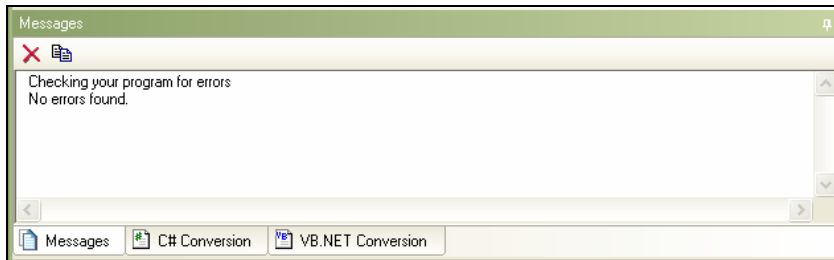
The **Files** tab shows files under the KPL install folder, and also shows folders under the KPL install folder, if those folders contain KPL program files. Folders in this display can be opened or closed. Double click or right-click on a file to "open" it in KPL.

The **Program Explorer** tab is commonly enough used when working with KPL that it's best to leave it as displayed by default. If you "unpin" it by clicking the small pin icon in the upper right of the pane, the pane will collapse to the edge of the KPL window. When the pane is collapsed, the Files and Program Explorer tabs will still be displayed, and clicking on them will expand the pane, showing the view matching the tab you clicked. Unpinning to a collapsed Explorer Pane can be useful at times, to allow for more space for viewing or working with code in the code pane. You can re-pin the pane open by clicking the pin icon again.



This pane can also be resized – place the mouse over the left edge of the pane until the cursor changes, then click and drag to move the left edge to where you would like it to be.

Messages Pane



The Messages Pane is the second optional pane in the KPL user interface. By default, it is located below the Code Pane, and is “pinned” open. There are three tabs at the base of the Messages pane: **Messages**, **C# Conversion** and **VB.NET Conversion**. In this case the tabs are displayed and selectable at the bottom of the pane. Horizontal and vertical scrollbars are automatically displayed if needed.

The **Messages** tab is where KPL displays status messages as KPL checks your program for errors. Note that if an error or warning referring to a particular line of code is displayed here in the messages tab, you can double-click that line in the messages tab, and the code editor will automatically display and place the cursor on that line of code.

The Messages tab is also where trace messages are displayed as your program runs, when you use the **Trace()** system method in your code.

KPL will also automatically inform you of certain important status information on the Message tab, such as when there is an update available to KPL available.

The Messages Pane is commonly enough used when working with KPL that it’s best to leave it as displayed by default. If you “unpin” it by clicking the small pin icon in the upper right of the pane, the pane will collapse to the edge of the KPL window. When it is collapsed, its tabs will still be available, and clicking on them will expand the pane, showing the view matching the tab you clicked. Unpinning to a collapsed Messages Pane can be useful at times, to allow for more space for viewing or working with code in the code pane. You can re-pin the pane open by clicking the pin icon again.

This pane can also be resized – place the mouse over the top edge of the pane until the cursor changes, then click and drag to move the top edge to where you would like it to be.

The **C# Conversion** tab and the **VB.NET Conversion** tab are where code in each language will be generated and displayed when you click the matching View menu option. This code is a syntactically complete and accurate conversion of the KPL code into that language. KPL’s system methods and functions will available soon to Visual Studio.NET programmers using C#, VB.NET or other .NET languages – but until they are, the conversions generated by KPL cannot be run in Visual Studio.NET, and this generated code is for educational purposes only.

Identifiers

Valid identifiers:

Invalid Identifiers:

Arrays

An Array is a collection of variables of the same type, and has the following structure

```
DEFINE <Identifier> AS <Type> [ <Expression> ]
```

Example:

```
DEFINE Values AS INT[ 3 ]
Values[0] = 10
Values[1] = 20
Values[2] = 30
Print( "Value of index 0 is: " + Values[0] )
```

Multidimensional array are not yet supported in the released version of KPL, but we do have them working in a pre-release build, and will be releasing that as soon as possible.

Structures

A STRUCTURE allows you to keep track of several pieces of information (variables) while treating them all as a single logical entity. It is important to note that the fields in a STRUCTURE do not require the DEFINE keyword, and cannot use an initializer, unlike regular variables.

```
STRUCTURE <Identifier>

<Identifier> AS <Type>

...

END STRUCTURE
```

Fields in a STRUCTURE variable are accessed by using the name of the variable, followed by a '.' (dot) character, followed by the name of the field.

Example:

```
STRUCTURE Ball
Color AS STRING
Size AS INT
END STRUCTURE

DEFINE MyBall AS BALL
MyBall.Color = "Blue"
MyBall.Size = 2
PRINT("My " + MyBall.Color + " is " + MyBall.Size + " inches in
diameter" )
```

Variables

KPL variables are defined as follows:

```
DEFINE <Identifier> AS <Data Type> [ = <Expression> ]
```

In English, this means that a variable declaration starts with the keyword **DEFINE**, followed by a valid identifier, the keyword **AS**, the Data Type that will be contained in the variable, and optionally an initial value. The initial value can be a literal value such as “Monkey” or 10, or a variable or other expression that evaluates to the correct data type.

Variables can be defined at the top of the program (after Structures) to be available to all methods, or can be defined within the body of a method to be available to that method only. A variable cannot be declared inside of a loop or If...Then...Else construct. A variable must be declared before the code that uses it. There are several **DEFINE** declarations show in context above, but we will repeat them here to reinforce them:

```
Define Monkey AS STRING = “Monkey”
```

Or

```
Define MONKEY as string = “Monkey”
```

```
...
```

```
Define Values AS INT[ 3 ]
```

```
...
```

```
STRUCTURE Ball
```

```
Color AS STRING
```

```
Size AS INT
```

```
END STRUCTURE
```

```
Define MyBall AS BALL
```

Key Words

The following words are part of the KPL language itself and cannot be used as an identifier. Note that not all of these keywords are in use at this time, and may be reserved for future use.

AND	EXIT	METHOD
AS	FALSE	MOD
DO	FOR	NOT
DEFINE	FUNCTION	OR
EACH	GOTO	PROGRAM
END	WHILE	RETURN

Comments

A comment can be a line comment or a block comment. A comment is a part of your program that is ignored when the program is run, and allows you to leave yourself “notes” directly in the program code.

Line comments

Line comments are preceded by two forward slashes and everything to the end of the line is treated as a comment.

Example:

```
// These are line comments. Everything to the right of the '//'  
// is a comment and is ignored. Line comments end at the end of  
// the line.
```

Block Comments

Block comments are started by the /* characters and end with the */ characters. Everything between is a comment.\

Example:

```
/*  
This comment can span multiple  
lines. You can leave really big comments  
using this style of comment.  
*/
```

Loops

KPL currently supports three types of loops:

Loop Loop

A very simple and intuitive loop construct, Loop, allows the specification of the number of times to run the block of code within the loop construct. This Loop can be controlled with either with a specific numeric value, or an integer variable:

```
Loop 5  
    PrintLine("Printing")  
End Loop
```

```

Define X As Int = 2
Loop X
    PrintLine("Printing")
End Loop

```

For Loop

A FOR loop executes a specific number of times based on the number of times it takes from reach <End Value> from <Start Value>, and has the following structure:

```

FOR <Variable> = <Start Value> TO <End Value> [ <Step> <Value> ]

    <Statements>

NEXT

```

Examples:

```

For Location = 0 To 300
    Delay ( 10 )
    MoveSpriteToPoint( "UFO", Location, Location )
Next

For Location = 0 To 300 Step 10
    Delay ( 10 )
    MoveSpriteToPoint( "UFO", Location, Location )
Next

```

While Loop

A WHILE loop executes while the <Boolean Expression> evaluates to TRUE, and has the following structure:

```

WHILE <Boolean Expression>

    <Statements>

END WHILE

```

Example:

```

While IsKeyDown("Q") = False

    Delay ( 20 )

    LocationX = LocationX + MoveX
    LocationY = LocationY + MoveY
    MoveSpriteToPoint( "UFO", LocationX, LocationY )

End While

```


Decisions and Branching

KPL supports decision structures that allow you to run code based on a defined condition using the If / Then / Else / End If construct, which has the following structure:

```
If X = 1 Then
    Alert("Value is 1", "Displaying Value")
End If

If X = 2 Then
    Alert("Value is 2", "Displaying Value")
Else
    Alert("Value is not 2", "Displaying Value")
End If
```

Functions and Methods

KPL allows the user to create methods and functions, which are chunks of executable code that can be called from elsewhere in the program to perform a specific set of functions. See section four below for a full list of the system functions and methods available to a KPL programmer.

