

HyperNext Studio

QuickStart

updated 27th November 2006

HyperNext Studio is an easy to use software development system that allows almost anyone to quickly create and run their own software. The HyperNext Creator can build both standalone applications as well as stacks that run on our freeware HyperNext Player.

HyperNext is cross-platform and runs on Windows XP, Macintosh OS X and OS 9. Furthermore, each version of HyperNext builds software for ALL the other platforms.

A companion to HyperNext Creator is HyperNext Developer, a visual development system that produces plug-ins for Creator. Plugins are self-contained programs (libraries) that bring extra functionality to the HyperNext programming language.

This QuickStart is constantly being improved so please keep checking our web site for newer versions. The latest QuickStart can be found on our downloads page.

Contents

1. Introduction.
 - a. HyperNext Stacks.
 - b. QuickStart Organization.
 - c. Programmer's Guide.

2. HyperNext Creator.
 - a. Creator Modes.
 - b. Layout
 - c. Key Shortcuts

3. Creator Lessons
 1. Your first card.
 2. Multiple cards.
 3. Adding controls to cards.
 4. Your first HyperNext script.
 5. Variables, Text, Messages and Speech.
 6. Fields and text.
 7. Building a stack.
 8. Building a standalone.

4. HyperNext Developer
 1. Your first plug-in.
 2. Testing a plug-in using Creator.
 3. Plug-in global variables.
 4. Preventing plug-in clashes.

5. Using AppleScript
6. Using REALbasic Script

1 Introduction

Welcome to HyperNext Studio, a cross-platform visual software development system aimed at anyone interested in creating their own software. The heart of HyperNext Studio is a programming language called HyperNext and so often the programming environment is simply referred to as HyperNext. The HyperNext Studio development system currently runs on Macintosh OS X and OS 9, and Microsoft Windows XP, and each version can build software for all platforms.

The HyperNext language was initially aimed at those who had no previous experience of programming but is powerful enough to be used by experienced programmers wishing to rapidly produce an application or prototype. HyperNext has now been combined with neural network technology and can quickly produce neural network applications.

HyperNext Studio is comprised from three complementary applications : Creator, Developer and Player. HyperNext Creator produces software in the form of standalone applications and stacks that run on our freeware HyperNext Player. HyperNext Developer builds plugins for Creator so enabling its functionality to be extended. The HyperNext Creator interface is much easier to understand than most modern programming environments as it has just one Design window and a simple Tool Bar. There is also a floating window called the Mode Switcher which allows switching between the Design, Preview and Run modes. Its really quite simple and not as daunting as "professional" software development systems.

The software structure in HyperNext software is like a set of linked cards or windows, with cards holding buttons, fields, canvases and multimedia. Its

HyperNext programming language is easy to learn having English-like commands and with variables not requiring their type or structure be declared.

a) HyperNext Stacks

HyperNext Creator can produce software called stacks. Stacks are like documents but are smarter as they can actively control your computer, interact with the user and process data. Unlike simple word processor documents stacks can only be run using our freeware cross-platform HyperNext Player. The HyperNext Player currently works on Macintosh OS X and OS 9, and Microsoft Windows XP.

b) QuickStart Organization

The first half of this QuickStart covers the basics of creating software using the Creator while the second half introduces plugin development using the Developer. If you are interested in producing plugins then the section on Creator usage will be helpful in understanding the Developer because they share similar functionality.

Chapters in this QuickStart are made up of single lessons, with each lesson having two main parts, a theory section and a practical section. The theory explains some aspect of HyperNext and the practical section gives a step by step guide to implementing some aspect of it. Sometimes the theory might seem quite involved in which case trying the practical exercise should clarify it. Using HyperNext is meant to be fun, so try not to get stuck on theory, often simply messing about and experimenting with it will enable you to make the most progress.

c) Programmer's Guide

Even if you are already an experienced programmer it is probably a good idea to at least browse the following lessons. You can also try the Help Guide built into both the Creator and Developer and which can be accessed from the Guide menu that resides next to the standard Help menu. There is also a Language Reference PDF document bundled with HyperNext Studio that greatly expands upon the built-in help.

This QuickStart covers the main aspects of HyperNext Creator, including its Graphical User Interface.

The built-in Guide has an introduction to HyperNext and briefly covers cards and controls. It also has sections on different aspects of the HyperNext programming language along with details of over 800 commands and functions. There are also some programming examples to help clarify the syntax and usage of many HyperNext keywords.

The Online-Guide is currently being improved and the latest version can be download from our web site. After decompressing it should be dropped into the data folder of the HyperNext directory.

2 HyperNext Creator

The Creator allows you to design and build software in a manner analogous to drawing a picture using a paper and pen - the Creator provides a surface for you to place controls upon, controls such as buttons and fields. It also enables a control's attributes to be changed, plus a way of previewing the layout and of testing/running your software creation.

Below are shown the main areas of control in the Creator when it is in design mode. The [Card Layout](#) is where controls are placed and edited. The [Tool Bar](#) can be used to create or delete controls, edit their properties and select different cards. The [Mode switcher](#) can switch between Design, Preview and Run modes.

