

Align



The alignment tool is used for aligning your objects to other objects. The reference object is always the first selected...

The buttons from top to bottom are:

Align to Top

Align to Bottom

Left Align

Right Align

Align Vertical (fixed at 8 pixels)

Align Horizontal (fixed at 8 pixels)

Same Width

Same Height

Same Size

Data Monitor



The Data Monitor is a simple but powerful traffic monitor that will help you diagnose and troubleshoot your projects. The first parameter is always the socket used for communication (separated by a TAB character), and the second parameter is the data. You can copy data from either field...a 'logger' and 'HEX' conversion setting will be included in an upcoming version.

The 'Outgoing' (TX) field is the top field.

The 'Incoming' (RX) is the bottom field.

The rightmost field shows all open sockets.

Editor Tools



The Editor Tools represent shortcuts to control Editor behavior:

Verbose

When this is enabled, all mouse activity, movement, resizing, and internal messages, will be displayed in the 'Messages' section of the 'Script Window'.

BG Edit

When a group is selected, the 'BG Edit' (i.e. Back Ground edit) button will be enabled, and you can enter a special mode for editing just that group. Pressing the button again will exit this mode. You may also press the 'Run' Mode' (Hand tool), or use 'CTRL-B' to enter/exit. If you press the hand tool, you will exit editing and clear all undo/redos.

Hide Palettes

A quick and convenient way to quickly hide/show all editor palettes.

Undo

It was not an easy task to provide undo/redo for a multiple window - multiple page environment (it is more complicated than you think!), so there is an 'Undo Meter' to provide visual feedback. The Undo Meter (next to the undo enable/disable button) is there to give you an indication of what is going on, and when. The bar is showing you when the RTE is finished taking a 'snapshot' of your state, so you should wait until the indicator shows 100% before you make any more changes. This typically happens so fast that you don't even see the bar move, but the more objects you have (and the more complicated the page is) you will notice that the indicator takes longer. If your page is so complicated that it is becoming cumbersome to work, you might want to consider breaking up the page into smaller components...ultimately, you may turn undo completely off.

Turning the undo/redo completely off may or may not be useful for you...it can be good for stopping at a point where you would like to return (disabling leaves your previous undos/redos intact), and turning it on again simply resumes from that point. However, you will no longer have any protection from deleting objects, or making dramatic changes that you make while the undo is disabled. It is on by default, and it resumes the 'On' state after a restart.

This feature was integrated to provide a way to turn undo off if you are working on a very slow computer, or an extremely large and complicated (i.e. intensive) project. Also, because this is a 'live' environment, you may find that the ability to totally turn off any demanding processes (yes, it is a demanding process!) may be useful.

If the indicator gets 'stuck' at any value other than 100%, it could mean that your undo state may not be stable...you should stop editing (choose the 'Hand' tool) or disable undo at this point to reset (and probably save a copy of your work). Here are some actions that will reset/clear all undo/redo points:

- Pressing the 'Hand' tool icon (i.e. exiting the edit mode)

- Going to another page
- Going to another window
- Loading a new project
- Creating a new project window
- Creating a new page

And a few more behind the scenes actions...

Image Selector



You can visually choose the ON/OFF and Fixed image for buttons (fixed images only for the background of a page) by double-clicking on any image in the Image Selector. You choose the type of image to assign by choosing in the 'Settings' menu (ON/OFF/ Fixed). The ON Image is the image that is seen when the user pushes the button, and the OFF Image is seen when released (or by default). The 'Fixed Image' will stretch to fit the button rectangle, and will override the OFF image.

The button will automatically become transparent when you click to give you a default ON/OFF effect without any further assignment. This will also show the ON image through the transparency, so you can attain some interesting effects. If you use a transparent image as a fixed (i.e. fixed to the size of the button - 'stretched') you can also see your OFF Image as well.

The 'Directory' button allows you to change the image directory that the Image Selector points to (this setting is saved in your preferences), and the images may be larger than the preview button. By selecting the 'Fit to Button' option, you can shrink or enlarge images so that you can see all of the data.

Images are stored in the project by default, but you can set the image to a file path using the 'File Path Only' setting in the 'Options' menu. The menu settings and options are transient (they are not remembered after you restart). You can right-click on any object to delete an image (this will also delete it from the project, but you can use 'Undo' to revert). There is no option to export images at this time, so maintain a backup of your images. This option will be in the next version for sure.

To assign a background image to a page, click anywhere on the page itself to make sure it is the current object, and then select 'Fixed Image' from the Image Selector, and double-click the image you want. The image will resize to fit the page. This works the same for buttons, but the button can easily be resized, and you can achieve some interesting effects with multiple assignments (ON/OFF/Fixed).

Note: There are alternate ways of providing hilighting by setting the 'HILITE_COLOR' property and 'HILITE_SHAPE' for an object. This will override any transparency hilighting for images, but can be useful for basic objects like rectangles, circles, and rounded rects. See the language section for more information.

Information

The RTE was written in Java, but has been precompiled and optimized using a special compiler so it will run independent of the Java version you have installed on your computer. In fact, the installation does not install any Java files at all, just the environment for the RTE. Your computer is safe from any alteration of your current Java configuration. This installer does modify your Windows registry to associate your 'rte' project files extension with the system so when you double click on a project, it will open the RTE3 (as well as your project). This also means you can use the project to start the RTE3.

Note: If you do not have a USB Master key, the RTE will run demo mode only. An Internet connection is required for this mode...

Tips for the RTE3

- The first time you start the RTE, it will take a while to load, but subsequent starts will be very fast.
- The password to get out of the full screen mode is 1234. If you have an RTE USB Master Key, the password will be different.
- Alt-clicking any object will bypass script, so you can select an object without triggering.
- Only some of the properties can be changed through the extended 'Property List', all other properties are changed using the 'Property Editor- The 'Build' number (in the about dialog) is the date of the build.
- Go through the 'Examples.rte' project, as well as the 'DVD Librarian' project for scripting examples.
- Save often.

RTE2 to RTE3 conversion

Please download the latest version of the RTE2 if you are converting projects to the new RTE3 format...there have been many changes. The conversion is mostly for objects (pages, buttons, images, etc.); IR and script conversion is rudimentary. The IR code from the 'Inspector' will be converted to a SEND.STRING command (which is what it was internally), and the driver information will be commented in that same script. This is a new environment with a totally new scripting engine, and it is not an XSYS Controller IDE...it's a network control IDE. When you create new IR objects in the RTE3, you can use the RTE2 to generate the strings for an XSYS Controller, and then paste these

strings into a SEND.STRING command. For other control systems, you will obviously be triggering internal code of some kind, so this is the normal network communication you would expect (i.e. using SEND.STRING for sending strings to a device).

All push commands become DO.PRESSED commands, but everything else is the same. The naming convention for PUSH has changed to be paired with RELEASE (as in PUSH/RELEASE) and the PRESSED action refers to the PUSH/RELEASE action (triggering a PUSH script followed by the RELEASE script). The parameters have changed order to be more in line with Industry convention where the socket is the first parameter, the string is the second, etc. For SEND.STRING commands in your scripts, this is already 'swapped' for you during the conversion.

The language is very different, but I feel that it is more professional, but there were some conventions carried over (SET, GET, DO, GO, etc.) so you should catch on quickly. For those of you that do real programming, the integration of a Java/JavaScript interpreter should be a great addition. I am sure that you will also appreciate the (rudimentary but useful) script help popup (at least for the custom commands). The Java/JavaScript language is very well documented already, so there is no online help at this time. The scripts can now be as big as you want, and the syntax highlighting is much improved. There is also a new script debugger (in addition to the script window) that allows you to step through code, view variables, etc.

The network communication seems to be improved, so I think you will like the overall feel of the new engine. There are a few Java like things that you will have to get used to for now (file dialogs for one), but I think this is a very small price to pay for what we get in return.

Unfortunately, this engine has not had the extensive field testing that the RTE2 did, but all of the experience that was gathered during that development of the RTE2 has been integrated into the RTE3. The test results show that the RTE3 is ready for 'prime time', but you may want to test it extensively on your end before integration. If you need help converting projects, just send them by email, and I will take a look.

Notes for RTE3 (things that need to be fixed or changed):

- Page and object FX are still slightly experimental, but useful.
- Background editing does not allow select-dragging at this time (you can shift-select to select multiple objects, or use ctrl-A to select all).
- Pages can get out of sync (the RTE thinks you are on a different page than you are) and you can re-sync things by pressing the left-most arrow on the Navigator...this will return you to page 1, and tell the RTE to reset it's logic the same.
- Menus are based on 'mouseup' actions, so 'sliding' your mouse may not work...click on the menus instead.