Methods

A method is defined by the METHOD keyword followed by a valid identifier, list of parameters, any number of code statements, and the END METHOD keywords.

Examples:

```
Method HelloWorld()
    PRINT( "Hello World" )
End Method

Method PrintText( Text As String )
    PRINT( Text )
End Method
```

Functions

A function is nearly identical to a METHOD, with the exception that it can return a value to the caller while a METHOD cannot. A function is defined by the FUNCTION keyword followed by a valid identifier, list of parameters, return type, any number of code statements, and the END FUNCTION keywords. A function must contain a RETURN statement that returns a value to the caller.

Examples:

```

Function TripleIt( Value As Int ) As Int
    Return Value * 3
End Function

Function ContainsMonkey( Text As String ) As Bool
    If SubString( Text, "Monkey" ) <> 0 Then
        Return True
    Else
        Return False
    End If
End Function

```

All method and function calls require the use of parenthesis around the values being passed in, unlike other languages like BASIC.

Examples:

- `PrintText("Hello World")` - will call the `PrintText()` function defined above with the value "Hello World"
- `Clear()` - will clear the screen. The `Clear()` method does not have any arguments, so an open parenthesis followed by a closed parenthesis is used to indicate that no values are being passed.
- `PrintText "Hello World"` - is **not** a valid method call because it lacks parentheses

Section Three: Learning Program Examples

KPL's current install contains a folder called Learning Programs – look for it in the File Explorer in KPL. There are currently 6 Learning Programs available within KPL, and more will be added over time. We will present each of the current Learning Program examples here, for reference, and for learning. Reviewing them here, in conjunction with the KPL language definition above and the System Methods and Functions below, will hopefully help the user understand KPL language and programming more quickly.

Learning Program 001 – Hello World!

The source code for the program below uses the same color coding as the KPL code editor: green indicates information comments, blue indicates language keywords, dark blue indicates string values.

This program introduces the basic program structure for a KPL program, plus the user of String variables (used for alphanumeric text values), and the defining and calling of user-defined Methods:

```
// All KPL programs start with the word "Program" followed by the
// name of your program, and end with the words "End Program", which
// you will see way down at the bottom of the file.
Program HelloWorld

    // These green-colored lines that start with a // are called "Comments"
    // in KPL. They are not actually instructions to the computer, and
    // are ignored by the computer when it runs your KPL program. They
    // are added by the programmer in order to explain their KPL code to
    // other programmers who are reading it – like you are now!

    // Method SayHello() will print Hello! onto the screen. In KPL you
    // can put code into a method which can be called from other KPL code.
    // Sometimes you will want to do that so you can call the method
    // over and over again. Sometimes you will want to do that just to
    // keep your KPL code better organized and easier to understand.
Method SayHello()
    // The PrintLine Method will print onto the screen whatever
    // value I give it.
    PrintLine ("Hello!")
End Method

    // Another important reason for putting code into a method is that by
    // sending "parameters" to a method, you can make it behave differently
    // each time you call it. In this method, we have added a parameter
    // which is a String variable. When we pass a String value to this
    // method, it will be stored in the Something parameter, and the method
    // will print the value we have passed in on the screen. No matter what
    // String value we send to the method, the method will print it on the
    // screen.
Method SaySomething(Something As String)
    // In this case, the value we give to PrintLine is actually
    // the parameter which was passed into this Method
    PrintLine (Something)
End Method

    // Method Main() is the starting point of all KPL programs, and is
    // 'called' by KPL automatically when the program is run.
Method Main()

    // In KPL, you can print and draw in lots of different colors.
    // One of the first things a KPL program does is use the Color()
    // method to pick the color which will be used.
    Color ( Blue )

    // Now we will call the SayHello() method, which we defined above,
    // and which will print Hello! onto the screen
```

```

SayHello()

// We want to be able to say different things, so we will make
// a String variable called WhatIWannaSay. A String variable
// holds text information, like anything that can be printed on
// a page or typed on a keyboard. A variable is declared using
// the Define keyword, plus the name I want to use for the variable,
// plus a definition of what kind of variable it is - in this
// case, 'As String'
Define WhatIWannaSay As String

// This is how I give a value to the variable I have declared.
// To give a string a value, you must put quote marks " and "
// around the value. In this case, the value is: Hello Again!
WhatIWannaSay = "Hello Again!"

// Now I call the SaySomething Method, passing my variable to it.
// The Method will print onto the screen whatever value I have
// given to the variable.
SaySomething(WhatIWannaSay)

// To prove that it works, let's change the value of our
// variable, and call the method again. Changing the value of
// a variable And Then using it's new value like this is one of
// the most basic and important principles of computer programming.
WhatIWannaSay = "\"Goodbye!\""

// Now I call the SaySomething Method with the same variable,
// but that variable has a new value, so the Method will Print
// that new value.
SaySomething(WhatIWannaSay)

// After you have run this to see it work, go ahead and type
// new values in for the WhatIWannaSay variable, and then
// run the program again to see what you typed!

End Method

End Program

```

Sometimes code without comments can be easier to understand as a unit, so here is the code for Learning Program 001, without any commenting:

```

Program HelloWorld

Method SayHello()
    PrintLine ("Hello!")
End Method

Method SaySomething(Something As String)
    PrintLine (Something)
End Method

Method Main()
    Color ( Blue )

    SayHello()

    Define WhatIWannaSay As String
    WhatIWannaSay = "Hello Again!"
    SaySomething(WhatIWannaSay)

    WhatIWannaSay = "Goodbye!"
    SaySomething(WhatIWannaSay)

End Method

End Program

```

Learning Program 002 – Introduction to Sprite Graphics

The source code for the program below uses the same color coding as the KPL code editor: green indicates information comments, blue indicates language keywords, dark blue indicates string values.

This program introduces KPL's coordinate system for displaying graphics, and the use of Sprites to display and animate graphical objects in a KPL program:

```
// All KPL programs start with the word "Program" followed by the
// name of your program, and end with the words "End Program", which
// you will see way down at the bottom of this KPL file.
Program UFO

    // Method Main() is the starting point of all KPL programs, and is
    // 'called' by KPL automatically when the program is run.
    Method Main()

        // KPL's default coordinate system matches the way the computers
        // normally think about graphics. In this coordinate system, left
        // edge of the screen is X = 0, and the top edge of the screen is
        // Y = 0, so the upper left corner is 0, 0. And X value increases
        // pixel-by-pixel As you move right across the screen. And Y value
        // increases pixel-by-pixel As you move down the screen.
        // So 200, 200 is near the center of a normal KPL window.

        // A Sprite is used by KPL to keep track of and to do things to
        // an object that you want to display on the screen. In this
        // example we are going to use a UFO as a Sprite. KPL comes with
        // many picture files which you can use as Sprites. In the
        // Files Explorer (normally to the right of this code area)
        // you can look in the Pictures folder to see those files.
        // Look for the file UFO.gif, right click on it, and select
        // View. A picture viewer will allow you to preview that image.

        // The first thing to do with a Sprite is to Load it, as shown
        // here. Loading it tells KPL what you will be calling this Sprite
        // (in this case, UFO), and also tells KPL what image file the
        // Sprite should be loaded from (in this case, UFO.gif)
        LoadSprite( "UFO", "UFO.gif" )

        // Sprites are moved to a specific location on screen using
        // an X, Y coordinate system. X = 0 is the left edge of the
        // screen. Y = 0 is the top of the screen. So, this method
        // call to MoveSpriteTo places our UFO at 0, 0 - which is the
        // upper left corner of the screen.
        MoveSpriteToPoint( "UFO", 0, 0 )

        // The Sprite will not actually be displayed on the screen
        // until you call the ShowSprite method as shown.
        ShowSprite( "UFO" )

        // We want the UFO to move across the screen, so we will
        // declare a variable which will hold it's Location as we
        // move it.
        Define Location As Int

        // A For loop tells KPL to count, in this case from 1 to 300,
        // and on each count KPL will run the code which is defined
        // inside the For loop, between the For line and the Next line.
        For Location = 1 To 300

            // The Delay method tells KPL to pause for a moment before
            // continuing to run the program. This is often important
            // because computers are so fast at displaying graphics that
            // we have to tell them to slow down so we can see the graphics
            // move. In our case, we are telling KPL to pause just a
            // moment before moving the UFO to it's new Location.
            Delay ( 10 )
```

```

        // We are going to use the MoveSpriteTo method again.
        // Remember that because we are in the For loop, the computer
        // is counting using the Location variable, from 1 to 300.
        // We are going to keep this simple by using the Location
        // variable for the X coordinate as well as the Y coordinate.
        // So the first time through the loop, Location is 1, and the
        // call below to MoveSpriteTo uses the value 1 and places the UFO
        // on the screen at 1,1 - just a little to the right and below
        // where it was before. The second time through the loop,
        // Location is 2, so the Sprite is placed at 2,2. And so on,
        // until the computer counts all the way to 300. By the time
        // the computer gets to 300 and stops counting, the For loop
        // has moved the UFO far down the screen, and to the right.
        MoveSpriteToPoint( "UFO", Location, Location )

    // The Next keyword defines the end of the For loop, and tells
    // KPL to count to the next value
    Next

End Method

End Program