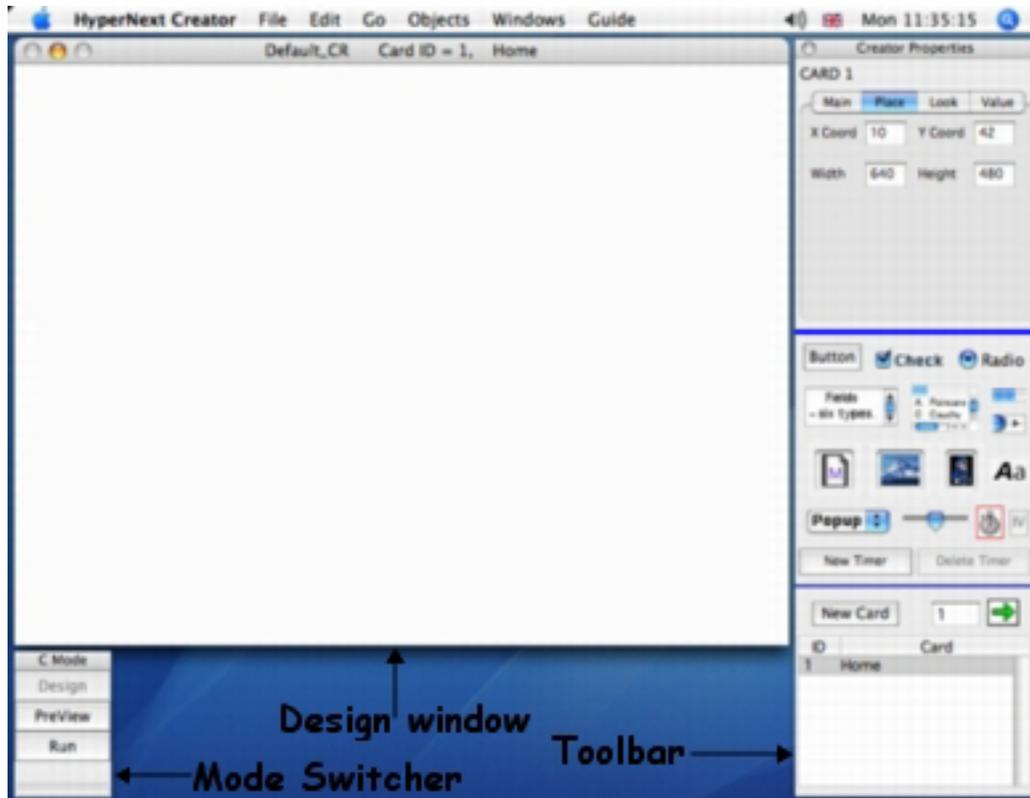
a) Creator Modes

The Creator has three main modes which can be quickly switched between using either the Mode switcher or else keyboard shortcuts listed below.



(a) Design Mode

Most of your time will be spent in Design mode as this is where controls such as buttons fields etc can be placed onto card, and can then have their sizes, colours and scripts changed. In Design mode the script editor, menu editor and multi-media libraries can also be called up.



(b) Preview Mode

The Preview mode shows how the currently selected card will appear at runtime. Although the actual controls do not work in Preview mode their

appearance is realistic and can help one judge if the layout is right. The Preview mode can also be accessed from the menu or by using the Command + "D" keys.

(c) Run Mode

The Run mode enables your software to be run and tested within the Creator. Except for Apple Events, all other functions work as if in the final application so enabling many problems to be ironed out before the final stack or application is built. When Run mode is started, your work will automatically be saved, the compiler will check for errors and if none are found the program will be run. Run mode can also be entered either from the menu or via the Command + "R" keys. To return to Design mode simply use the Mode switcher or either the "Quit" menu option or the Command + "Q" keys.

Key Shortcuts

In the Creator some shortcuts using the Command Key(CMD) plus another key are:

CMD+D - Preview(display) current card (to revert back use CMD+D)

CMD+K - Compile and test stack - it does not run the stack.

CMD+R - Run the stack. To end the run simple use CMD+Q

CMD+M - Open the Main code for editing.

CMD+Q - Quit Creator or end a run and return to Creator.

CMD+S - Save

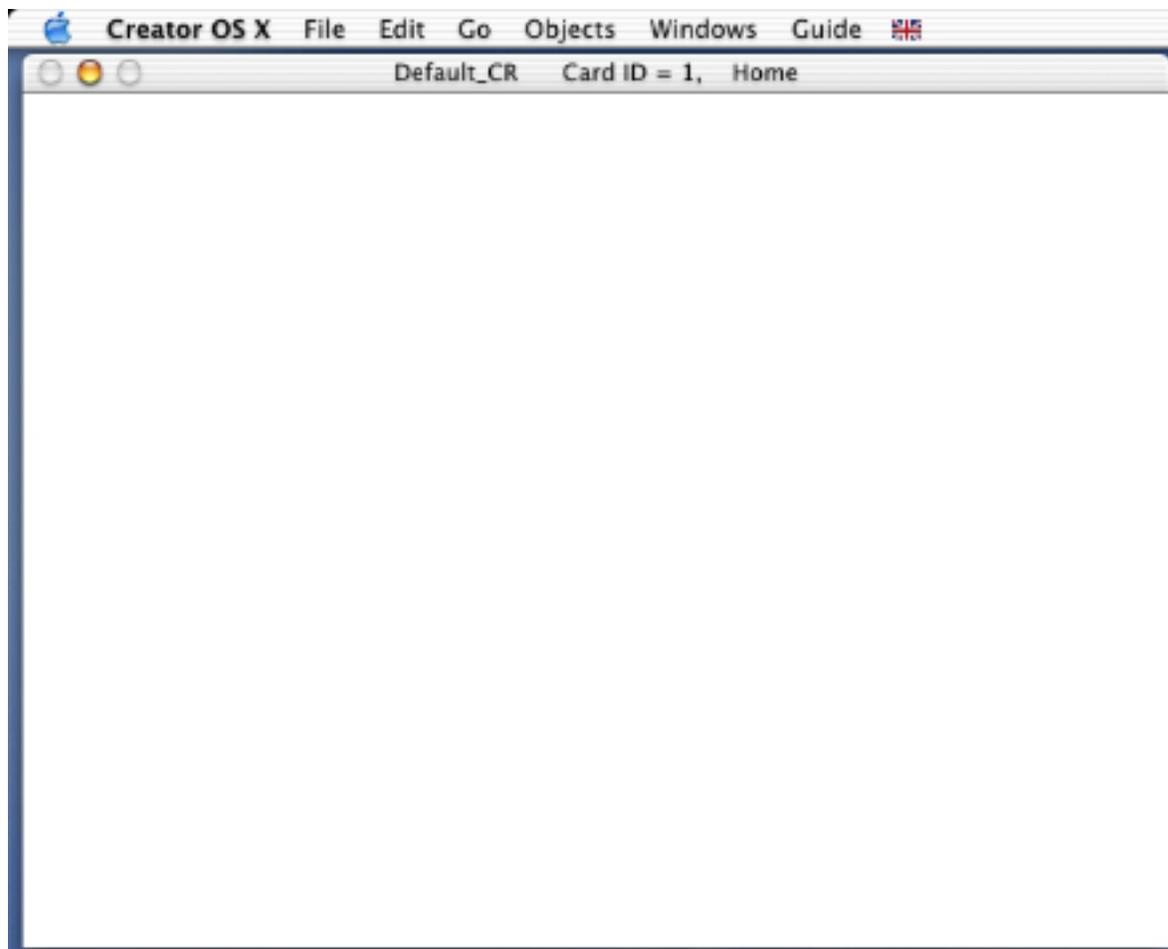
On the Windows platform replace the "CMD" key with the "Control" key.

b) Creator Layout

As mentioned above, in Design mode the Creator layout has just two main windows, the Card window and the Tool Bar.