- Tool palette positions are not remembered (i.e. they are docked every time you startup the RTE).
- Java is a great language, but also has some 'special' features. The startup time still takes quite a while the first time. I am still optimizing the performance, but I have found it to be very acceptable, especially on dual-core systems.

Coming soon (maybe next time!):

- More levels of Undo/redo for the script editor.
- The 'real' Object Editor. The current editor is more like the Inspector (which is what the 'Property List' and current editor will be combined into).
- Editing from the Project View.
- Streaming audio and video.
- More RTE2 to RTE3 conversion features.

About the Undo

There is tremendous amount of 'juggling' required to provide undo/redo for a multiple window - multiple page environment (it is more complicated than you think!). The 'Undo Meter' next to the undo enable/disable button is there to give you an indication of what is going on, and when. The bar is showing you when the RTE is finished taking a 'snapshot' of your state, so you should wait until the indicator shows 100% before you make any more changes. This typically happens so fast that you don't even see the bar move, but the more objects you have (and the more complicated the page is) you will notice that the indicator takes longer. If your page is so complicated that it is becoming cumbersome to work, you might want to consider breaking up the page into smaller components...ultimately, you may turn undo completely off. If you turn it off, you should save your projects often (this is recommended anyway).

Turning the undo/redo completely off can be both good and bad...it can be good for stopping at a point where you would like to return (disabling leaves your previous undos/redos intact), and turning it on again simply resumes from that point. However, you will no longer have any protection from deleting objects, or making dramatic changes that you didn't want. It is on by default, and it resumes the 'On' state after a restart.

If the indicator gets 'stuck' at any value other than 100%, it could mean that your undo state may not be stable...you should stop editing (choose the 'Hand' tool) or disable undo at this point to reset (and

probably save a copy of your work). Here are some actions that will reset/clear all undo/redo points:

- Pressing the 'Hand' tool icon (i.e. exiting the edit mode)
- Going to another page
- Going to another window
- Loading a new project
- Creating a new project window
- Creating a new page

And a few more behind the scenes actions...

The Background Edit mode is rather unique, and is a carry over from the RTE2. This is basically for editing groups (so you don't have to un-group them just to edit). While you are in this mode, the Undo meter indicates yellow for 'caution'...your undo stops at this point, so your undo/redo point is before and after your BG edit session. You enter/exit the BG Edit mode by pressing the 'BG Edit' button on the tool palette (on the 'North' palette). If you press the Hand tool to exit, you will reset the undo.

Backup

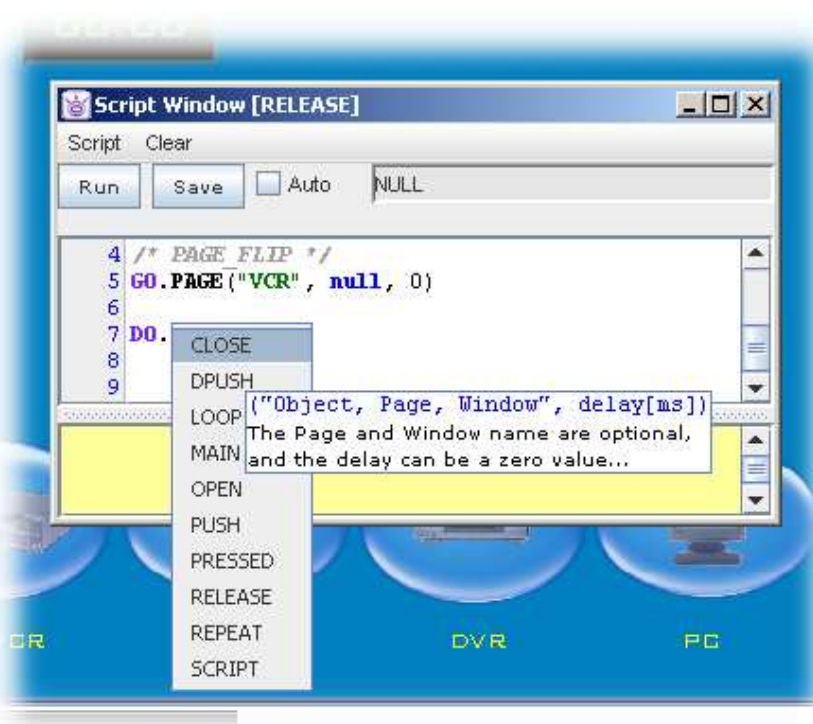
The RTE does not have a formal option for choosing backup at this time, but a copy of your project is saved when you 'Save' over your current project (either from the menu or by ctrl-s). The name of the file is '~temp.bkp', and it represents a copy of your project file before it is deleted (i.e. written over). This will be the same project you loaded before any change, so it may represent several minutes/hours ago, but it is some protection for your work. You can either load this file directly from the 'Open' file dialog (select 'All Files' option for viewing) and work with it normally (rename, edit, save, etc.). This will eventually become part of the 'Revert' option that was not ready for this release.

Disclaimer:

THIS SOFTWARE IS PROVIDED GRATUITOUSLY AND, ACCORDINGLY, WE MAKE NO WARRANTIES, EXPRESS OR IMPLIED, REGARDING THIS PRODUCT OR ANY SERVICE OR ITEM OFFERED, SOLD OR OTHERWISE PROVIDED TO YOU AS A RESULT OF YOUR USE OF THIS PRODUCT. WE SPECIFICALLY DISCLAIM ANY WARRANTIES, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE REGARDING THIS PRODUCT OR ANY SUCH SERVICE OR ITEM. WE MAKE NO WARRANTY THAT THE

INFORMATION CONTAINED IN THIS PRODUCT IS COMPLETE OR ACCURATE. WE DO NOT ASSUME AND HEREBY DISCLAIM ANY LIABILITY TO ANY PERSON OR ENTITY FOR ANY LOSS OR DAMAGES (INCLUDING, WITHOUT LIMITATION, SPECIAL, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES) CAUSED BY ERRORS OR OMISSIONS IN THE INFORMATION CONTAINED IN THE PRODUCT, YOUR INSTALLATION AND/OR USE OF THE PRODUCT AND/OR ANY SERVICE OR ITEM PROVIDED TO YOU AS A RESULT OF YOUR USE OF THIS PRODUCT, REGARDLESS OF WHETHER RESULTING FROM NEGLIGENCE, ACCIDENT OR ANY OTHER CAUSE WHATSOEVER.

xScript Language



You have online help for the xScript language. Typing the correct command will result in the appropriate highlighting for that word (black-bold for keywords and purple for commands).

When you type the period character after a keyword, after a few seconds a menu will popup with the commands available for that keyword. You can now move the mouse over each term to get more help (a brief description and syntax) and you can click or press enter to paste example text into your script.

As the title indicates, this is a language, and is actually a combination of three:

Java

Javascript

xScript

The Java and JavaScript part of the scripting are thanks to the Rhino scripting engine technology from the Mozilla organization, and xScript is a custom language developed by RCS Programming. If you are familiar with Java and JavaScript, you should be able to just dive right in, if you never worked with the RTE before, don't worry, it's simple! Almost everything is based on an event trigger (Push, Release, Repeat, etc.) and you can trigger any objects script at any time. This allows you to perform 'real' object oriented programming (ha ha) by managing triggers on pieces of stored code. The hardest part of this concept is that there is no Main Loop...every object is a totally independent object that has its own scripting.

It will become more obvious to you as you go through the examples how powerful and easy this is. The added bonus is that the scripting is threaded by default, so you do not have to worry about anything other than creating! In fact, it can almost be fun!