```

Sometimes code without comments can be easier to understand as a unit, so here is the code for Learning Program 002, without any commenting. This is our favorite example for showing people how little KPL code is required to make a visually interesting KPL program!

```

Program UFO

Method Main()

    SetDeviceCoordinates()

    LoadSprite( "UFO", "UFO.gif" )
    MoveSpriteToPoint( "UFO", 0, 0 )
    ShowSprite( "UFO" )

    Define Location As Int
    For Location = 1 To 300 Step 10
        Delay ( 10 )
        MoveSpriteToPoint( "UFO", Location, Location )
    Next

End Method

End Program

```

Learning Program 003 – Arrays, and Sprite Animation Timelines

The source code for the program below uses the same color coding as the KPL code editor: green indicates information comments, blue indicates language keywords, dark blue indicates string values.

This program introduces the use of Animated Sprites – the basic and simple component for a fun graphical game, as well as the use of arrays, which are required for sprite animation timelines:

```

Program UFO

Method Main()

    // The first thing to do with a Sprite is to Load it, as shown
    // here. Loading it tells KPL the name you will use to refer to
    // this Sprite (in this case, UFO), and also tells KPL what image
    // file the Sprite should be loaded from (in this case, UFO.gif)
    LoadSprite( "UFO", "UFO.gif" )

    // Sprites are moved to a specific location on screen using

```

```

// an X, Y coordinate system. This method call to MoveSpriteTo
// places our UFO at 0, 0 - which is the upper left corner of the
// screen.
MoveSpriteToPoint( "UFO", 0, 0 )

// Our UFO image actually supports 6 frames of animation, which make
// it look like the UFO is spinning around like a top. UFOs do that,
// right? :) We are going to use an array of 6 Integer values to
// tell the sprite how long to display each animation frame before moving
// on to show the next frame. Try changing these values to 20 and the
// UFO will spin much more quickly. Change them to 200 and it will
// spin much more slowly.
Define Timeline As Int[6]
Timeline[1] = 50
Timeline[2] = 50
Timeline[3] = 50
Timeline[4] = 50
Timeline[5] = 50
Timeline[6] = 50

// This method call to SetSpriteAnimationTimeline tells KPL to
// use the timeline we just set up to animate the UFO. When the
// animation has shown all 6 parts, we want it to start over again,
// so we send True as the value for the AutoRewind parameter. If
// you change True to False, the UFO will only spin around once.
SetSpriteAnimationTimeline( "UFO", True, Timeline )

// The Sprite will not actually be displayed on the screen
// until we call the ShowSprite method as shown.
ShowSprite( "UFO" )

// We want the UFO to move across the screen, so we will
// declare a variable which will hold it's Location as we
// move it.
Define Location As Int

// A For loop tells KPL to count, in this case from 1 to 300,
// and on each count KPL will run the code which is defined
// inside the For loop, between the For line and the Next line.
For Location = 1 To 300

    // The Delay method tells KPL to pause for a moment before
    // continuing to run the program. This is often important
    // because computers are so fast at displaying graphics that
    // we have to tell them to slow down so we can see the graphics
    // move.
    Delay ( 40 )

    // We are going to keep this simple by using the Location
    // variable for the X coordinate as well as the Y coordinate.
    MoveSpriteToPoint( "UFO", Location, Location )

// The Next keyword defines the end of the For loop, and tells
// KPL to count to the next value
Next

End Method

End Program

```

Sometimes code without comments can be easier to understand as a unit, so here is the code for Learning Program 003, without any commenting:

Program UFO

```
Method Main()

    LoadSprite( "UFO", "UFO.gif" )
    MoveSpriteToPoint( "UFO", 0, 0 )

    Define Timeline As Int[6]
    Timeline[1] = 50
    Timeline[2] = 50
    Timeline[3] = 50
    Timeline[4] = 50
    Timeline[5] = 50
    Timeline[6] = 50
    SetSpriteAnimationTimeline( "UFO", True, Timeline )

    ShowSprite( "UFO" )

    Define Location As Int

    For Location = 1 To 300
        Delay ( 40 )
        MoveSpriteToPoint( "UFO", Location, Location )
    Next
End Method
End Program
```

Learning Program 004 – While Loops and If...Then...Else logic

The source code for the program below uses the same color coding as the KPL code editor: green indicates information comments, blue indicates language keywords, dark blue indicates string values.

This program introduces the use of **While** Loops and **If...Then...Else** logic, and the use of **ScreenHeight()** and **ScreenWidth()** functions:

Program UFOInABox

```
Method Main()

    LoadSprite( "UFO", "UFO.gif" )

    // We want the UFO to move around the screen in a dynamic and
    // unpredictable way, so we will declare two variables which
    // will keep track of the current Location - one for the X
    // location and one for the Y location. Two other variables
    // will keep track of how the UFO is moving - one for the X
    // direction move, and one for the Y direction move.
    Var LocationX As Int
    Var LocationY As Int
    Var MoveX As Int
    Var MoveY As Int

    // 0, 0 is the X, Y coordinate for the upper left corner of the
    // screen, where the UFO will start
    LocationX = 0
    LocationY = 0
    MoveSpriteToPoint( "UFO", LocationX, LocationY )

    // The Sprite will not actually be displayed on the screen
    // until we call the ShowSprite method.
    ShowSprite( "UFO" )

    // We use the ScreenWidth() method to find out how wide the screen is,
    // but since the UFO is about 65 pixels across, we need it to bounce
    // as soon as it gets within 65 pixels of that right edge.
    Var RightEdgeOfScreen As Int
    RightEdgeOfScreen = ScreenWidth() - 65
```



```

// We use the ScreenHeight() method to find out how high the screen is,
// but since the UFO is about 35 pixels high, we need it to bounce as
// soon as it gets within 35 pixels of that bottom edge.
Var BottomEdgeOfScreen As Int
BottomEdgeOfScreen = ScreenHeight() - 35

// Random() generates a random number between the two values specified.
// We want the UFO to move differently each time the program runs, so we
// will use Random() in this way to make it different and unpredictable
// when the program starts. This will mean that the UFO will be
// moving anywhere from 5 to 10 pixels at a time to the right, and 5 to
// 10 pixels at a time down.
MoveX = Random(5, 10)
MoveY = Random(5, 10)

// We will use a While loop this time, so that the UFO will keep
// moving until the Q key is pressed on the keyboard, or until we click
// the "Stop the program" button.
While IsKeyDown("Q") = False

    Delay ( 20 )

    // Now we will move the UFO by changing its location, and
    // then calling MoveSpriteTo() using that new location.
    LocationX = LocationX + MoveX
    LocationY = LocationY + MoveY
    MoveSpriteToPoint( "UFO", LocationX, LocationY )

    // If the UFO has reached the right edge of the screen,
    // we need it to bounce by changing its MoveX value to move
    // back in the opposite direction.
    If LocationX > RightEdgeOfScreen Then
        MoveX = MoveX * -1
    Else
        // Else if the UFO has reached the left edge of the screen,
        // we need it to bounce by changing its MoveX value to move
        // back in the opposite direction.
        If LocationX < 0 Then
            MoveX = MoveX * -1
        End If
    End If

    // If the UFO has reached the bottom edge of the screen,
    // we need it to bounce by changing its MoveY value to move
    // back in the opposite direction.
    If LocationY > BottomEdgeOfScreen Then
        MoveY = MoveY * -1
    Else
        // Else if the UFO has reached the top edge of the screen,
        // we need it to bounce by changing its MoveY value to
        // move back in the opposite direction.
        If LocationY < 0 Then
            MoveY = MoveY * -1
        End If
    End If

End While

End Method

End Program