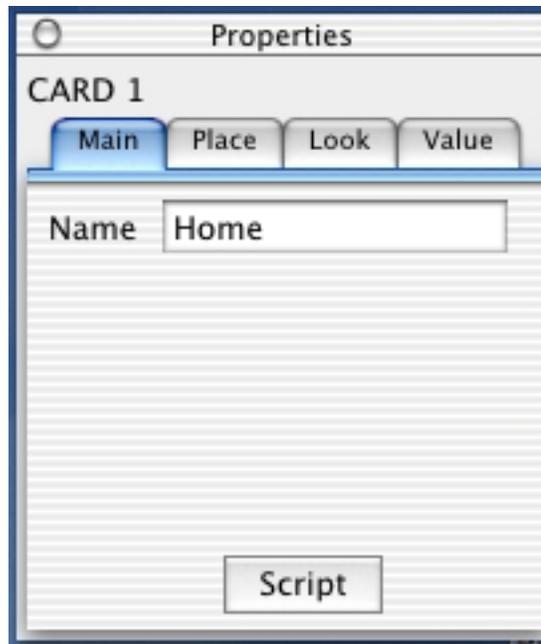
(a) Card Window

The Card window is where cards are created and have controls placed upon them. Once controls are placed upon the Card window they can be dragged around and resized.



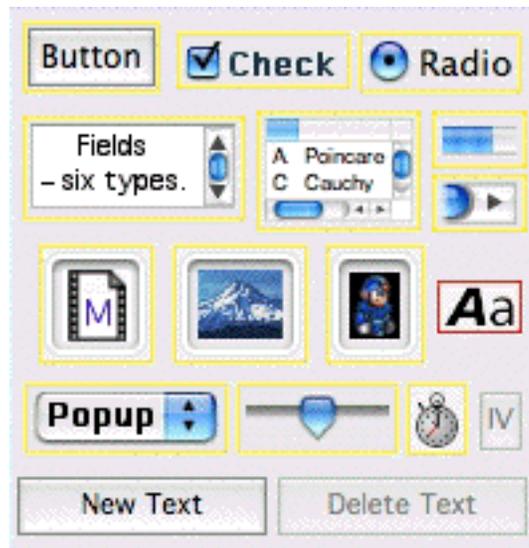
(b) Tool Bar - Properties Section

The Properties section enables the properties of cards and controls to be changed and its appearance depends upon the item currently selected. Controls can have their location and size set more accurately and quickly using the Properties section than by simply physically dragging/stretching them.



(b) Tool Bar - Controls section

The control's section allows the type of control to be selected, for instance, button, field etc. To make a new button control simply select the control icon for button, select the New Button button, then simply click anywhere on the Card window and a default sized button will appear there. To change the button's properties just select the button by clicking on it and then use the Properties section of the Tool Bar window to make any necessary changes.



(c) Tool Bar - Card Chooser

The Card chooser section window allows cards to be created and then selected for editing. When there are many cards it is often easier to select the card using the green goto button - simply enter the card's number into the nearby field and click the goto button.



3 Creator Lessons

Here you will learn about cards, what they are and how you can use them. You will also learn about controls, how to create and then modify them to fit your needs. Later on you will learn about HyperNext programming or scripting as it is sometimes termed.

Remember, HyperNext is great for creating software that has a stack structure where a stack is just a collection of one or more cards or windows. It is important to note that in a HyperNext stack the first card is the Home card and cannot be deleted. The Home card is important because when a stack initially runs the Home card is loaded first. During most of these lessons the Home card is used as the main card for experimenting with controls and their associated scripts.

Lesson 1 - Your first card

In this first lesson you will see how easy it is to produce a card with one button on it. The button does nothing special, it just beeps when pressed. However, this lesson will demonstrate how to use the Card windows and various sections of the Tool Bar.

Theory - What are cards?

A card is actually a window and in some ways is like a piece of paper because initially it is blank and ready for controls to be placed upon it. The Creator enables you to select different types of control such as button, field, canvas and then allows them to be placed upon the card. Once placed, controls can have their properties changed, properties such as size, appearance and scripting.

Cards can also have properties such as location and size, and every card has a start script that is executed when the card is loaded.

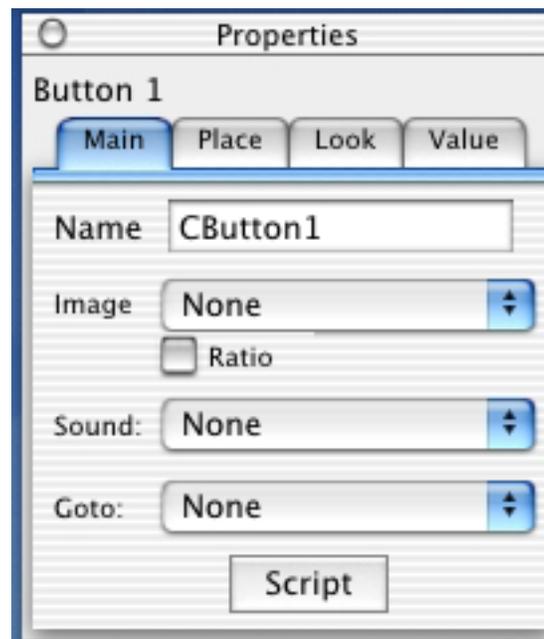
When a stack is running only one card can be visible or in focus at any one time. This allows a user to access this card but prevents them accessing out of focus cards. It is easy to change cards either by setting a GotoCard command on a button or else by placing a script in a button, field, canvas, menu etc.

Generally each card will be given a specific function such as to receive customer data, to act as preferences, manage file operations etc. Some types of stack though have many cards which are duplicates of one basic card, as for instance in a database, where each card stores one record. In

HyperNext it is very easy to produce a stack comprising two basic cards and then at runtime allow the user to create new duplicate cards as required.

Practice - Building a card

- (1) Open the Creator application and select "New" from the File menu. When prompted for the file name just enter "First" and save the new project folder in a suitable location. Note, the newly created project will be called "First".
- (2) We will now create a button and place it on the card. From the Control area of the Tool Bar select the Button control although by default it should already be activated, then click the "New Button" button and click anywhere on the card to place the new button.
- (3) Click on the newly created button to select it and look at the Property area of the Tool Bar - the properties displayed there refer to this button.