Notes:

Some of the conventions used for describing the 'types' of parameters are ^ (the pointer symbol) for a return of some kind, and double quotation marks for regular strings. Everything else is assumed to be an implicit variable (integer, data, chars, etc.). Very often the first parameter is "Object, Page, Window" which represents a single string (variable length) with three parameters. The "Object" is required, but the "Page" and "Window" can be omitted, and you do not have to include commas if you only pass the "Object":

examples:

```
/* Only the object name is passed */  
  
/* Use this method for objects on the same page */  
DO.RELEASE("My Button", 0);  
  
/* The object name, and page name are passed */  
/* Use this method for objects in the same window */  
DO.PRESSED("My Button, My Page", 500);  
  
/* All three references are passed */  
/* Use this method for very accurate referencing */
```

```
DO.LOOP("My Button, My Page, My Window", 250);
```

```
/* For triggering a page script */
```

```
DO.OPEN("", My Page, "", 25);
```

```
/* For triggering a window script */
```

```
DO.MAIN("", My Window", 0);
```

(For more script examples, see the 'Language - Examples' section)

The **DO.RELEASE** command will trigger the **RELEASE** script for the button "My Button" in 0 milliseconds (i.e. immediately). This syntax is for 'triggering' a release script in another object. When you are defining the script within the release script of the object itself, you do not have to use this syntax...the script is triggered just the way it is. So, if you have a **SEND.STRING** command in the **RELEASE** script of "My Button", it will send the string as if you pressed the button. You might think of each action script as being a method or function, but you only need the definition part. And, rather than passing parameters, you use global variables.

You store values in variables that do not require declaration (like `int i`, or `string myString`) because all variables are cast to the 'implied' type in the interpreter. This might seem strange at first, but you will quickly get used to how intuitive it is. You use the 'var' keyword to identify 'global' variables, and no keyword for local variables. Local variables will automatically display their contents in the messages window of the 'Script Window'.

The large black/bold keywords are the classes, and the purple/bold words are the commands (methods). If a black/bold keyword does not have a period after it, and only has parentheses, it does not have any further methods, and you use the parameters on that keyword (the 'root' class). Please take a look at the 'Examples' project (available from the 'Help' menu, and the 'DVD Librarian' project for almost all of the examples you may ever need!).

Most commands have been tested (all of the important ones anyway!) but if you find something that doesn't work the way you expected (or the way you would like) please let [RCS Programming](#) know!

The Language

DATABASE.

CONNECT("dbType", "dbName", "dbUser", "dbPassword")

Returns an iterator...see the 'Examples' project

DISCONNECT(**Connection** con)

Disconnects from and closes the database referred to in the connection object

^ResultSet **QUERY**(**Connection** con, **String** sql)

Returns a ResultSet object based on the SQL query string

DO.

Note: For all pushes, the 'Page' and 'Window' name are optional, and the delay can be a zero value. As long as the object is on the same page, you can do a push without conflict, otherwise you need to address the object directly (Page and Window) to make sure you are triggering that object.

CLOSE ("Object", "Page", "Window", + delay[ms])

This is triggered when you move from one page to another.

DPUSH ("Object", "Page", "Window", + delay[ms])

This double-push script is triggered when a push is determined to happen within the System double click timing. You need to handle any other scripts you may have in this object so they don't inadvertently get triggered. For example, you would set a variable as false until the DPUSH is triggered, and only then will the REPEAT trigger.

LOOP ("Object", "Page", "Window", + delay[ms])

This is a convenience placeholder for any script, but is generally useful for looping script (i.e. scripts that need to run continuously). Take a look at the loop script in the 'XSYS Controller' button on the 'Utility' page of the DVD Librarian, or the Sleep timer logic (in the big Logo button on the main page of that same project).

When you start a loop, you are actually starting a thread. This thread reference is returned to you so that you can control your logic, and so you can 'cancel' the thread if you need to. Generally, you control any loop with a simple boolean variable, and your loop checks that every time through. The timer reference is the actual system reference, and you need to use the Java syntax to cancel:

example:

```
var myTimer

myTimer = DO.LOOP("My Loop Btn", 500) // sends a loop trigger to My Loop
Btn every 500 milliseconds

/* anywhere in the project */

myTimer.cancel();
```

MAIN ("Object, Page, Window", + delay[ms])

Another 'holder' for any type of script that can be triggered, or looped.

OPEN ("Object, Page", Window", + delay[ms])

This script is triggered every time you go to a page.

PUSH ("Object, Page, Window", + delay[ms])

This script is triggered on the Push of an object.

PRESSED ("Object, Page, Window", + delay[ms])

The action emulates an object 'push/release' combination automatically.

RELEASE ("Object, Page, Window", + delay[ms])

Triggered when an object is released.

REPEAT ("Object, Page, Window", + delay[ms])

Triggered after "INIT_REPEAT" milliseconds. By default, the value is 250, and is settable through a script using SET.PROPERTY("INIT_REPEAT"). The delay for the actual repeating is default 25 milliseconds, and is settable through a script.

SCRIPT("Script")

The "Script" is a string for evaluation.

GET.

^String **BUFFER**("socket")

example socket: "10.0.1.129:128"

* passing null (rather than a "socket") returns the main RX buffer

^color **COLOR**(r, g, b)

Returns a java color object.

^chars[] **CHARS**("fileName", count)

Returns an array of 'count' chars from fileName.

^byte[] **DATA**("fileName", count)

Returns an array of 'count' bytes from fileName.

^int **CHARCOUNT**("fileName")

Simply returns the number of chars in fileName.

^String **DIRECTORY**()

Opens a JDialog for selecting a directory (returns 'dirPath').

^String **DIRECTORIES**("filePath")

Returns all of the directories from a given filePath.

^String **FIELD**("Object, Page, Window")

Returns the contents of a text field.

^String **FILE**("fileName", count)

Reads 'count' number of lines from fileName.

^String **FILES**("path")

Returns a list of files in the given path.

^String **FILENAME**("filter")

Opens a JDialog for getting a files name; use the "filter" to filter out files by extension type (ex. ".txt").

^int **HEIGHT**("Object, Page, Window")

Returns the height of an object.

^boolean **HILITE**("Object, Page, Window")

Returns the hilite state of an object.

^String **ITEM**("data" , "delimiter", index)

index of 0 will return the last item...the "delimiter" is any char.

^String **LIST**("Object, Page, Window")

Returns the contents of a list field.

^point **LOC**("Object, Page, Window")

Returns the location (point) of an object.

^object **OBJECT**("Object, Page, Window")

Returns a java object reference for an RTE object.

PROJECT("fileName")

Opens the given project "fileName".

^String **PROPERTY**("Object, Page, Window", Property)

Returns the stored property for the given object.

^String **SOCKETS**()

Returns a (CR delimited) list of all open sockets.

^String **SPLIT**("fileName", "delimiter", count)

Returns a CR (carriage return) delimited string based on the "delimiter" and 'count' values.

^String **TEXT**("Object, Page, Window")

Returns the text from the given field.

^String **URL**("url")

Returns (as text) the given URL.

^boolean **VISIBLE**("Object, Page, Window")

Returns the visible state of the given object.

^int **WIDTH**("Object, Page, Window")

Returns the width of an object.

GO.

PAGE("name", "effect", delay[ms])

*The 'effect' is optional (use 'null' for optional parameters).

*The first parameter can be a name or page number.

URL("website")

URL("http://www.mywebsite.com")

Automatically opens the default browser...

MIDI.

INIT(String "Device Name")

Initializes the Midi I/O buffers for the given device (if available).

CLOSE()

Disposes of all Midi connections and structures.

TX(String "Device Name", int data1, int data2, int channel, int type)

Sends a Midi control message (please see included project "Midi_Control.rte" and the Midi Control Specification for more information).

Note: Use **GET.BUFFER**(null) to get any incoming data.

PLAYER.

DISPOSE()

Disposes (unloads) the current file object.

^int LENGTH()

Returns the 'loaded' files length (in seconds)

LOAD()

Prepares (loads) the file for control.

^int POSITION()

Returns the 'loaded' files position (in seconds)

SKIP(int position)

'Skips' to the files position (in seconds <int>)

START()

Starts playing the file at the current position.

STOP()

Stops playback.

SEND.

DATAGRAM("socket", "data", delay[ms])

Send any type of data to a UDP (Universal Datagram) socket. This is a connectionless protocol:

```
SEND.DATAGRAM("192.168.0.1:3210", "Hello from UDP!", 0);
```

param 1: UDP socket

param 2: Data

param 3: delay

OSC("socket," data, delay[ms])

Send an "Open Sound Control" (OSC) encoded message to an OSC server:

*/*This command will open a bidule project file. */*

```
SEND.OSC("192.168.0.1:3210", "/open", "s", "C:/bidule_projects/test2.bidule", 0);
```

param 1: OSC server socket

param 2: Command

param 3: Message Type

param 4: data

param 5: delay

STRING("socket", "data", delay[ms])

Send any type of data to a socket. The polling is controlled by the PING_STRING.

SET.

BUFFER("data")

Sets the main RX Buffer to data.

DATA("fileName", data)

Sets (deletes/creates) fileName contents to data.