```

Sometimes code without comments can be easier to understand as a unit, so here is the code for Learning Program 004, without any commenting:

```

Program UFOInABox

Method Main()

    SetDeviceCoordinates()

```

```

LoadSprite( "UFO", "UFO.gif" )

Var LocationX As Int
Var LocationY As Int
Var MoveX As Int
Var MoveY As Int

LocationX = 0
LocationY = 0
MoveSpriteToPoint( "UFO", LocationX, LocationY )
ShowSprite( "UFO" )

Var RightEdgeOfScreen As Int
RightEdgeOfScreen = ScreenWidth() - 65

Var BottomEdgeOfScreen As Int
BottomEdgeOfScreen = ScreenHeight() - 35

MoveX = Random(5, 10)
MoveY = Random(5, 10)

While IsKeyDown("Q") = False

    Delay ( 20 )

    LocationX = LocationX + MoveX
    LocationY = LocationY + MoveY
    MoveSpriteToPoint( "UFO", LocationX, LocationY )

    If LocationX > RightEdgeOfScreen Then
        MoveX = MoveX * -1
    Else
        If LocationX < 0 Then
            MoveX = MoveX * -1
        End If
    End If

    If LocationY > BottomEdgeOfScreen Then
        MoveY = MoveY * -1
    Else
        If LocationY < 0 Then
            MoveY = MoveY * -1
        End If
    End If

End While

End Method

End Program

```

Learning Program 005 – Controlling a game with Keyboard input

The source code for the program below uses the same color coding as the KPL code editor: green indicates information comments, blue indicates language keywords, dark blue indicates string values.

This program introduces the use of keyboard control of a graphical game:

Program LearningProgram005

```

Method Main()

    // We want the UFO to move across the screen, so we will
    // declare a variable which will hold it's Location as we
    // move it.
    Var LocationX As Int = 250
    Var LocationY As Int = 200

```

```

// The first thing to do with a Sprite is to Load it
LoadSprite( "UFO", "UFO.gif" )

// Sprites are moved to a specific location on screen using
// an X, Y coordinate system.
MoveSpriteToPoint( "UFO", LocationX, LocationY )

// Our UFO image actually supports 6 frames of animation, which make
// it look like the UFO is spinning around like a top. We are going
// To use an array of 6 Integer values to tell the sprite how long to
// display each animation frame before moving on to show the next frame.
Var Timeline As Int[6]
Timeline[1] = 50
Timeline[2] = 50
Timeline[3] = 50
Timeline[4] = 50
Timeline[5] = 50
Timeline[6] = 50

// This method call to SetSpriteAnimationTimeline tells KPL to
// use the timeline we just set up to animate the UFO.
SetSpriteAnimationTimeline( "UFO", True, Timeline )

// The Sprite will not actually be displayed on the screen
// until we call the ShowSprite method as shown.
ShowSprite( "UFO" )

// When looping to check for
Var TimeLastChecked As Decimal = 0.0
Var CurrentTime As Decimal

// Since True is always True, this While loop will never automatically
// end. The program will only end when the user clicks the "Stop" button
// at the top of the window.
While True

    // Each time in the While loop, we will check the current time
    CurrentTime = TickCount()

    // If 50 milliseconds have not passed since the last time we
    // checked, we don't want to do anything yet. Computers are very
    // fast - this check allows our program to slow down to human
    // speed. 50 milliseconds is 1/20th of a second, so even though
    // we are slowing the compute down, our program will still run
    // very fast.
    If CurrentTime - TimeLastChecked > 50 Then

        // If they pressed the Right Arrow key, the UFO will move
        // to the right
        If IsKeyDown("Right") Then
            LocationX = LocationX + 10
        End If

        // If they pressed the Left Arrow key, the UFO will move to
        // the left
        If IsKeyDown("Left") Then
            LocationX = LocationX - 10
        End If

        // If they pressed the Up Arrow key, the UFO will move up
        If IsKeyDown("Up") Then
            LocationY = LocationY - 10
        End If

        // If they pressed the Down Arrow key, UFO will move down
        If IsKeyDown("Down") Then
            LocationY = LocationY + 10
        End If

        // Now that we have decided where the UFO will move, we

```

```

        // will move it to that location
        MoveSpriteToPoint( "UFO", LocationX, LocationY )

        // After we move the UFO, we will set the TimeLastChecked
        // to now, so that we can wait 50 milliseconds from now
        // before checking again.
        TimeLastChecked = CurrentTime

    End If
End While
End Method
End Program

```

Sometimes code without comments can be easier to understand as a unit, so here is the code for Learning Program 005, without any commenting:

Program LearningProgram005

```

Method Main()

    Var LocationX As Int = 250
    Var LocationY As Int = 200

    LoadSprite( "UFO", "UFO.gif" )

    MoveSpriteToPoint( "UFO", LocationX, LocationY )

    Var Timeline As Int[6]
    Timeline[1] = 50
    Timeline[2] = 50
    Timeline[3] = 50
    Timeline[4] = 50
    Timeline[5] = 50
    Timeline[6] = 50

    SetSpriteAnimationTimeline( "UFO", True, Timeline )

    ShowSprite( "UFO" )

    Var TimeLastChecked As Decimal = 0.0
    Var CurrentTime As Decimal

    While True

        CurrentTime = TickCount()

        If CurrentTime - TimeLastChecked > 50 Then

            If IsKeyDown("Right") Then
                LocationX = LocationX + 10
            End If

            If IsKeyDown("Left") Then
                LocationX = LocationX - 10
            End If

            If IsKeyDown("Up") Then
                LocationY = LocationY - 10
            End If

            If IsKeyDown("Down") Then
                LocationY = LocationY + 10
            End If

            MoveSpriteToPoint( "UFO", LocationX, LocationY )

            TimeLastChecked = CurrentTime

        End If
    End While

```

```
End Method
End Program
```

Learning Program 006 – Using Structures, and 2-player gaming

The source code for the program below uses the same color coding as the KPL code editor: green indicates information comments, blue indicates language keywords, dark blue indicates string values.

This program introduces the use of Structures, and keyboard control of a two-person game:

Program LearningProgram006

```
// Structures are useful for grouping together all the necessary information
// for handling a particular object. In this case, we are grouping the
// information needed to represent a Sprite in a KPL program:
Structure PlayerSprite
    Name As String
    Image As String
    X As Decimal
    Y As Decimal
End Structure

Method Main()

    // We will have one Sprite for each player, separately controlled from
    // the keyboard.
    Var UFO As PlayerSprite
    Var SpaceStation As PlayerSprite

    // Player One will control a UFO
    UFO.X = 400
    UFO.Y = 240
    UFO.Name = "Player One"
    UFO.Image = "UFO.GIF"

    // Player Two will control a Spaceship
    SpaceStation.X = 150
    SpaceStation.Y = 200
    SpaceStation.Name = "Player Two"
    SpaceStation.Image = "SpaceStation.GIF"

    // Note that we are using the Structure to access the
    // required information about each player's sprite:
    LoadSprite( UFO.Name, UFO.Image )
    MoveSpriteToPoint( UFO.Name, UFO.X, UFO.Y )

    // Our images actually support 6 frames of animation, which make
    // it look like they are spinning around, like a top. We are going
    // to use an array of 6 Integer values to tell the sprite how long to
    // display each animation frame before moving on to show the next frame.
    Var Timeline As Int[6]
    Timeline[1] = 50
    Timeline[2] = 50
    Timeline[3] = 50
    Timeline[4] = 50
    Timeline[5] = 50
    Timeline[6] = 50
    SetSpriteAnimationTimeline( UFO.Name, True, Timeline )

    // Now display the UFO
    ShowSprite( UFO.Name )

    // Our second sprite is going to be a Space Station. Notice how we
    // can use the same animation timeline for the Space Station as we
    // did for the UFO.
    LoadSprite( SpaceStation.Name, SpaceStation.Image )
    MoveSpriteToPoint( SpaceStation.Name, SpaceStation.X, SpaceStation.Y )
    SetSpriteAnimationTimeline( SpaceStation.Name, True, Timeline )
```

```

ShowSprite( SpaceStation.Name )

Var TimeLastChecked As Decimal = 0.0
Var CurrentTime As Decimal

// Since True is always True, the program will only end when the user
// clicks the "Stop" button at the top of the window.
While True

    // Each time in the While loop, we will check the current time
    CurrentTime = TickCount()

    // If 50 milliseconds have not passed since the last time we
    // checked, we don't want to do anything yet. 50 milliseconds is
    // 1/20 of one second, by the way. Computers are FAST!
    If CurrentTime - TimeLastChecked > 50 Then

        // Player One controls the UFO, using the arrow keys on the
        // keyboard

        // If they pressed the Right Arrow key, the UFO will move
        // to the right
        If IsKeyDown("Right") Then
            UFO.X = UFO.X + 10
        End If

        // If they pressed the Left Arrow key, the UFO will move to
        // the left
        If IsKeyDown("Left") Then
            UFO.X = UFO.X - 10
        End If

        // If they pressed the Up Arrow key, the UFO will move up
        If IsKeyDown("Up") Then
            UFO.Y = UFO.Y - 10
        End If

        // If they pressed the Down Arrow key, UFO will move down
        If IsKeyDown("Down") Then
            UFO.Y = UFO.Y + 10
        End If

        // Now move the UFO
        MoveSpriteToPoint( UFO.Name, UFO.X, UFO.Y )

        // Player Two will controls the Space Station using the
        // W A S and Z keys at the other end of the keyboard

        // The 'S' key moves the Space Station right
        If IsKeyDown("S") Then
            SpaceStation.X = SpaceStation.X + 10
        End If

        // 'A' key moves the Space Station left
        If IsKeyDown("A") Then
            SpaceStation.X = SpaceStation.X - 10
        End If

        // 'W' key moves the Space Station up
        If IsKeyDown("W") Then
            SpaceStation.Y = SpaceStation.Y - 10
        End If

        // 'Z' key moves the Space Station down
        If IsKeyDown("Z") Then
            SpaceStation.Y = SpaceStation.Y + 10
        End If

        // Now we move the Space Station:
        MoveSpriteToPoint( SpaceStation.Name, SpaceStation.X,

```

```

        SpaceStation.Y )

        // After we move them, we will set the TimeLastChecked to
        // now, so that we can wait 50 milliseconds from now before
        // moving them again.
        TimeLastChecked = CurrentTime

    End If
End While
End Method
End Program

```

Sometimes code without comments can be easier to understand as a unit, so here is the code for Learning Program 006, without any commenting:

```

Program LearningProgram006

Structure PlayerSprite
    Name As String
    Image As String
    X As Decimal
    Y As Decimal
End Structure

Method Main()

    Var UFO As PlayerSprite
    Var SpaceStation As PlayerSprite

    UFO.X = 400
    UFO.Y = 240
    UFO.Name = "Player One"
    UFO.Image = "UFO.GIF"

    SpaceStation.X = 150
    SpaceStation.Y = 200
    SpaceStation.Name = "Player Two"
    SpaceStation.Image = "SpaceStation.GIF"

    LoadSprite( UFO.Name, UFO.Image )
    MoveSpriteToPoint( UFO.Name, UFO.X, UFO.Y )

    Var Timeline As Int[6]
    Timeline[1] = 50
    Timeline[2] = 50
    Timeline[3] = 50
    Timeline[4] = 50
    Timeline[5] = 50
    Timeline[6] = 50
    SetSpriteAnimationTimeline( UFO.Name, True, Timeline )

    ShowSprite( UFO.Name )

    LoadSprite( SpaceStation.Name, SpaceStation.Image )
    MoveSpriteToPoint( SpaceStation.Name, SpaceStation.X, SpaceStation.Y )
    SetSpriteAnimationTimeline( SpaceStation.Name, True, Timeline )
    ShowSprite( SpaceStation.Name )

    Var TimeLastChecked As Decimal = 0.0
    Var CurrentTime As Decimal

    While True

        CurrentTime = TickCount()

        If CurrentTime - TimeLastChecked > 50 Then

            If IsKeyDown("Right") Then
                UFO.X = UFO.X + 10
            End If


```

```

    If IsKeyDown("Left") Then
        UFO.X = UFO.X - 10
    End If

    If IsKeyDown("Up") Then
        UFO.Y = UFO.Y - 10
    End If

    If IsKeyDown("Down") Then
        UFO.Y = UFO.Y + 10
    End If

    MoveSpriteToPoint( UFO.Name, UFO.X, UFO.Y )

    If IsKeyDown("S") Then
        SpaceStation.X = SpaceStation.X + 10
    End If

    If IsKeyDown("A") Then
        SpaceStation.X = SpaceStation.X - 10
    End If

    If IsKeyDown("W") Then
        SpaceStation.Y = SpaceStation.Y - 10
    End If

    If IsKeyDown("Z") Then
        SpaceStation.Y = SpaceStation.Y + 10
    End If

    MoveSpriteToPoint( SpaceStation.Name, SpaceStation.X,
        SpaceStation.Y )

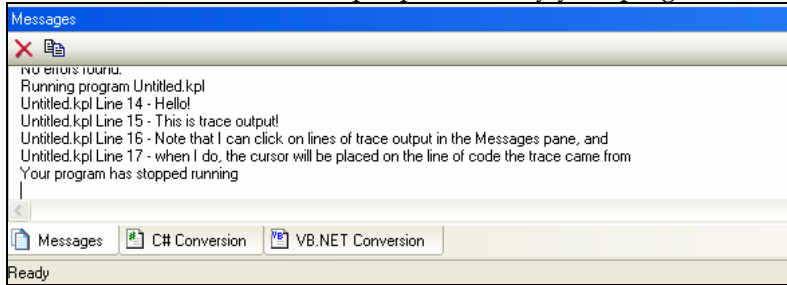
    TimeLastChecked = CurrentTime

End If
End While
End Method
End Program

```

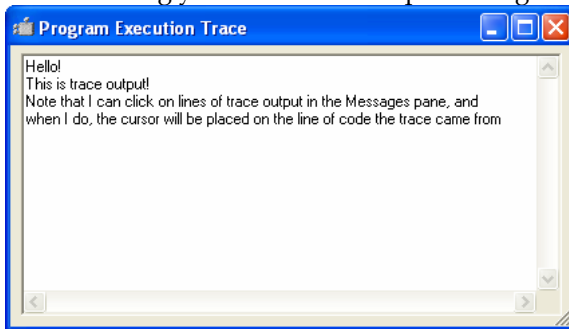

Section Four: Debugging KPL programs

Version 2 of KPL will include standard debugging and breakpoint support, to help the programmer test and debug her KPL program. Meanwhile, debugging in KPL uses the same kinds of run-time output that is typical used to debug interpreted scripts – except that the messages pane (at the bottom of the KPL IDE) makes for a convenient way to write and review, and make use of the trace output produced by your program:

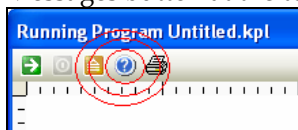


Note that you can click on lines of trace output in the Messages pane, and when you do, the cursor will be moved to the line of code the trace output came from.