- (4) Try moving the button about by dragging it and then by entering new coordinates into its Properties using the Place tab.
- (5) We will now enter some text for the button's text indicator. In the Property section select the Tab entitled Value and in the large field enter the text "Beep"
- (6) By default text button is left aligned but for most buttons it looks best when centralized. Therefore in the Property area select the Appearance tab and click the center radio button.
- (7) To make the button beep we have to add the "Beep" command to the button's script handler. To do this, click the "Script" button on the Property area the editor will open. Type the command Beep into the script and close the editor window. Note, clicking the close button automatically saves any changes to memory.
- (8) Run the project using the Mode switcher or else from the "Go" menu or else by the keyboard shortcut CMD + "R". The project should now run and display the card with one button in it.
- (9) Click the button and it should beep.
- (10) To stop the program running simply use the Mode switcher "Design" button or else press the keyboard shortcut CMD + "Q". This will return you to the Creator designer.

Congratulations, you have just created your first card and object.

Lesson 2 - Multiple cards

In this lesson you will see how easy it is to produce a stack comprising multiple cards that allows the user to switch between the cards.

Theory - Why use multiple cards?

Most software needs more than one window, even a simple card game with one main window for the playing cards probably still needs some way of allowing the user to enter preferences, display rules information, and perhaps to display a high score table. In HyperNext only one card can be visible at a time and so the easy way to make the windows for the above game is to have one card for each function - playing window, preferences, rules etc.

Once you have designed the cards in your stack then a method is needed to hide one card and replace it by another. HyperNext uses the GotoCard command and its several variations such as GotoNextCard, GotoPriorCard etc.

Practice - Switching between cards

Use the project from the previous lesson. To load it just double click the project file called "First.prj" which will be located in the "First" folder. You could also use "Open" command from the file menu.

(1) Create a new card by clicking on the New Card button in the Card window.

A new card will appear in the card list and will also become the current card in the Design mode. The heading of both the Card and Properties area should reflect this.

(2) Create a new button on this second card by using the "New Button" button located in the Controls area.

- (3) We will now edit this button so that it goes to the home card when pressed. To do this, select the button by clicking it, then in the Properties area select the "Goto" popup menu and choose "1 Home" as the destination.
- (4) Switch back to the first card by selecting it from the card list of the Card chooser area.
- (5) Next, select the button on the current card and in the Properties area use the Goto popup menu to tell the button to goto "2 Card 2".
- (6) Run the project using CMD + "R" and the first card will appear.
- (7) Click the lone button and it should cause the second card to appear.
- (8) On the second card, click the lone button and it should cause the first card to reappear.

Lesson 3 - Adding controls to cards

In the previous lesson you learnt how to set a button so that it would take you to another card but the cards themselves looked indistinguishable except for their buttons. In this lesson you will add a text object to each card that make it easier to tell where you are.

Theory - What are controls

Controls are objects which can be placed upon a card and allow the user to interact with the card (program). Controls have properties that define their size and appearance, as well as their behaviour. There are currently several types of control in HyperNext, they are Button, Field, Canvas, Text, CheckBox, Radio Button, Popup Menu, Scroll bar, Slider, Progress bar, Listbox, Movie, Sprite Surface and Timer. Just a brief description of each is given in this lesson but further details are presented later on in this QuickStart and in the built-in help of the Creator/Developer.

- (1) **Buttons** are the most obvious type of on/off control in HyperNext and can have either show a text or an image to indicate their function. However, buttons can also be hidden or deactivated - when deactivated they are still visible but cannot be pressed. They can also have both a sound and a Gotocard connected to them via the Properties area. They can be made much more powerful though by assigning them a script which is activated only when the button is pressed. Note, adding the Gotocard via the Properties area disables a button's script.
- (2) **Fields** have many uses as they can accept text from users either via the keyboard or via drag and drop. They can also receive text from another control or from a script, for instance when a button is pressed. Fields can

be made read-only which prevents users from changing the text within them. When in read-only mode they receive mouse events and these mouse events can trigger the field's script. As detailed later there are six types of field ranging from single line with no border to multi-line scrolling with a border.

(3) **Canvases** are the most versatile control in HyperNext as they can be set up to provide a wide range of functionality. For instance at their most basic they can display an image, perhaps via the GUI, via drag and drop or else from a script that might copy it to them or else load it from a file. Canvases can also receive mouse events and can be made to act like graphical buttons. The mouse coordinates within the canvas can be used to make complex area type buttons.

(4) **Texts** are generally used just to display a header or title to indicate the function of another control although they can though be used dynamically as a counter or indicator.

(5) **Movies** provide a way of allowing the stack or user to present movie/sound/mp3 either via the GUI at design time or else via drag and drop.

(6) **CheckBoxes** have two states and allow the user to initiate an action or set a state by simply clicking the checkbox.

(7) **Radio Button** are sometimes referred to as option buttons and allow the user to select one item in a group and have the other group items automatically deselected.

(8) **Popup** menus allow the user to select one item from many and are especially useful when a list of items is large as for instance when selecting

a font name.

(9) **ScrollBars** allow the user to control the position or value of some other object such as when scrolling continuous text or a picture, or changing a numeric value. The position of the control can be changed by dragging the central bar, clicking anywhere in the scrollbar to make it jump or to click the end arrows for smooth movement. HyperNext supports both horizontal and vertical scrollbars.

(10) **Sliders** are similar to scrollbars except they do not have end arrows and are usually just dragged to change their output value in a more abrupt fashion.

(11) **ProgressBars** give a visual indication of a task's progress. They can also be clicked on to change their value and can be made to display the Barber Pole when a task's state is indeterminate.

(12) **Listboxes** are used to display one or more columns of information. Their contents can be sorted and items can have checkboxes.

(13) **Sprite Surfaces** allow sprite animations to be easily created. Sprites are graphic objects which are controlled automatically by the sprite surface and do not need constant control by the user's program. They are especially useful for making platform type games and other animations where smooth interactive movement is needed.

(14) **Timers** call their handler/script at times determined by their period setting. They are the only control that is invisible to the user. Card timers should not be confused with the Main Timer which is accessed via the menu bar.

All of the above objects have sizes, locations etc, they are often self evident and can be played around with to see the effects.

Practice - Controls and their properties

Here we will add a text control to each of the cards and then change their properties.