*Use this for binary data.

FIELD("Object, Page, Window", "string")

Sets the given text field to string.

FILE("fileName", "string")

Sets (deletes/creates) fileName to string.

HEIGHT("Object, Page, Window", integer)

Sets the height of the given object to 'integer'.

HILITE("Object, Page, Window", boolean)

Sets the hilite state of the given object to 'boolean'.

HILITE_COLOR("Object, Page, Window", "r, g, b")

Sets the (optional) hilite color property.

*Uses the 'SHAPE' property.

HILITED_LINE("Object, Page, Window", integer)

Sets the hilited line of a text field to 'integer'

IMAGE("Object, Page, Window", "fileName")

Sets the objects 'IMAGE' property to the fileName.

ITEM("Object, Page, Window", "string", "delimiter", integer)

Sets item 'integer' (based on the delimiter) to string.

*Setting 'integer' to 0 will set the last ITEM.

LIST("Object, Page, Window", "string")

Sets the given text list to the given string.

*Lines are determined by carriage return.

LIST_COLOR("Object, Page, Window", "r, g, b", "r, g, b")

Sets the 'alternating' color lines (odd, even) of the given list field.

LOC("Object, Page, Window", int, int)

Sets the object location to the given coordinates.

PING("string")

Sets the main PING string (for all sockets) to the given "string".

PROPERTY("Object, Page, Window", "Property", value)

Sets the objects Property to the value.

*Runtime only (i.e. changes are not saved)

*Value' can be string or int

ROTATION("Object, Page, Window", int)

Sets the object rotation to the given degrees (0-360).

TEXT("Object Page, Window," "string")

Sets the text field to the given string

VISIBLE("Object, Page, Window", boolean)

Sets the visible of the object to 'boolean'.

WIDTH("Object, Page, Window", integer)

Sets the width of the object.

SHAPE("Object, Page, Window", "OVAL")

Sets the shape property of the object.

*Used in conjunction with 'HILITE_COLOR'.

FX.

SHOW("Object, Page, Window", "direction", speed, granularity)

Hides the object, and then shows it according to the "direction", speed, and granularity. The speed may vary according to the object size and the granularity setting, and should be considered a 'rate' control rather than literal milliseconds. The granularity controls the 'smoothness' of the transition (i.e. larger numbers are faster, but more 'choppy').

The directions can be "left", "right", "up", and "down".

example: `FX.SHOW("FX_Button", "up", 3, 4);`

HIDE("Object, Page, Window", "direction", speed, granularity)

Hides the object (shows it if hidden) according to the "direction", speed, and granularity. The speed may vary according to the object size and the granularity setting, and should be considered a 'rate' control rather than literal milliseconds. The granularity controls the 'smoothness' of the transition (i.e. larger numbers are faster, but more 'choppy').

The directions can be "left", "right", "up", and "down".

example: `FX.HIDE("FX_Button", "down", 3, 4);`

GROW("Object, Page, Window", "type", amount, speed, granularity)

Grows the object from the current size according to the "type", amount, speed, and granularity. The speed may vary according to the object size and the granularity setting, and should be considered a 'rate' control rather than literal milliseconds. The granularity controls the 'smoothness' of the transition (i.e. larger numbers are faster, but more 'choppy').

The type can be "center" and "TL" (for Top Left). The amount is in pixels.

example: `FX.GROW("Button_1", "center", 100, 10, 4);`

SHRINK("Object, Page, Window", "type", amount, speed, granularity)

Shrinks the object from the current size according to the "type", amount, speed, and granularity. The

speed may vary according to the object size and the granularity setting, and should be considered a 'rate' control rather than literal milliseconds. The granularity controls the 'smoothness' of the transition (i.e. larger numbers are faster, but more 'choppy').

The type can be "center" and "TL" (for Top Left). The amount is in pixels.

example: `FX.SHRINK("Button_1","center",100, 10, 4);`

NET.

UDPSERVER(int port)

Sets the UDP server to listen on the given port. Passing an argument of "0" will effectively disable the UDP server.

HTTPSERVER(int port)

Sets the HTTP server to listen on the given port. This will also reset the server if it was already initialized, and if "0" is passed as the port, the server will be reset and disabled.

DLL()

`^String DLL("fileName", "string")`

Sends the string to the dll, and returns the result.

HIDEMODE(bool auto)

This single command will put your project into a 'hide mode': all of the elements of the operating system will be hidden, and the background will become the color of the current page. You can double-click on any area to popup the password to return to the regular mode (SHOWMODE). If the boolean parameter is 'true', your project will be centered on the screen, and the OS will be hidden. If set to 'false', your window (and the 'Main' window) will become fixed (borderless) where it is, and the OS will not be hidden.

SHELL()

`^String SHELL("ipconfig /all")`

Returns the result from the Shell command.

SHOWMODE(int password)

This single command will get you out of the hidemode. You must pass the current password as the parameter for this to work.

SYSTEM.

^String **OS()**

Returns the current operating system (ex. Windows XP).

LAUNCH("fileName")

Attempts to launch the given fileName.

```
addScriptPopupMenu(new JMenuItem(LAUNCH), LAUNCH, HTMLTip(text, notes))
```

WAIT()

WAIT(milliseconds)

Attempts to 'pause' the current script for 'milliseconds'.

Language - Examples

& more

Tips and Tricks:

You can quickly get the value of a variable by setting the variable in a script without the 'var' identifier before it. This creates a 'local' variable, and it also tells the parser to return the value to the Messages field (that is where you will see the result). As you use projects like the 'Examples' project, and the 'DVD Librarian' project, you will notice values showing up in the messages field occasionally. This was done purposely to view the contents of variables during development, and you can do the same thing.

To step through a script, and view a list of variables while you are executing, you need to use the 'heavy-weight' debugger (the 'Script Window' is a light-weight debugger), and you access this by using the 'Windows/Script Debugger' menu. This will open the main script debugger, and you can get more information on the 'Script Debugger' page.

The combination of the 'Script Field' and the 'Messages' window is very powerful once you get the hang of it. You can type in a script at any time and press the 'Run' button to execute without actually assigning the script to any object. If you have the 'auto' button selected, however, the script will get saved to the current object.

How to:

Create a loop

Using any objects 'Loop' script (from the 'Script Window' menu select 'Script' and then 'Loop Script') enter your code:

(for this example, we created a button called "myLooper")

```
1  var loopContinue;
2  var myData = null;
3  var myData = GET.BUFFER(null);
4
5  if(myData != null) {
6  // do something with the data
7  SET.BUFFER("");
8  } else {
9  if (loopContinue != null) {
9  loopContinue.cancel(); // cancel the last thread
11 loopContinue = DO.LOOP("myLooper", 2000);
12 } // end if loopContinue
13 } // end if myData
14
```

As you can see, we start with two variables: loopContinue and myData .

The 'loopContinue' var we do not initialize (not here anyway) and the myData var we initialize with 'null'. On line 3, we use the command GET.BUFFER to get all of the buffers contents (from socket communications) so that we can do something useful on line 6. After we use our data, we should set the BUFFER to 'null' or EMPTY so that we don't trigger our code again (you don't have to do it this way of course). The 'LoopContinue' variable is just a 'flag' that controls whether we call our LOOP script again (line 9). This way we can control the start/stop of our loop, or just let it run indefinitely.

On line 10, we actually call the object itself (called 'recursion'), and we receive a thread object reference from Java that we store in the

'loopContinue' variable. This value allows us to cancel the last call to our loop (in case we are finished, or any other reason). Now, how do we start this loop? Easy! You can use the 'DO.LOOP("myLooper", 0)' call from any object, or even from the PUSH or RELEASE of the object itself. This also allows you to set the variables that it may use beforehand (like loopContinue) with any special values.

Note:

You need to include the page parameter, and possibly the window parameter if you plan on having this loop run in the background (or indefinitely) while you switch pages/windows (i.e. the "myLooper" reference will change when you leave the current page).