You can also optionally call the **ShowTrace()** method, which will pop up a window which presents all the trace output generated by the program. The trace output in this window can be selected using your mouse and copied using **Ctrl+C**:



You can also interactively display the trace window at any time by clicking the View Trace Messages button at the top of the KPL program window – the one highlighted here:



The KPL code which generates the traces shown above is:

```
Trace("Hello!")
Trace("This is trace output!")
Trace("Note that I can click on lines of trace output in the Messages pane, and")
Trace("when I do, the cursor will be placed on the line of code the trace came from")
ShowTrace()
```

The following system methods are available in KPL for debugging:

Method Alert(Message As String, Title As String)

Pops up a window that can be used to notify the user of something important

Method ClearTrace ()

*Clears the **Program Trace Messages** window*

Method ShowTrace ()

*Displays the **Program Trace Messages** window*

Method Stop ()

Stops execution of the running program

Method Trace (Text As String)

*Sends the string stored in **Text** to the 'Program Trace' window. Use **ShowTrace()** or press the 'Trace Messages' button to view the trace window*

Section Five: Resources for KPL programmers

www.KidsProgrammingLanguage.com is the official website for KPL. The download page at that site will always offer the latest download of KPL, as well as the latest available documentation – such as this user manual. The download page will also offer new KPL programming content which has not yet been included in the KPL download. And last, it will offer the international language translation downloads – KPL volunteers have so far translated it to Spanish, Chinese, Dutch, French, German, Polish, Romanian, Italian, Swedish and Catalan. Thanks to all of our volunteers, beginners around the world can use KPL in their native language!

[Morrison Schwartz](#), the software development and consulting company that created KPL, also hosts online communities for KPL. Those online communities include all of the following areas:

[Cool KPL programs](#)

[Kids' Discussion of KPL](#)

[Parents' discussion of KPL](#)

[Teachers' discussion of KPL](#)

[KPL Questions and Answers](#)

[International Language Versions of KPL](#)

Hundreds of members participate in KPL's online community – and it really is only getting started. Join the fun! New KPL programs, questions and answers, and lots of other information are available in the online community.

We are also working on three books for KPL users – and more when those are done! First and most important is a tutorial book which will enable beginners to learn computer programming on their own, using only KPL and our tutorial book. That book is tentatively titled “**Create Your Own Computer Games with KPL!**” If you'd like to be added to a mailing list to be informed when the book is available, please email us at kpl@morrison schwartz.com.

The other books we are working are a **Teacher's Coursebook**, and a **Student's Workbook** – which are intended for use in the classroom environment – whether that classroom is at home or at school! If you'd like to be added to a mailing list to be informed when our classroom materials are available, please email us at kpl@morrison schwartz.com.

Section Six: System Methods available in KPL

The current list of supported System Methods and Functions available from KPL code, divided by category of function:

MATH FUNCTIONS

Function Abs (Value AS INT) AS Int

Returns the absolute (positive) value of the integral number stored in Value

Function Acos (d AS Decimal) AS Decimal

Returns the arccosine or inverse cosine in degrees for the cosine value specified in d

Function ArcTan (d AS Decimal) AS Decimal

Returns the angle whose tangent is specified in d

Function ArcTan2 (X AS Decimal, Y as Decimal) AS Decimal

Returns the angle in degrees that is the arctangent of Y/X

Function Asin (d AS Decimal) AS Decimal

Returns the angle in degrees that is the arcsine of the value in d

Function Ceiling (d AS Decimal) AS Decimal

Returns the smallest integer that is not smaller than the value d

Function Cos (d AS DECIMAL) AS Decimal

Returns the cosine of the angle specified in d

Function CosH (d AS DECIMAL) AS Decimal

Returns the hyperbolic cosine of the value specified in d

Function Exp (d AS DECIMAL) AS Decimal

Returns e raised to the power d

Function Floor (d AS Decimal) AS Decimal

Returns the largest integer that is not larger than the value d

Function Log (d AS DECIMAL) AS Decimal

Returns the natural (base e) logarithm of the number specified in d

Function Log10 (d AS DECIMAL) AS Decimal

Returns the base 10 logarithm of the number specified in d

Function LogBase (value AS DECIMAL, base AS DECIMAL) AS Decimal

Returns the logarithm of the value in the base of base

Function Max (ValueA AS Any, ValueB as Any) AS Any

Returns whichever of ValueA or ValueB is the larger value

Function Min (ValueA AS Any, ValueB as Any) AS Any

Returns whichever of *ValueA* or *ValueB* is the smaller value

Function Power (value AS DECIMAL, power AS DECIMAL) AS Decimal

Returns the result of *value* raised to the power of *power*

Function Random (MIN AS INT, MAX AS INT) AS Int

Returns a random Integer (whole) number in the range from *MIN* to *MAX*, inclusive. For example:

Random (1, 10) returns a random number between 1 and 10.

Function Round (d AS Decimal) AS Decimal

Returns the integer that is closest to the value *d*

Function RoundToPlace (d AS Decimal, decimalPlaces As Decimal) AS Decimal

Rounds a decimal *d* to *decimalPlaces* decimal places.

Function Sin (d AS DECIMAL) AS DECIMAL

Returns the sine of the angle specified in *d*

Function Sqrt (d AS DECIMAL) AS DECIMAL

Returns the square root of *d*

Function Tan (d AS DECIMAL) AS DECIMAL

Returns the tangent of the angle specified in *d*

Function TanH (d AS DECIMAL) AS DECIMAL

Returns the hyperbolic tangent of the value specified in *d*

Function TickCount () AS DECIMAL

Returns the number of seconds since KPL started running – useful as a timer

DATA FUNCTIONS

Function ArrayLength (List as ARRAY) as INT

Returns the length of the array stored in *List*

Function ConvertToBool (value as ANY) as BOOL

Converts *value* to a *BOOL*

Function ConvertToDecimal (value as ANY) as DECIMAL

Converts *value* to a *DECIMAL*

Function ConvertToInt (value as ANY) as INT

Converts *value* to an *INT*

Function ConvertToString (value as ANY) as STRING

Converts *value* to a *STRING*

Function FormatString (FormatString AS STRING, Value AS ANY) AS STRING

Function IndexOf (Value AS STRING, Pattern AS STRING) AS INT

Returns the position in *Value* of the first occurrence of the string stored in *Pattern*

Function ParseInt (Value AS STRING) AS INT

Converts the string stored in *Value* into an Integer value

Function Split (Text AS STRING, Delimiter AS STRING) AS ARRAY

Splits the string stored in *Text* into an array of strings. Splits *Text* at each occurrence of *Delimiter*

Function StringLength (Text AS STRING) AS INT

Returns the length of the string stored in *Text*

Function StringReplace (Text AS STRING, Pattern AS STRING, NewPattern AS STRING) AS STRING

Replaces any occurrence within *Text* of the string in *Pattern* with the string in *NewPattern*

Function Substring (Text AS STRING, StartPos AS INT, Length AS INT) AS STRING

Returns the part of *Text* starting at *StartPos*, to a maximum of *Length* letters

Function ToLower (Value AS STRING) AS STRING

Converts the string stored in *Value* to all lowercase letters

Function ToUpper (Value AS STRING) AS STRING

Converts the string stored in *Value* to all uppercase letters

DATE FUNCTIONS**Function AddDays (Date as String, NumberOfDays as Int) as String**

Returns result of *Date* value plus *NumberOfDays* days

Function AddHours (Date as String, NumberOfHours as Int) as String

Returns result of *Date* value plus *NumberOfHours* hours

Function AddMinutes (Date as String, NumberOfMinutes as Int) as String

Returns result of *Date* value plus *NumberOfMinutes* minutes

Function AddMonths (Date as String, NumberOfMonths as Int) as String

Returns result of *Date* value plus *NumberOfMonths* months

Function AddSeconds (Date as String, NumberOfSeconds as Int) as String

Returns result of *Date* value plus *NumberOfSeconds* seconds

Function AddYears (Date as String, NumberOfYears as Int) as String

Returns result of *Date* value plus *NumberOfYears* years

Function DateDifference (Date1 as String, Date2 as String) as Int

Returns an integer value representing the number of days difference between two date strings

Function Day (Date as String) as INT

Returns day of month integer from a Date string

Function DayName (DayOfWeek as Int, Abbreviated as Bool) as String

Returns local-language-specific day of week name, either abbreviated or not

Function DayOfWeek (Date as String) as Int

*Returns numeric day of week based on input **Date** string*

Function DayOfYear (Date as String) as Int

*Returns numeric day of year based on input **Date** string*

Function DaysInMonth (Year as Int, Month as Int) as Int

Returns number of days in the specified Month of the specified Year

Function FirstDayOfWeek () as Int

Returns the index of the first day of the week, based on local culture settings

Function FormatDate (Year As Int, Month as Int, Day as Int, UseLongFormat as Bool) As String