- (1) If it is not already open, open the project from the previous lesson and go to the home card.
- (2) From the Control area select the Texts button, click on the New Text button and then click anywhere within the Card window to place the text object.
- (3) By default Texts have no initial value and so will be invisible at preview or runtime therefore click on the Text to select it. Now in the Properties area select the Value tab and enter "Home Card" into the empty field.
- (4) Preview the card by pressing CMD + "D" and the Text should become visible. Now cancel the Preview. Increase the Text's size by clicking in the small right hand lower corner of its drag outline and then drag until the size seems reasonable. Try the Preview again and repeat until the Text looks right.
- (5) Just as for a button, the size and location of a text object can be changed using the Properties window. Some other properties include the colour of the text, the font, the text size, the style and the alignment. By using the Properties window and the Preview it is easy to experiment with a Text's appearance.
- (6) Goto the second card and create a Text object on it, repeating the above steps but with a suitable text value, perhaps "Card 2".

(7) Run your project and use the buttons to move between the cards. See how these simple text aids identify which card is in focus.

The simple property changes you have practised above can be applied to other control types - selecting the object, changing some property, then Previewing the card and repeating until things look right.

Lesson 4 - Your first HyperNext Script

In this lesson you will see how easy it is to write your first HyperNext script. Actually you wrote a simple script earlier when making a button beep but the script here is dynamic in nature. The script will be attached to a button and will simply cause a new card to be loaded when a button is pressed. Although in the second lesson you achieved this by using the Goto popup menu via the Properties area, that method was very restrictive and prevented the button executing its script.

Theory - What are scripts?

Scripts, programs and code are terms frequently used interchangeably and often seem to mean the same thing. However, in HyperNext, the term program refers to the whole stack and its related scripts. Scripts on the other hand refer to the set of instructions attached to a control such as a button, a field or even a card. These instructions are English-like and tell the computer to do something. Another term for an instruction is a statement, and statement is the preferred terminology here when talking about scripts.

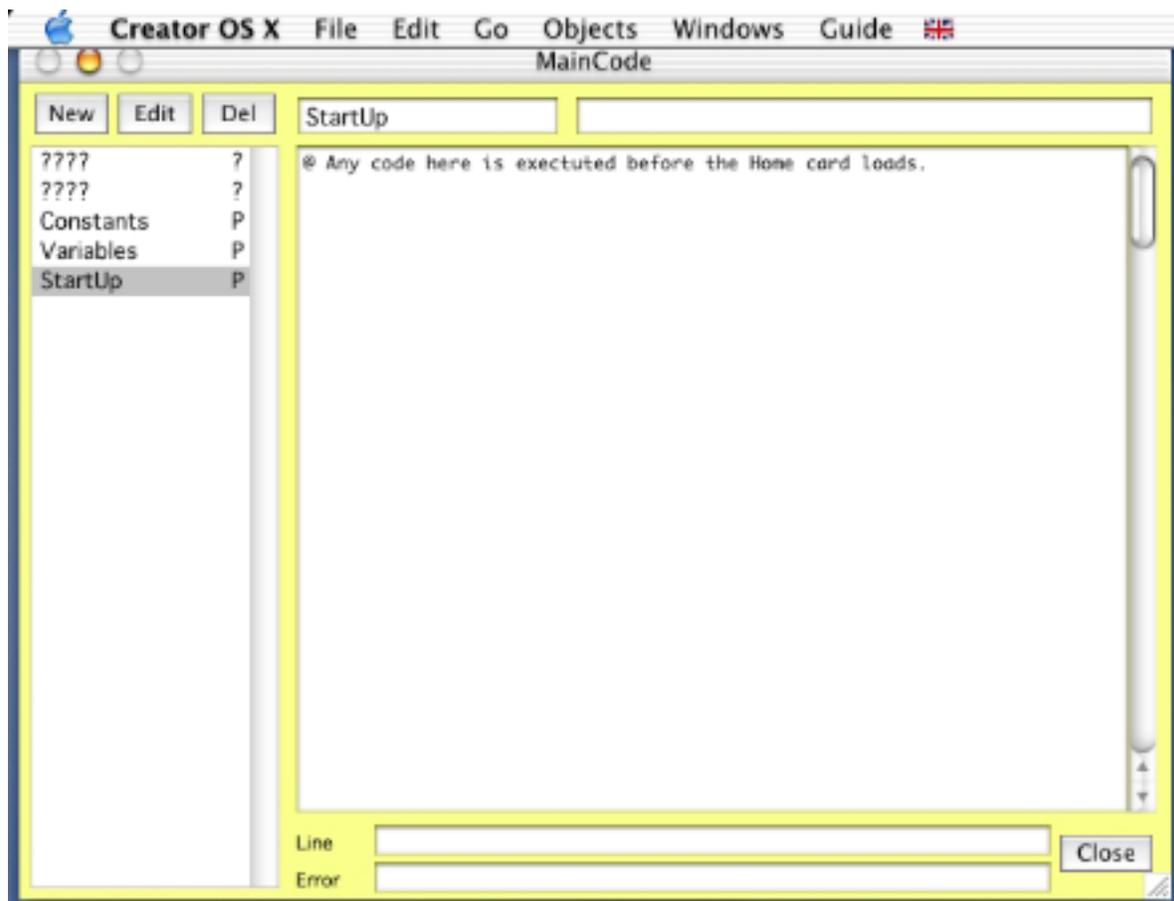
Some scripts might have hundreds or even thousands of statements and so to make it easier to both understand and find errors in the script, the script will be broken down into a number of units called procedures or subroutines. A procedure is a set of self contained statements that are grouped together and are given a name, for instance "DrawBoat". When "DrawBoat" is called from another procedure then the statements making up the "DrawBoat" procedure will be executed.

As an example, consider a button that when pressed must extract some data from a field, process it, then put the results into another field. In the main part of the button's script there will probably be three calls to

procedures having names like "GetData", "ProcessData", and "PutData" . Each of these procedures will probably be defined within the button's script in which case they will not be accessible by other controls but they could also be defined as global procedures in HyperNext's Main Code section so that they could be accessed from anywhere within the program. Using descriptive names for procedures is important because if six months later you come back to examine the button's script it will be clear to you what is happening.

Practice - Your first script

Here you will learn how to put a simple script into a button in order to make it switch cards. To do this you will use the script editor as shown below.