More examples:

From the 'Examples.rte' project

The following script is from the 'RELEASE' script of the 'Animate' button:

```
/*
The commands for repeat loops are the same common language that you are accustmed to...

You are not required to declare variables before using them. For example, notice the 'i' variable
was not declared or initialized until the first time it was used. Initialization is done for you,
and you can use variables as strings, integers, floats, etc.

The following 'for' loops simply set the x and y coordinates of the object to make it move...
*/

for (i=8; i<300; i=i+8){ SET.LOC("Button_1", i, i); WAIT(10); }
for (i=300; i>4; i=i-8){ SET.LOC("Button_1", i, i); WAIT(10); }
for (i=8; i<300; i=i+8){ SET.LOC("Button_1", i, 8); WAIT(10); }
for (i=300; i>4; i=i-8){ SET.LOC("Button_1", i, 8); WAIT(10); }
for (i=8; i<300; i=i+8){ SET.LOC("Button_1", 8, i); WAIT(10); }
for (i=300; i>8; i=i-8){ SET.LOC("Button_1", 8, i); WAIT(10); }

SET.LOC("Button_1", 8, 8);
```

Explanation: What this script is doing is starting a 'for' loop for each of the movements (left, right, up, down, etc.), and then finally setting the object to it's starting position. The 'WAIT' commands control the speed by adding a delay in script evaluation.

The following script is from the Slider2 'PUSH' script that controls the animated object:

vertically

```
/* PUSH_SCRIPT */

var slider2 = me.getValue();
var lastValue = slider2;
me.setMaximum(296);
me.setMinimum(8);
```

Explanation: This script sets initial variable values for the slider when the mouse is pushed so that the 'REPEAT' script (below) will have the correct numbers to work with:

```

/* REPEAT_SCRIPT */

var num1 = me.getMaximum()+8;
var num2 = me.getValue();
var num3 = num1 - num2;

if(me.getValue() != lastValue){
SET.TEXT("value_2", num3);
lastValue = me.getValue();
SET.LOC("Button_1",slider1, num3);
}

```

The delay value for the 'REPEAT' script can be set using the following command:

```
SET.PROPERTY("Slider_2", "RDELAY", "5");
```

The delay value is a custom property rather than a specific property value. This allows the value to stay with the object (yes, all scripts and settings follow an object as you cut and paste them!).

From the 'Examples.rte' Text' page '(Set Text' RELEASE script):

```

var myText = GET.FIELD("Field_0")

/* Let's get the current milliseconds so we can time the script execution speed */
var date1 = new Date();
var start = date1.getTime();

/* Now we set each field (named 'Field_1' to 'Field_48') by using the loop variable 'i' as the
last part of the name, so we don't have to type 48 lines! */

for(i=1;i<49;i++){
SET.FIELD("Field_"+i,myText)
}

/* Lets do the 'start' milliseconds minus 'end' milliseconds
calculation to figure out how long it took to do this */

var date2 = new Date();
var end = date2.getTime();

x = end - start;

```

```
/* You can set the text of the 'Messages' field in the Script Window! */
```

```
fldMessages.setText(x);
```

From the 'MP3 Player' page of the 'Examples.rte' project:

(The 'Load' button 'PUSH' script)

```
/* You can get the OS type to do any special OS specific work */
```

```
var myOS = SYSTEM.OS();
```

```
if(myOS.equals("Linux")){
```

```
var mySong = GET.FILENAME(".wav")
```

```
if(mySong != null){
```

```
var name = mySong.split("/");
```

```
}
```

```
} else {
```

```
var mySong = GET.FILENAME(".mp3");
```

```
if(mySong != null){
```

```
var name = mySong.split("\\\\");
```

```
}
```

```
}
```

```
if(mySong != null){
```

```
var isPlaying = false;
```

```
PLAYER.STOP();
```

```
PLAYER.DISPOSE();
```

```
var title = name[name.length-1];
```

```
SET.TEXT("Song", title);
```

```
PLAYER.LOAD(mySong);
```

```
SET.TEXT("Name", PLAYER.FILE());
```

```
SET.TEXT("Position", PLAYER.POSITION());
```

```
var lastThread = DO.RELEASE("Get Length", 1900);
```

```
}
```

Explanation: This script will load any .mp3 file, and 'prepare' it for playback. You must load a song first before any of the Player commands will work. The very last line has a delay of almost 2 seconds to allow the song to load, and then the song length is calculated by the 'Get

Length' button (called with 'DO.RELEASE').

(From the 'Position' button 'RELEASE' script)

```
var isPlaying;

if(isPlaying){
if(PLAYER.POSITION() == PLAYER.LENGTH()){
isPlaying = false;
PLAYER.STOP();
PLAYER.SKIP(0);
SET.TEXT("Position", 0);
} else {
SET.TEXT("Position", PLAYER.POSITION());
var myTimer = DO.RELEASE("Get Position", 1000);
}
// update the slider settings
var obj = GET.OBJECT("Slider_3");
obj.setValue(PLAYER.POSITION());
}
```

Explanation: This is the script that will update the song position field, and then loop (call itself) every second while the song is playing. The 'var myTimer' variable is keeping track of the timer reference for each call so you can .cancel() if you need to. This script also controls the slider while the song is playing with the last line 'obj.setValue(PLAYER.POSITION());'. The 'obj' reference was acquired by using the 'GET.OBJECT()' command that will give you a native reference to almost any object so that you can use direct java and javascript language commands (like 'setValue()') rather than using an xScript command.

It is the combination of native language commands and the RCS xScript custom commands that provide this flexible and powerful environment for you to use.

From the 'Database' page of the 'Examples.rte' project:

(The 'Open' button 'RELEASE' script)

```
var con;

con =
DATABASE.CONNECT(GET.TEXT("dbType"), GET.TEXT("dbName"), GET.TEXT("dbUser"), GET.TEXT("dbPassword"));

if(con != null){
obj = GET.OBJECT("disconnected");
obj.setBackground(java.awt.Color.GRAY);
obj = GET.OBJECT("connected");
}
```

```
obj.setBackground(java.awt.Color.GREEN);
}
```

Explanation: The 'DATABASE.CONNECT' command will return a connection object that you can use to do your datanase access. The object is a native object that you use normal iterative language on (see the 'Next Record' button 'RELEASE' script), and this example tries to make this as simple as possible. The 'obj' commands are simply controlling the hiliting of the buttons used for providing feedback to the user. The direct 'setBackground' method is one method of providing a hilite condition for an object, or you can use the 'HILITE' method for more advanced control.

(The 'Get Record' button 'RELEASE' script)

```
var con;

var rs = DATABASE.QUERY(con, "SELECT * FROM " + GET.TEXT("dbRecord"));

if(rs != null){
var rsmd = rs.getMetaData();
for (i=1; i <= rsmd.getColumnCount(); i++) {
MESSAGES.append("\n" + rsmd.getColumnLabel(i));
}
obj = GET.OBJECT("isData");
obj.setBackground(java.awt.Color.CYAN);
}

DO.PRESSED("dbNext",250);
```

Explanation: This is the button that takes the connection reference that the 'Open' button (the 'con' variable) , and retrieve the record information that we are interested in. Once we have this record 'pointer', we can access all of the rows and columns normally.

(From the 'Next Record' button)

```
var rs;

SET.HILITE(me.getName(), false);

if (rs != null){
if (rs.next()) {
for (i = 1; i <= rsmd.getColumnCount(); i++) {
switch (i){
case 1:
SET.TEXT("ISDNN", rs.getString(i));
break;
case 2:
```

```
SET.TEXT("Title", rs.getString(i));  
  
break;  
  
case 3:  
SET.TEXT("Author", rs.getString(i));  
  
break;  
  
case 4:  
SET.TEXT("Publisher", rs.getString(i));  
  
break;  
  
}  
  
} // end for  
  
} // if next  
  
} // if null
```

For an advanced example of file access, parsing strings, and object manipulation, see the 'DVD Librarian' project. The two buttons you will be interested in are the 'Update' button that is part of the 'Titles' group (on every movie page), and the 'Select' button that is part of the 'DVD Select' group (also on every movie page). The 'RELEASE' script of both buttons also has comments/notes.

For socket communications, you can take a look at the 'XSYS Controller' button on the 'Utility' page of that same project. The 'RELEASE', 'PUSH', and 'LOOP' script of that button form the polling required to get the main socket buffer to parse data.

Have fun!

RCS

Menu Bar



Opens a project file.



Creates a new project window.



Opens the Image Selector.



Opens the Property Editor.



Opens the Preferences window.



Opens the Network window.



Opens the Script Window. All system messages are also shown in the Messages field that is part of this window. Enabling the 'Verbose' mode will increase the number of messages in the Messages field, and provide more 'verbose' activity feedback.



Opens the Data Monitor.



Opens the Rhino Script Debugger.



Locks the Editor. The first project opened will become the main project as the menu bar is hidden, and the background fills the screen to provide the 'Full Screen User Mode'. The Editor is now only accessible by double-clicking on the screen to show the security keypad, and entering the correct password (1234 for the demo).