Returns a long or short format of a Date string, using the local language and culture settings

Function FormatTime (Hour As Int, Minute as Int, Second as Int, UseLongFormat as Bool) As String

Returns a long or short format of a Time string, using the local language and culture settings

Function Hour (Time as String) as INT

Returns hour integer from a Time string

Function IsLeapDay (Date as String) as Bool

*Returns **True** if specified **Date** is a leap day*

Function IsLeapMonth (Date as String) as Bool

*Returns **True** if specified **Date** is in a leap month*

Function IsLeapYear (Date as String) as Bool

*Returns **True** if specified **Date** is in a leap year*

Function Minute (Time as String) as Int

Returns minute integer from a Time string

Function Month (Date as String) as Int

Returns month integer from a Date string

Function MonthName (Month as Int, Abbreviated as Bool) as String

Returns local-language-specific month name, either abbreviated or not

Function Second (Time as String) as Int

Returns second integer from a Time string

Function TimeDifference (Time1 as String, Time2 as String) as String

Returns a time string representing the difference between two time strings, in hours, minutes and seconds

Function TimeNow () as String

Returns a time string specifying the time of day right now

Function Today () as String

Returns a date string specifying today's date

Function WeekOfYear (Date as String) as Int

Returns week of the year integer from a Date string

Function Year (Date as String) as Int

Returns year integer from a Date string

KEYBOARD FUNCTIONS

Function Confirm(Message As STRING, Title As STRING) As BOOL**Function GetKey () AS STRING**

Returns the last key that the user pressed, or an empty string ("") if no key has been pressed.

Function IsKeyDown (Key AS STRING) AS BOOL

*Returns **True** if the named **Key** is currently being pressed by the user. For example, **IsKeyDown("Right")** will return **True** if the user is pressing the right arrow key*

CONSOLE FUNCTIONS

Function ConsoleReadDecimal(Prompt As String, EchoToConsole As Bool) As DECIMAL**Function ConsoleReadInt(Prompt As String, EchoToConsole As Bool) As INT****Function ConsoleReadKey(EchoToConsole As Bool) As STRING****Function ConsoleReadLine(Prompt As String, EchoToConsole As Bool) As STRING**

CONSOLE METHODS

Method ConsoleWrite(Message As STRING)**Method ConsoleWriteLine(Message As STRING)****Method ClearConsole()****Method HideConsole()**

Method MaximizeConsole()

Method SetConsoleBackgroundColor(Color As INT)

Method SetConsoleBulletMode(IsBulletModeOn As BOOL)

Method SetConsoleFont(Name As STRING, Size As INT)

Method SetConsoleFontColor(Color As INT)

Method SetConsoleFontSize(Size As DECIMAL)

Method SetConsoleFontStyle(Bold As BOOL, Italic As BOOL, Underline As BOOL)

Method SetConsoleHangingIndent(Indent As INT)

Method SetConsoleHeight(Height As INT)

Method SetConsoleIndent(Indent As INT)

Method SetConsoleTextAlignment(Alignment As STRING)

Method ShowConsole()

SCREEN FUNCTIONS

Function ScreenHeight () AS INT

Returns the height of the KPL screen, in pixels

Function ScreenWidth () AS INT

Returns the width of the KPL screen, in pixels

SCREEN METHODS

Method BeginFrame ()

Causes KPL to not refresh the screen until RefreshScreen() is called

Method Clear ()

This method clears the screen

Method Delay (Milliseconds AS INT)

*Causes the running KPL program to wait for **Milliseconds** milliseconds.*

Method Circle (Size AS INT, Filled AS BOOL)

Draws a circle on screen using the current pen location and pen color

Method Maximize ()

Makes the running KPL program take up the whole screen

Method Print (text AS STRING)

*Prints the contents of **text** at the location of the drawing pen, using the current pen color. Numeric variables, if passed to this method, will be automatically formatted and printed as strings.*

Method PrintInRect (text AS STRING, X AS INT, Y AS INT, Width AS INT, Height AS INT)

*Prints the contents of **text** within the specified rectangle. Words will be wrapped to fit horizontally, and cropped vertically if they go beyond the bottom of the rectangle. Numeric variables, if passed to this method, will be automatically formatted and printed as strings.*

Method PrintLine (text AS STRING)

*Prints the contents of **text** at the location of the drawing pen, using the current pen color, then adds a **Carriage Return** to move the pen to the next line. Numeric variables, if passed to this method, will be automatically formatted and printed as strings.*

Method RefreshScreen ()

Causes KPL to refresh all sprites and graphics on the screen

Method ScrollBackground (X AS INT, Y AS INT, Width AS INT, Height AS INT, ScrollX AS INT, ScrollY AS INT)

*Scrolls a rectangular background region defined by **X**, **Y** as the upper left corner, and the specified **Width** and **Height**. **ScrollX** and **ScrollY** specify the amount to scroll along each axis.*

Method SetAlgebraCoordinates ()

Causes KPL to use algebraic coordinates, with the (X, Y) origin of (0, 0) in the center the KPL window

Method SetDeviceCoordinates ()

Causes KPL to use computer device coordinates (X, Y) origin of (0, 0) in the upper left of the KPL window

Method SetFont (FontName AS STRING, Size AS INT, Bold AS BOOL, Italic AS BOOL, Underline AS BOOL)

Sets the current font used for printing text

Method SetScreenSize (Width as INT, Height as INT)

*Causes KPL to set the runtime screen size to **Width** and **Height***

Method Status (Text AS STRING)

*Displays the string value stored in **Text** in the status bar at the bottom of the running program's window*

Method TileBitmap (SpriteName AS STRING, X AS INT, Y AS INT, Width AS INT, Height AS INT, SourceX AS INT, SourceY AS INT)**Method TileSprite (SpriteName AS STRING, X AS INT, Y AS INT, Width AS INT, Height AS INT, OffsetX AS INT, OffsetY AS INT)**

PEN METHODS

Method Circle (Diameter AS INT, Filled AS BOOL)

Draws an circle on screen using the current pen location and pen color

Method Color (Color AS INT)

Sets the color of the drawing pen using color constants, which can be specified either by name or by number as shown. Examples: Color (Blue) or Color (10).

Transparent = 0	DodgerBlue = 42	MediumOrchid = 84
AliceBlue = 1	Firebrick = 43	MediumPurple = 85
AntiqueWhite = 2	FloralWhite = 44	MediumSeaGreen = 86
Aqua = 3	ForestGreen = 45	MediumSlateBlue = 87
Aquamarine = 4	Fuchsia = 46	MediumSpringGreen = 88
Azure = 5	Gainsboro = 47	MediumTurquoise = 89
Beige = 6	GhostWhite = 48	MediumVioletRed = 90
Bisque = 7	Gold = 49	MidnightBlue = 91
Black = 8	Goldenrod = 50	MintCream = 92
BlanchedAlmond = 9	Gray = 51	MistyRose = 93
Blue = 10	Green = 52	Moccasin = 94
BlueViolet = 11	GreenYellow = 53	NavajoWhite = 95
Brown = 12	Honeydew = 54	Navy = 96
BurlyWood = 13	HotPink = 55	OldLace = 97
CadetBlue = 14	IndianRed = 56	Olive = 98
Chartreuse = 15	Indigo = 57	OliveDrab = 99
Chocolate = 16	Ivory = 58	Orange = 100
Coral = 17	Khaki = 59	OrangeRed = 101
CornflowerBlue = 18	Lavender = 60	Orchid = 102
Cornsilk = 19	LavenderBlush = 61	PaleGoldenrod = 103
Crimson = 20	LawnGreen = 62	PaleGreen = 104
Cyan = 21	LemonChiffon = 63	PaleTurquoise = 105
DarkBlue = 22	LightBlue = 64	PaleVioletRed = 106
DarkCyan = 23	LightCoral = 65	PapayaWhip = 107
DarkGoldenrod = 24	LightCyan = 66	PeachPuff = 108
DarkGray = 25	LightGoldenrodYellow = 67	Peru = 109
DarkGreen = 26	LightGreen = 68	Pink = 110
DarkKhaki = 27	LightGray = 69	Plum = 111
DarkMagenta = 28	LightPink = 70	PowderBlue = 112
DarkOliveGreen = 29	LightSalmon = 71	Purple = 113
DarkOrange = 30	LightSeaGreen = 72	Red = 114
DarkOrchid = 31	LightSkyBlue = 73	RosyBrown = 115
DarkRed = 32	LightSlateGray = 74	RoyalBlue = 116
DarkSalmon = 33	LightSteelBlue = 75	SaddleBrown = 117
DarkSeaGreen = 34	LightYellow = 76	Salmon = 118
DarkSlateBlue = 35	Lime = 77	SandyBrown = 119
DarkSlateGray = 36	LimeGreen = 78	SeaGreen = 120
DarkTurquoise = 37	Linen = 79	SeaShell = 121
DarkViolet = 38	Magenta = 80	Sienna = 122
DeepPink = 39	Maroon = 81	Silver = 123
DeepSkyBlue = 40	MediumAquamarine = 82	SkyBlue = 124
DimGray = 41	MediumBlue = 83	SlateBlue = 125