- (1) Assuming the previous lesson's project is open and the home card is in focus. Select the lone button and from the Properties window click the "Script" button, this will open the script editor.
- (2) The script editor is a very simple text editor with an area for typing in text. On the left hand side there is the list of handlers. For this button there should only be one handler listed, the Action handler, and in the text area you should see the command "Beep".
- (3) Add the statement "GotoCard 2" after the Beep statement and then close the script editor by using the Close button in the lower right. The close button automatically saves any changes to the script.
- (4) As this button already has a Goto instruction defined in the Properties window, this instruction must be removed otherwise your script will be ignored. In the Properties area click on the Goto popup menu and change it to none.
- (5) Try to run the project, if you made an error when typing in the script resulting in a statement which HyperNext cannot understand then the compiler will abort and open the script editor with the offending statement indicated and an error message displayed.
- (6) Assuming your project is running then just click on the lone button and it should Beep and cause the second card to load.

Congratulations you have just written your first dynamic script - it was only one extra line but still it changed cards and also gave you a basic understanding of creating and running a script.

Lesson 5 - Variables, Text, Messages and Speech

Now that you know the basics of writing a script you are ready to try out some more commands. In this lesson you will learn how to make your stack display a message box and then speak the message. Although you have not really covered fields yet, your stack will also use a field as a text source. Text is very important because it is the basis of nearly all data in HyperNext.

Theory - Text and HyperNext

In HyperNext all variables are stored as text whereas in most other languages there are several types of variable and each is stored using a specific format. Variables are just locations in memory which hold values such as whole numbers, decimal point numbers, strings of characters(text) and pointers to other variables. In HyperNext when declaring a variable you do not have to specify what type of data it will hold because everything is stored as text. In most other languages variables have to have their type of data declared and different types cannot easily be mixed.

To declare a variable called "xdata" in HyperNext you can use statements like

Local xdata

Global xdata

The first states that the variable xdata can only be accessed from within the procedure or handler in which it is declared whereas the second allows the xdata variable to be accessed from anywhere within the program.

When using xdata you can put just about anything into it.

Put 'hello' into xdata

Put 25 into xdata

Put 3.142 into xdata

Put field 1 into xdata

The last statement uses a field and shows that when users of your program put some data into a field it is very easy for the field's data to be placed into a variable. Also the variable itself doesn't care what kind of data the field contains.

In many ways fields and variables are very similar except that your program users can usually see the fields whereas the variables are hidden from them. Another major difference is that processing the data in a field is quite slow whereas processing the same data in a variable is much much faster.

Practice - Processing text from a field

Here we will create a new project comprising a card with two buttons and a field. One button displays the contents of the field using a message box and the other button speaks the field's text.

- (1) Create a new project and on the card add two buttons. Using the "Value" section of the Properties area label one button "Show" and the other button "Say".
- (2) Select the field and in the "Place" section of the Properties area set the field type to "Multi border scroll". By default fields are set to accept only one line of text.
- (3) In the script of the button labeled "Show" type in the following statements.

```
Local mess  
Put field 1 into mess  
Message mess
```

(4) In the script of the button label "Say" type in the following

```
Local mess  
Put field 1 into mess  
Say mess
```

(5) Instead of running the project we will first check the program is fully understood by HyperNext using the compiler via the keys CMD + "K". If the compiler finds an error it will open the script editor and highlight the errant line.

(6) Now run the project and in the field type some text.

(7) Press the "Show" button and a message box should appear with the field's text in it. Dismiss the message box by clicking the "OK" button.

(8) Press the "Say" button and your computer should speak the field's text. If speech has not been running since the computer started up then it will take a few seconds while your computer initializes the speech. If your computer has the speech disabled then you will hear and see nothing.

Lesson 6 - Fields and Text

In the previous lesson you learnt a bit about fields and used one as a text source. In this lesson you will see how to use fields residing on both cards and backs, and on cards which are out of view.

Theory - Fields, their identity and location

There can be many fields on a card and therefore HyperNext needs some way of distinguishing between them when a statement refers to them. The following two statements refer to two fields, having identities 1 and 2:

Put xdata into field 1

Put xdata into field 2

Most statement when referring to objects assume that the object is on the current card but sometimes it is important to access a field on cards which are not visible, for instance when processing a set of record where each record has its own card. When referring to a field located on another card the following commands for setting and fetching text are used

FieldCardSet(cardid,fieldid,value)

FieldCardFN(cardid,fieldid)

for example

FieldCardSet(21,3,'just some data')

Put FieldCardFN(21,3) into xdata

Practice - Working with fields

In this part we will create a new project and try the field command mentioned above.

- (1) Open the Creator and make a new project.
- (2) On the card create two buttons and using the Properties area give them the labels "Card" and "Hidden"
- (3) Place the following script in the button "Card"
Put field 1 into field 2
- (4) Create two fields on the card and with the Properties area give the first one the value "visible card"
- (5) Now run the project and press the button labeled "Card". It should copy the contents of field 1 into field 2.
- (6) End the Run and return to Design mode.
- (7) Now we will try accessing a field on a hidden card which we will now create.
- (8) On the new card place a field and using the Properties area give the field the value "hidden card".
- (9) Go back to the first card and in the button labeled "hidden" put the following script

Put FieldCardFN(2,1) into field 2

(11) Run the project and press the button labeled "hidden". This should copy the contents of card 2's field into field 2.

Lesson 7 - Building a Stack

This lesson assumes that you already have the Registered version of HyperNext Creator because only the Registered version can build stacks.

This will explain some more about stacks and also demonstrate how to build one from your project.

Theory - What are stacks?

A stack is a document that when loaded into HyperNext Player defines the structure of the running software. Just like its original project within the Creator, a stack has exactly the same card structure, layouts, controls, scripts, variables etc.

Stacks are intended to provide easy-to-use software and therefore when a stack quits it can automatically save its state including all variables, fields, canvas images etc. This means that for many stacks the user never needs to save them, it's done automatically.

Although the File menu has "Save", "Save As" menu items, they are intended for use by the stack author in case special settings, preferences or other data files must be saved.

Stacks are encrypted so that their structure, scripts, variables and data are protected from prying eyes. In principle a stack can be run on any computer platform which has a Neural Player, although at the moment the Player only runs on Macintosh OS X and OS 9 and Windows XP.

Practice - Building a stack

Building a stack is easy, just do the following.