Navigator



The Navigator is a simple navigation tool for moving between pages in your project, and the up/down arrows are used for moving between windows. The Project View can also be used as a navigator, but this may be more convenient:

Left Arrow Down - goes to first page of the current window. This also resets much of the editing logic, and generally gets everything 'back in sync'. If things seems 'off', you might want to press this.

Left Arrow - goes to the previous page.

Right Arrow - goes to the next page.

Right Arrow Down - goes to the last page.

Up Arrow - goes to the next open window.

Down Arrow - goes to the previous window.

Special Features:

Pressing the left-most button (the 'Home' button) will reset the page logic for the window.

Network Monitor



The Network Monitor is a simple monitoring tool for any open sockets into and out of the RTE. As soon as a connection is made, the Disconnected button will turn to grey, and the Connected button will start the 'heartbeat' to indicate that there is a socket connection that is open. The 'Connected To' field will also show the last socket that had communication:

Buttons (from left to right)

Connected - The 'green' heartbeat will indicate a socket connection.

Disconnected - The 'red' disconnected button will show when all sockets are closed. Double-clicking this button will close the currently connected socket (ALT/Double-click will close all sockets).

TX - This 'light blue' button flashes whenever a valid string is being sent to a socket (connected or not).

RX - This 'amber' button flashes whenever data is received, and will only flash if a socket is open and communicating data.

Once a connection is made, the RTE will start pinging with a 'Ping String'. There are two reasons for this:

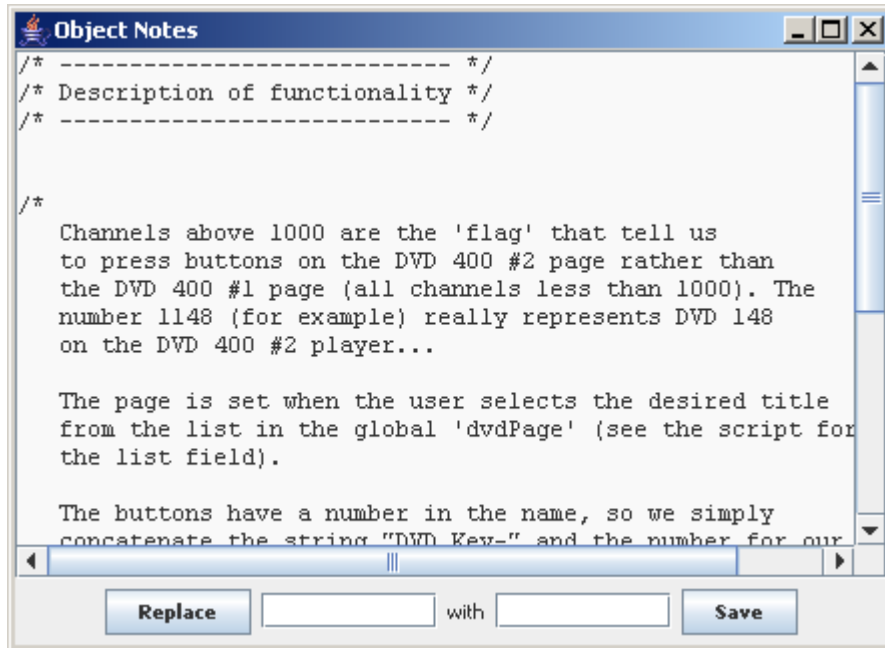
- 1) To keep the socket alive.
- 2) To keep the device we are connecte to alive.

You can change the ping string in the Preferences window (Network section) by typing in a text string, or using the standard notation for hex with a backslash ("\10" for linefeed). If you type in a hex character (without quotes) it will be converted to the appropriate 'real' hex character for your ping.

Notes:

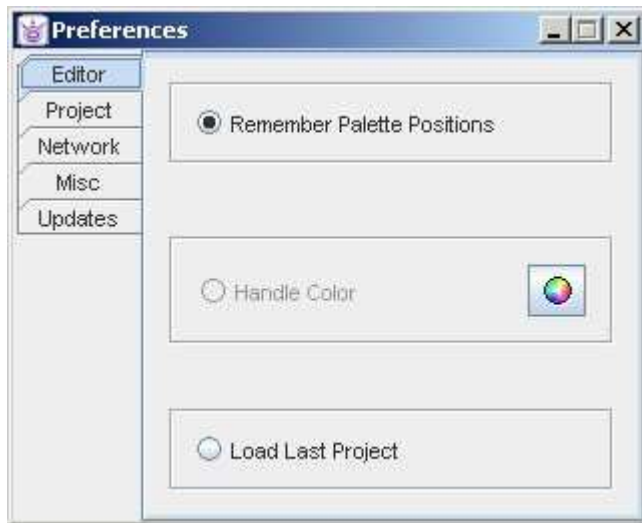
- The ping happens every time the heartbeat indicator turns green (pulses).
- You can disable the ping by deleting the ping string.

Object Notes



A simple text editor allowing notes for each object is included. You also have very basic 'search' and 'replace' features as well. The text area will automatically change as soon as a new object is selected, so you need to 'Save' your text into the object before you select any other object, or even select another palette (you may cause a page refresh).

Editor Preferences



Editor Settings

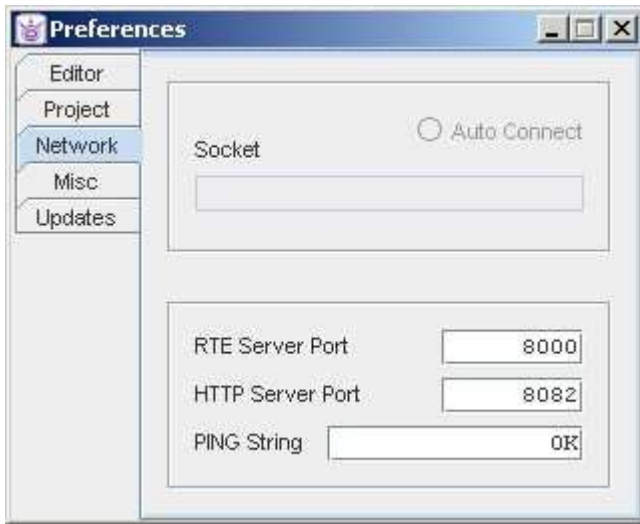
- Remember Palette Positions

This is selected by default. Disabling this will reset all position information.

- Load Last Project

You can have the last project loaded automatically by selecting this option (last project opened).

Note: Some of your settings are automatic (Image directory, last project folder, etc.).



Network Settings

- RTE Server Port

Change the RTE startup server port here (default is 8000). If you have the script window open by default, the message field will display the server startup message:

RTE Server started at Thu Jan 31 10:04:30 CST 2008

Listening on Port: 8000...

- HTTP Server Port

You can change the default HTTP server port number here. This is for a future release.

- PING String

Once a connection is made, the RTE will start pinging with a 'Ping String'. There are two reasons for this:

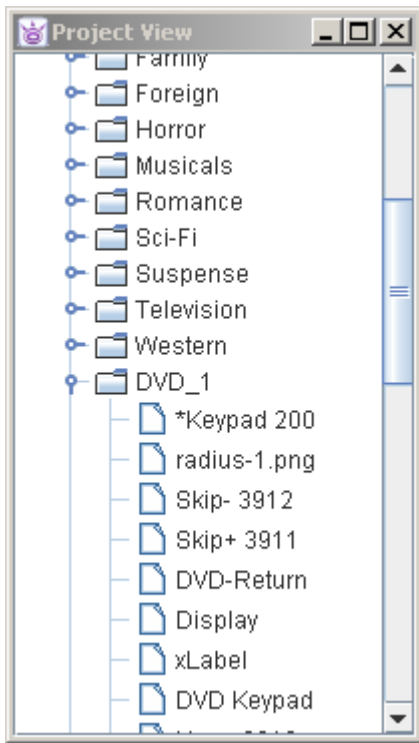
- 1) To keep the socket alive.
- 2) To keep the device we are connecte to alive.

You can change the ping string in the Preferences window (Network section) by typing in a text string, or using the standard notation for hex with a backslash ("\10" for linefeed). If you type in a hex character (without quotes) it will be converted to the appropriate 'real' hex character for your ping.

Notes:

- The ping happens every time the heartbeat indicator turns green (pulses).
- You can disable the ping by deleting the ping string.