SlateGray = 126
Snow = 127
SpringGreen = 128
SteelBlue = 129
Tan = 130

Teal = 131
Thistle = 132
Tomato = 133
Turquoise = 134
Violet = 135

Wheat = 136
White = 137
WhiteSmoke = 138
Yellow = 139
YellowGreen = 140

Method ColorRGB (Red AS INT, Blue AS INT, Green AS INT)

Sets the color of the drawing pen using combined intensities of Red, Green and Blue. Examples: ColorRGB (0, 0, 0) is black, and ColorRGB (255, 255, 255) is white.

Function DrawLine (StartX AS INT, StartY AS INT) to (EndX AS INT, EndY AS INT)

Draws a line on the screen from (StartX,StartY) to (EndX,EndY), in the current pen color

Method Ellipse (Width AS INT, Height AS INT, Filled AS BOOL)

Draws an ellipse on screen using the current pen location and pen color

Method MoveBy (DeltaX AS INT, DeltaY AS INT)

Moves the drawing pen/cursor by the specified amounts along the x and y axes

Method MoveTo (X AS INT, Y AS INT)

Moves the drawing pen/cursor to the (X, Y) location

Method Pen (IsPenDown AS BOOL)

Controls whether the drawing pen is 'down' and thus whether it will draw as it is moved

Method PenWidth (PenWidth AS INT)

Specifies the drawing pen's drawing width in pixels

Method PutPixel (X AS DECIMAL, Y AS DECIMAL)

Puts a pixel on the screen at the coordinates (X, Y), in the current pen color

Method Rectangle (Width AS INT, Height AS INT, Filled AS BOOL)

*Draws a rectangle in the current pen color, starting at the current pen position, **Width** pixels wide and **Height** pixels high. If **Filled** is **True**, the rectangle will be filled.*

SPRITE FUNCTIONS

Function GetSpriteFrameCount (SpriteName AS STRING) AS INT

Returns the number of frames that can be shown for the sprite

Function GetSpriteHeight (SpriteName AS STRING) AS INT

Returns the current height of the named sprite's bounding box

Function GetSpriteLeft (SpriteName AS STRING) AS INT

Returns the X position of the named sprite's upper-left corner

Function GetSpritesThatIntersectWith (SpriteName AS STRING) AS UNKNOWN[]

Returns a string array with the names of all sprites that are touching (intersecting) with the named sprite

Function GetSpriteTop (SpriteName AS STRING) AS INT

Returns the Y position of the named sprite's upper-left corner

Function GetSpriteWidth (SpriteName AS STRING) AS INT

Returns the current width of the named sprite's bounding box

Function SpriteContainsPoint (SpriteName AS STRING, X AS INT, Y AS INT) AS BOOL

*Returns **True** if the point specified by (X,Y) is occupied by the sprite named by **SpriteName***

Function SpritesIntersect (SpriteName1 AS STRING, SpriteName2 AS STRING) AS BOOL

SPRITE METHODS

Method ClearSprites ()

Clears all sprites from the screen

Method HideSprite (SpriteName AS STRING)

Causes the named sprite to no longer be visible

Method FlipSpriteHorizontally (SpriteName AS STRING)

Flips the displayed Sprite image, right to left

Method FlipSpriteVertically (SpriteName AS STRING)

Flips the displayed Sprite image, top to bottom

Method LoadSprite (SpriteName AS STRING, FileName AS STRING)

*Creates a new sprite whose name is specified by **SpriteName**. The sprite's picture will be loaded from **Filename**.*

Method MoveSpriteBy (SpriteName AS STRING, DeltaX AS INT, DeltaY AS INT)

*Moves the named sprite by adding **DeltaX** to the sprite's current X value, and adding **DeltaY** to the sprite's current Y value.*

Method MoveSpriteTo (SpriteName AS STRING, X AS INT, Y AS INT)

Moves the named sprite so that it's upper-left corner will be at (X, Y)

Method MoveSpriteToPoint (SpriteName AS STRING, X AS INT, Y AS INT)

Moves the named sprite so that it's upper-left corner will be at (X, Y)

Method RotateSprite (SpriteName AS STRING, RotationAmount AS STRING)

*Rotates the named sprite. **RotationAmount** is the angle (in degrees) to rotate the sprite*

Method RotateSpriteBy (SpriteName AS STRING, AmountOfChange AS STRING)

*Rotates the named sprite by changing it's current rotation by the amount specified in **AmountOfChange**, which is in degrees*

Method ScaleSprite (SpriteName AS STRING, ScaleFactor AS DECIMAL)

*Sets named sprite's size by scaling it to the indicated **ScaleFactor** (as a percentage)*

Method SetSpriteActiveFrame (SpriteName AS STRING, FrameNumber AS INT)

*Sets the active frame for the sprite. **FrameNumber** must be less than or equal to the number of available frames*

Method SetSpriteAnimationTimeline (SpriteName AS STRING, AutoRewind AS BOOL, Timeline AS ARRAY)

*Begins automatic animation of the named sprite by setting an **INT[]** array, where each element of the array specifies how long to show the frame with the same index as the element*

Method SetSpriteCanCollide (SpriteName AS STRING, CanCollide AS BOOL)

Sets whether the named sprite can collide (intersect) with other sprites. This is useful for background sprites where collision information is not needed

Method SetSpriteClip (X AS INT, Y AS INT, Width AS INT, Height AS INT)

Sets a clipping region for the sprite which prevents any of it from being drawn outside of the specified rectangle

Method SetSpriteOpacity (SpriteName AS STRING, OpacityPercent AS INT)

*Sets how transparent the named sprite is. An **OpacityPercent** of 100 means that the sprite is fully visible, and a value of 0 means that it is completely invisible*

Method SetSpriteZIndex Method)

Sets the named sprite's Z-Index, which is the order in which a sprite is drawn. A sprite with a higher Z-Index will always be drawn on top of sprites with a lower Z-Index

Method ShowSprite (SpriteName AS STRING)

*Makes the sprite whose name is specified by **SpriteName** visible*

Method StampSprite (SpriteName AS STRING)

Causes the named sprite to be drawn permanently on the background canvas

Method StopSpriteAnimation (SpriteName AS STRING)

Stops the named sprite from being automatically animated. Used to stop animation after a call to

Method SetSpriteAnimationTimeline()

Method StretchSprite (SpriteName AS STRING, ScaleFactorX AS DECIMAL, ScaleFactorY AS DECIMAL)

*Stretches the displayed image of a sprite using the specified **ScaleFactors** (as percentages)*

Method UnloadSprite (SpriteName AS STRING)

*Unloads the sprite whose name is specified by **SpriteName***

SOUND METHODS

Method PlaySound (FileName AS STRING)

Plays the sound file specified by Filename

TRACE AND RUNTIME METHODS

Method Alert(Message As STRING, Title As STRING)

Pops up a window that can be used to notify the user of something important

Method ClearTrace ()

*Clears the **Program Trace Messages** window*

Method ShowTrace ()

*Displays the **Program Trace Messages** window*

Method Stop ()

Stops execution of the running program

Method Trace (Text AS STRING)

*Sends the string stored in **Text** to the 'Program Trace' window. Use **ShowTrace()** or press the 'Trace Messages' button to view the trace window*