- (1) Load your project into the Creator, just as you normally would when planning to edit or run it.
- (2) From the "Go" menu select "Build Stack". The Creator will then try to compile your project and if successful will proceed to build your stack.
- (3) The built stack will have the same name as your project but its ending will be ".stk". Note, the stack will be cross-platform although you must ensure that you also bundle its folder and data folder with it.
- (4) Test your stack by double clicking on its file icon, this will launch the Player causing it to load and run the stack.
- (5) Your stack will run almost exactly as its project did within the Creator except that the stack is more responsive and any AppleEvent scripts are now able to respond to an AppleEvent.
- (6) If you plan to distribute your stack then you must ensure that you distribute the folder which it shares with the project. This folder must also contain the "data" folder because the stack will look for it. Of course it does not require the project file. Also, the stack does not need the "Resources" folder as it was only required by the Creator when it built the stack.

Lesson 8 - Building a Standalone

This lesson assumes that you already have the Registered version of HyperNext because only the Registered version can build standalones.

This will explain what a standalone is, why you might need one, and then will demonstrate how to build one from your project.

Theory - What are standalones?

HyperNext can produce Macintosh and Windows applications, here called standalone programs because they do not need either the HyperNext Creator or Player to run. By changing some details such as the About Box, Splash Screen, Mac creator code etc, the identity of the software tool which created the standalone can be hidden from the final user.

Standalones are generally used by commercial ventures as they are perceived as being more professional than a stack that requires a HyperNext Player to run.

The disadvantage of standalones is that they are much larger than the equivalent stack. A stack may only be 50k but the equivalent standalone will be at least 5MB, although perhaps not a lot by today's standards.

Practice - Building a standalone

Building a standalone is easy, just do the following.

- (1) Load your project into the Creator, just as you normally would when planning to edit or run it.

- (2) From the "Go" menu select either "Build OS 9" , "Build OS X" or "Build Win XP" , depending upon your target platform. The Creator will then try to compile your project and if successful will proceed to build your standalone.
- (3) The built standalone will have two main files, the standalone application itself, and the standalone's definition file. The standalone has the same name as the project but with a "_C" suffix for Classic and a "_X" suffix for OS X. The definition file has the same name as your project but its ending will be ".app". Note, if you are building on Windows XP for a Macintosh standalone then the built standalone will be bundled inside a .bin file that must be decoded on the Macintosh - there are many freeeware utilities to do this and modern Macs using OS X can handle this seamlessly.
- (4) Assuming your target platform is the same as your build platform then the standalone can be tested by double clicking either on it or on its definition file, so launching the standalone.
- (5) Your standalone will run almost exactly as its original project did within the Creator except that the standalone will be more responsive and any AppleEvent scripts are now able to respond to an AppleEvent.
- (6) If you plan to distribute your standalone then you must ensure that you distribute the folder which it shares with the project. This folder must also contain the "data" folder because the standalone will look for it. Of course it does not require the project file. Also, the standalone does not need the "Resources" folder as it was only required by the Creator when it built the standalone.

4 Developer Lessons

The HyperNext Developer allows you to make Creator plugins, these are libraries or programs that add extra functionality to the HyperNext language. A plugin when dropped into the Creator plug-in folder immediately gives the Creator access to the functionality of that plug-in. The Creator is designed to operate simultaneously with many plugins and has a simple system for avoiding plugin name clashes.

The Developer's design environment is very similar to that of the Creator as it shares much of the same functionality. However, whereas the Creator can produce stacks and standalone applications, the Developer can only produce plugins for use by the Creator.

a. Functionality

At the present time the Developer cannot produce plugins having cards. The following aspects of the Creator are not accessible in the Developer:

- More than one card,
- Menus.
- Sound, image and movie libraries.
- Apple events

The Developer has access to most of the HyperNext language, including to AppleScripting and to RBScripting. These two scripting languages have different roles and greatly increase the capability of plugins. For instance, an AppleScripting plugin could allow stack developers to control another application such as AppleWorks, Eudora, Quark etc. In contrast, RBScripting is a very fast typed language and an RBScripting plugin might add some image

or text processing capabilities that most stack developers would find difficult to write. A good example of a complex plugin is the BP1 neural network plugin that comes with HyperNext Studio.

The following areas assume that you have already worked through the Creator lessons and are familiar with creating and editing scripts.

Lesson 1 - Your first plugin

In this first lesson you will see how easy it is to produce a simple plugin. This plugin will just take a parameter *n* and then Beep *n* times. The practical section will demonstrate how to write the plugin, build the card on which to test it, and then run the plugin within the Developer.

There is also a plugin demo called "Circles" on our web site and this demonstrates how to access canvases and use RBScripting.

Theory - About plug-ins?

In one sense writing plugins is easier than developing a whole stack because the plugin scripts are located in just one place, the MainCode section, and so it is easier to keep track of your code. The MainCode section can be accessed from the "Edit" menu in the Developer.

The MainCode script comprises both global procedures and global variables. As with any other script the MainCode script can have its own procedures but these procedures are currently all global. These procedures can be called from a stack that uses the plugin but the user needs to know their name and required parameters. Each procedure can of course have its own local variables and has access to all of the HyperNext language except for sending and receiving AppleEvents. Both AppleScripting and RBScripting are available to plugin authors so increasing the capability of plugins.

Plugins can access controls such as fields and canvases on any card although unless specifically referred to, any reference to a control will be directed to the control with that identity and location on the current card. For instance if the plugin issues the following statement

Put 'some data' into field 3

The effect will be to put the "some data" into field 3 of the card in focus. The plugin can still access fields on out of focus cards using the FieldCardSet and FieldCardFN keywords.

The plugin environment has been designed with security in mind so that plugins can only run when they are explicitly called from the stack in which they have been linked. If a plugin needs to be initialized before it can be used then the plugin author must specify this to their users who will then explicitly use the specified call in their stack.

Practice - Building a plugin

In this lesson you will create the plugin and then just test it by running it within the Developer.

- (1) Run the Developer and create a new project called "First".
- (2) On the card add two buttons and using the Properties area label the first one "Init" and the second one "Beep".
- (3) Begin writing the plugin by opening up the MainCode script from the "MainCode" item on the "Edit" menu.
- (4) Create a new handler called "Init" and in the handler just put the statement

```
Message 'Initialising Plugin'
```
- (5) Create another handler called "TestBeep" and give it one parameter

called "num".

(6) In the TestBeep handler add the following code

```
Local n
For n=1 to num
  Beep
EndFor
```

(7) Close the MainCode script editor.