Project View



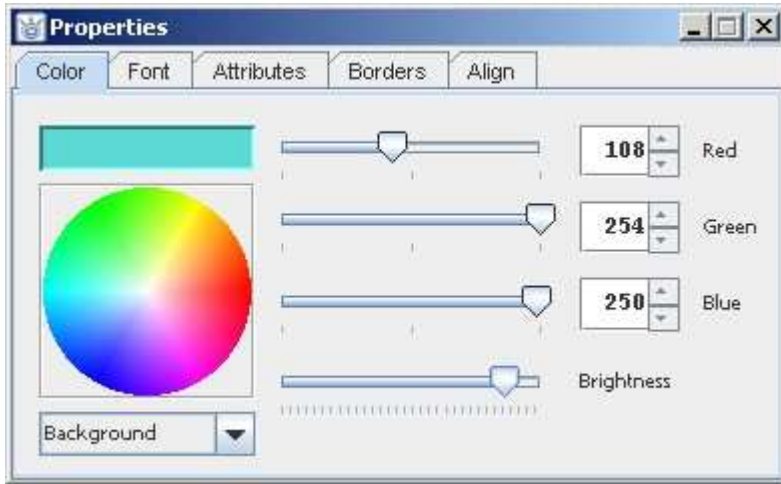
The Project View (accessible from the 'Windows' menu) gives you a hierarchical view of each open project window, and all pages/objects in that project. Clicking on a project/object page will bring that window forward, and go to that page. If you have the 'Select Tool' selected and you press an object, the handles will show for that object, giving you a quick visual reference.

The list does not allow any editing at this time, but will refresh to reflect changes in your project (added/deleted objects). It makes an attempt to maintain your folder expanded views, but you may have to re-expand folders to get back to where you were.

Note:

When you see an asterisk before the name (like *Keypad 200 in the picture) the object is hidden. You can double-click on the object to make it visible.

Property Editor



The Property Editor is the main palette for making changes like color, font, border, etc. Depending on the object type, each section may have different controls enabled/disabled. For example, the slider orientation controls (on the 'Align' tab) are only active when a slider control is selected.

**Note: This palette replaces the 'Borders', 'Color Selector', 'Font Selector', and 'Property List' editors and their functionality.*

Those of you familiar with the Run Time Editor 2 will recognize this Editor right away, but there are a few differences. You have several new controls for attributes that were not available before, and some controls are not yet available. However, all of the important controls are here, and they are live as usual. The term 'live' refers to editing while either in the edit mode (handles visible) or at any time during runtime use. This is a very important distinction because there is no undo available during live editing, and you should be aware of this.

The tabs are laid out as intuitively as they are in the RTE2, and there is a new tab called 'Borders' that replaces the borders palette in older versions of the RTE3. However, since this editor also gives you the important feature of text and icon alignment (found in the 'Align' tab section), the 'Property List' palette is no longer included. The only control feature the property list provided was for this type of alignment.

Color

You can change the current color of the controls 'Background' or 'Foreground' using this tab section. By default, background is selected, and it does not matter how many controls are selected (i.e. you can change multiple controls at one time). When you click on an object (either in runtime or select mode) the sliders and fields will update to provide the current values for that mode (depending on the mode selected). There are four methods for altering the current color:

1) Color wheel

Just 'mouse around' on the color wheel (mouse down) to change the color. When you release the mouse, the sliders and fields will update.

2) Sliders

You can control individual R G B color channels by using the respective slider. The fields will update in real time.

3) Spinners and Fields

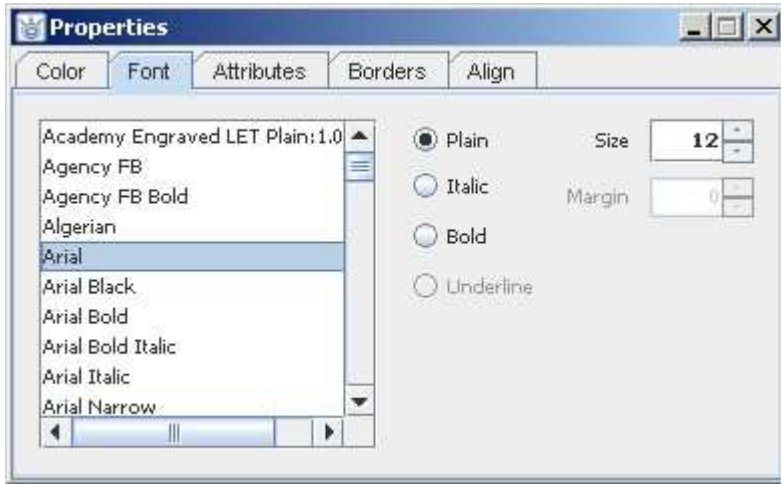
The spinner fields are using for small increments, or you can input values directly into the fields (and press 'Enter').

4) Brightness

The brightness control will affect the current R G B color as a whole.

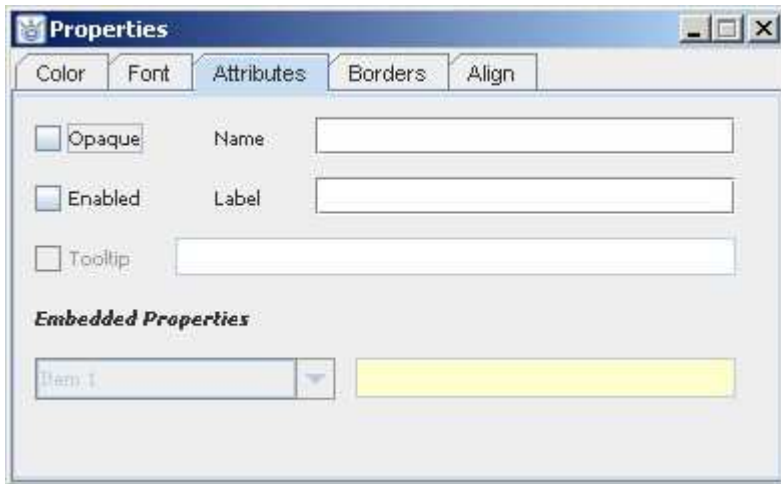
The small color section above the color wheel will change to show you the current color.

The only choices that are enabled in the combo drop down are 'Background' and 'Foreground' at this time.



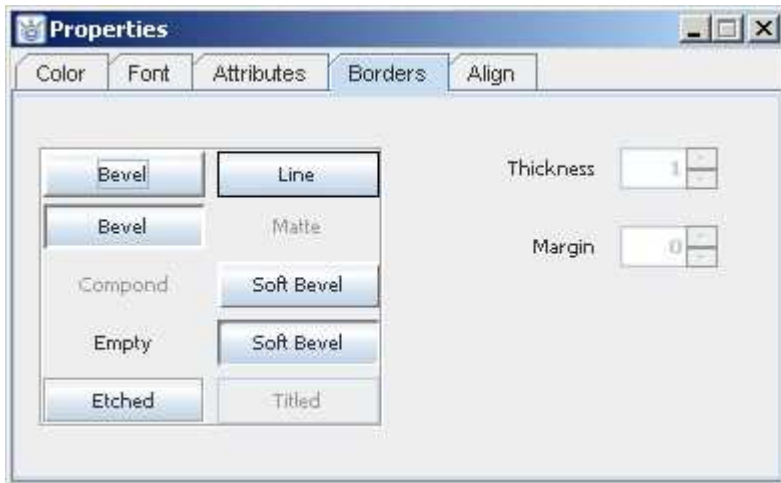
Font

The font list will update every time an object is selected to reflect the objects font and style. You can have multiple objects selected.



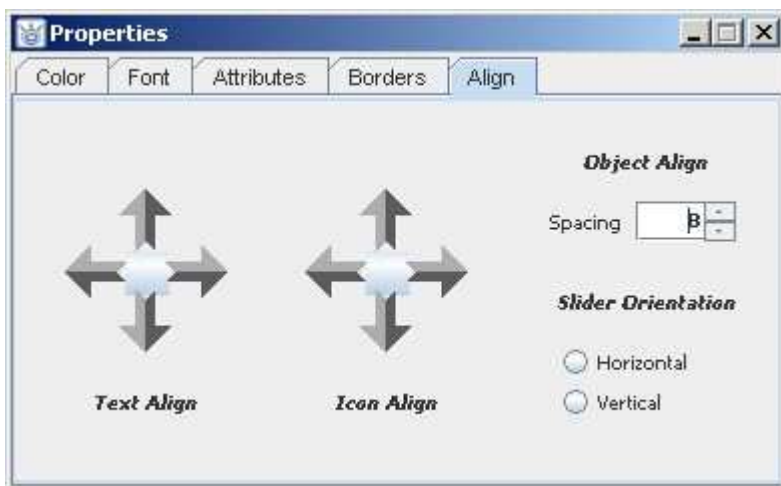
Attributes

You can set the name and label here, as well as the 'Opaque' and 'Enabled' values. Several controls are disabled on this tab, and are used as placeholders for future control. You can have multiple objects selected for many of these controls, but the 'Name' field will only affect the first selected control (objects must have unique names, but you can have the same label for many objects).