(8) Select the "Init" button and edit its script. Remove the Beep command and replace it with the statement

```
Call Init
```

(9) Close the button's script editor.

(10) Select the "Beep" button and edit its script. Remove the Beep command and replace it with the statement

```
Call TestBeep(5)
```

(11) (9) Close the button's script editor.

(12) Test the plugin by running the project.

(13) Imagine that the plugin needs initializing before it can be run. Therefore, press the "Init" button and a message box containing "Initializing" should be displayed.

(14) Press the "Beep" button and five beeps should be sounded.

Congratulations, this is your first HyperNext plugin.

Lesson 2 - Testing a plugin using the Creator

This lesson assumes that you already have the Registered version of HyperNext because only the Registered version of Developer can build plugins.

You will need to have both the Developer and Creator open at the same time so that after the Developer has built the plugin the Creator can test it.

Theory - The Creator and Plugins?

The Creator only checks the Plugins folder when it has to compile, run or build a project. Any plugins it detects are processed and their handlers and global variables are incorporated into the executable unless disabled in the Preferences. This means that even if your projects do not access any plugins, those enabled plugins within the plugin folder will still make your final stack/program larger and use resources. The amount of resources used depends upon the plugin although if the plugin is not initialized from within your stack its required memory space will be minimal as uninitialized variables are empty.

Practice - Building a plugin

Here you will use the plugin project that you created in the previous lesson and during the lesson will make some changes to the plugin to see how the arrangement works.

(1) Open up both the Creator and the Developer. If you do not have sufficient free RAM to open both at once then just open up the Developer.

- (2) In the Developer load the plugin project "First" from the previous lesson.
- (3) Build the plugin by choosing the "Build Plugin" item from the "Go" menu. This will build the plugin entitled "First.xcm" and place it in the plugins folder.
- (4) Switch to the Creator. If the Creator is not already running because you are short of RAM, close the Developer and then run the Creator.
- (5) In the Creator make a New project and call it "TestPlugin".
- (6) On the card add two buttons and using the Properties window label the first one "Init" and the second one "Beep".
- (7) Select the "Init" button and edit its script. Remove the Beep command and replace it with the statement

Call First.Init
- (8) Close the button's script editor.
- (9) Select the "Beep" button and edit its script. Remove the Beep command and replace it with the statement

Call First.TestBeep(5)
- (10) Close the button's script editor.
- (11) Test the scripts by running the project.
- (12) Imagine that the plugin needs initializing before it can be run. Therefore,

press the "Init" button and a message box containing "Initializing" should be displayed.

(14) Press the "Beep" button and five beeps should be sounded.

Lesson 3 - Plugin global variables

The Developer uses the same HyperNext language as the Creator and so also supports global variables. Plugin global variables can be accessed from outside the plugin so allowing users to pass data to the plugin and also receive data from it. However, it is not possible for a plugin to access either the global variables or global procedures of the HyperNext stack/program in which it resides.

Theory - Plugin Globals?

Accessing a global variable in a plugin uses a similar syntax as accessing a plugin's global procedure in the previous lesson. Assuming the global variable is called "count" and it is in the plugin called "First", then it can be accessed from the stack with the following statement

```
Put First.count into mycount
```

If the plugin has very long name and your stack will access its global variables and handler a lot, then its probably a good idea to rename the plugin to something shorter.

Lesson 4 - Plugin clashes

If you use several plugins then perhaps one day someone might produce a plugin sharing the same name as another plugin that you are currently using or are intending to build, In such a case you need to prevent them clashing because after all it is not possible two have files sharing the same name within a folder.

Theory - How can plugins clash?

Plugins can only clash in the sense that they both share the same filename but in reality only one of them could be placed in the plugins folder. The solution is simple, just rename one of the plugins. You might do this anyhow as it can save you a lot of typing if a plugin has a long name.

Practice - Using two plugins.

To save us time writing another plugin we will just go to the Plugins folder and make a copy of the "First.xcm" plugin already there and then modify the earlier Creator project "TestPlugin".

- (1) Go to the Plugins folder and make a copy of the "First.xcm" plugin".
Rename the copy plugin to "Second.xcm".
- (2) Open up the Creator project "TestPlugin".
- (3) On the card add two more buttons and using the Properties window label the first one "Init2" and the second one "Beep2".
- (4) Select the "Init2" button and edit its script. Remove the Beep command

and replace it with the statement

```
Call Second.Init
```

(5) Close the button's script editor.

(6) Select the "Beep2" button and edit its script. Remove the Beep command and replace it with the statement

```
Call Second.TestBeep(3)
```

(7) Close the button's script editor.

(8) Test the scripts by running the project.

(9) Imagine that the plugin needs initializing before it can be run. Therefore, press the "Init" button and a message box containing "Initializing" should be displayed.

(10) Press the "Beep" button and five beeps should be sounded.

(11) Now we will test out the second plugin although clearly a better test would have been to write another plugin having different actions to those of the first plugin.

(12) Imagine that the plugin needs initializing before it can be run. Therefore, press the "Init2" button and a message box containing "Initializing" should be displayed.

(14) Press the "Beep" button and three beeps should be sounded.

5 Using AppleScript

AppleScript is a scripting language that is understood by all modern Macintosh computers and allows you to control the computer without using the keyboard or mouse.

Many applications also understand AppleScript and so it is possible to control these applications from your HyperNext stack or standalone.

Note, at the present time only one script can be active(compiled) at any one time. So if you have several AppleScripts then each must be compiled immediately before they can be run. For instance

```
AScriptCompile src1
```

```
AScriptRun
```

```
AScriptCompile src2
```

```
AScriptRun
```

There is a demo project and stack on our web site that shows to add an AppleScript source text, compile it and then examine the results. However, it only has one simple AppleScript and leaves it up to the user to type in their own AppleScript source code.

At the moment, if you need more information on HyperNext AppleScripting please look at the Creator/Developer built-in help.

6 Using REALbasic Script

HyperNext is written in REALbasic and therefore can give access to RBscript, a typed object orientated language similar to modern BASICs.

RBscript runs much faster than the native HyperNext language and so is very useful where speed is important. For most tasks HyperNext is more than fast enough but for graphics, image or heavy text processing RBscript can be tried.

On our website is an example of using RBscript, it is the Plug-in example called "Circles". The example shows how to enter RBscript source, compile and run it, and also how to use a canvas from RBscript.

At the moment, if you need more information on HyperNext RBscripting please look at the Creator/Developer built-in help.

The End