Borders

This tab section replaces the Borders window, and there are several border buttons that are disabled, but used as placeholders for future control. Border line attributes are not editable at this time, but you can typically achieve the effect you want by using different methods for now (like layering or images).



Align

This section is dedicated to unique alignment for buttons that is typically not available:

Text Align

You can align just the text of an object by left, right, top, or bottom. If an icon is assigned, the icon will shift to accommodate the new text alignment but this is not the same as the 'Icon Alignment'.

Icon Alignment

This alignment actually aligns everything, but it will include the text alignment as part of the overall alignment. For example, if you have text to the left of an icon and you select the left arrow, both the text and the icon will shift to the left, but only as far as the left most letter in the text.

Object Align

This value affects the spacing of the objects when you use the align button on the 'Align' tool palette.

Slider Orientation

The controls associated with slider orientation are only available if a slider control is selected. By default, a new slider has a horizontal orientation.

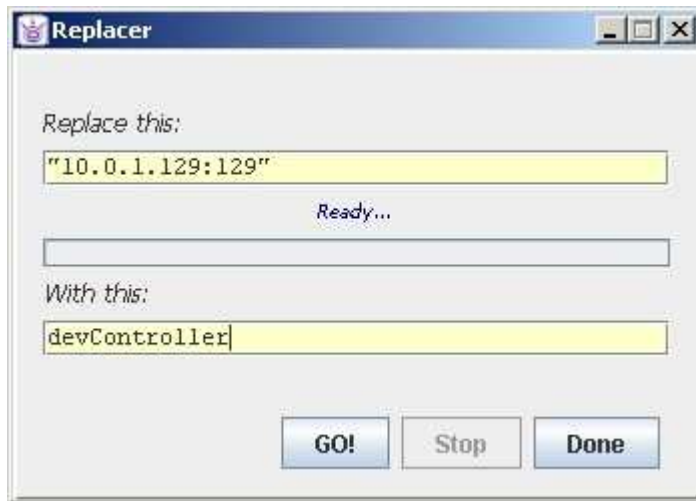
Note: You can set the stretched image from the 'Image Selector' palette, or from a script:

```
SET.IMAGE("myButton", "path");
```

or set the property directly:

```
SET.PROPERTY("myButton", "IMAGE", path);
```

Replacer



The Replacer is a simple but powerful tool for replacing any text with any other text. You will need this for managing changes in all of your scripts. Pressing 'GO!' will start the replacement process, and the progress bar and status field will provide you with status. Press 'Done' to hide this palette.

Script Debugger

The screenshot displays the Script Debugger application window. The title bar reads "Debugger". The menu bar includes "Edit", "Debug", and "Window". Below the menu bar is a toolbar with buttons for "Break", "Go", "Step Into", "Step Over", "Step Out", and "Save Script".

The main area is a code editor titled "Update: RELEASE_SCRIPT". The code is as follows:

```
1  /* see PUSH script for setup logic */
2  var myFiles;
3  var gDiscs;
4
5  ↘ if(myFiles != null){
6    list = myFiles.split("\n");
7    myList = null;
8    gDiscs = null;
9    count = list.length;
10
```

The cursor is positioned at the start of line 5. Below the code editor is a watch window with the following table:

Expression	Value
myList	null
gDiscs	001 004 002 001 007 1103 004 1081 1326 1138 ...
myFiles	007 Collection Goldfinger Special Edition 007 C...
count	58

At the bottom of the watch window are "Watch" and "Evaluate" buttons. The status bar at the very bottom shows "Thread: Thread[Thread-115,6,main]".

The Script Debugger differs from the Script and Messages Window in several very important ways, but the most significant is the ability to 'step' through your code as well as look at variable values while the script is being evaluated. **The first time you open this debugger (or if the window is restored with "Remember Palette Positions"), it will be in the intercept mode...that is, it will be waiting to step through the first script that it is asked to interpret. You need to press the 'Go' button (or step through) in order for any script to be evaluated. This is actually very important because nothing will be evaluated otherwise.** After that initial setup, your scripts will trigger normally. If you want this behavior later for evaluating a script (i.e. intercept), you will select "Break on Function Enter" from the file menu of the Debugger.

The reason that this is separate in this version of the RTE is because RCS is using the Mozilla Rhino scripting engine for some of the interpretation. The license for this integration does not allow complete integration at this time, but this may change in the near future. The 'other' script window may also become the main script debugger in a future version of the RTE (and it does have colorized syntax!) so you may want to stay away from explicit functionality of the Rhino Script engine. Use the example projects as a guideline for what will work, and for good programming practices to use for the RTE.

Some of the Rhino functionality has been disabled purposely so that you do not develop a programming style that becomes dependent on features only in this engine, so if something you are trying to do based on the Mozilla documentation is not working, it may be because of this. Mostly, things like object creation, serialization, script evaluation, class loading, and streaming have been disabled so that you focus on the RTE scripting language and syntax.

The instructions for using this debugger do not exist (as far as we know) and the following instructions are for use with the RTE only:

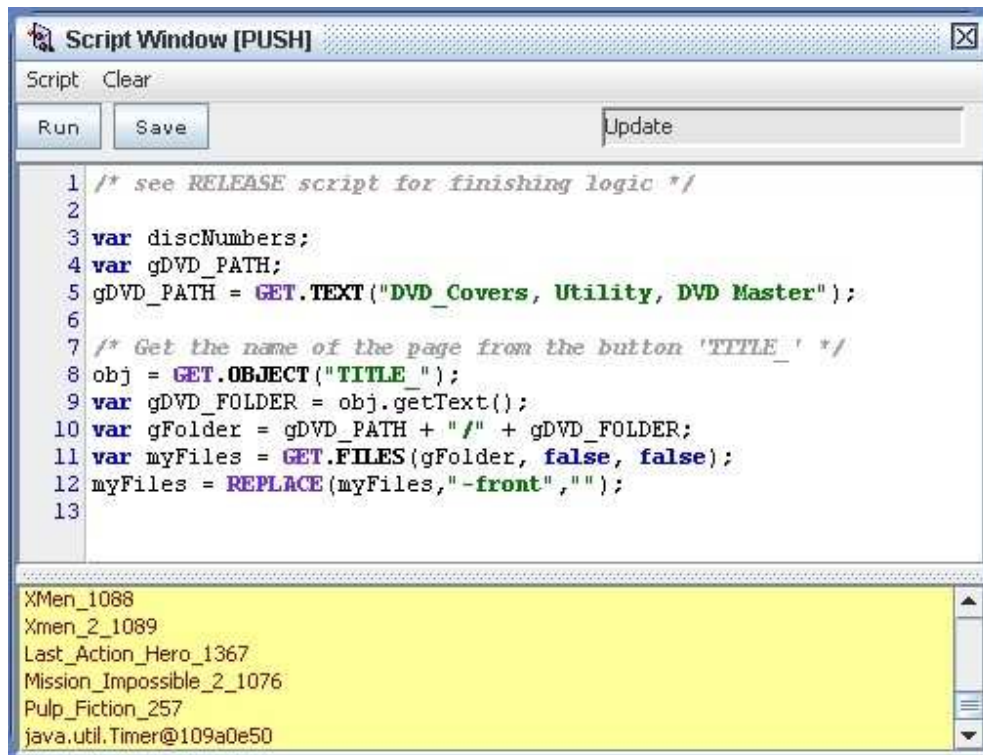
(To step through a script)

From the 'Debug' menu, you can choose the 'Break on Function Enter' (remember to turn this off when you are done!) to have the debugger automatically go into the step mode on the next trigger, or when you press the 'Run' button on the RTE Script Window. The theory of this debugger goes something like this:

- For every script triggered, a new evaluation window is opened, and will stay opened until closed.
- The foremost script window is the current 'scope' for the rhino debugger, but triggering things in the RTE will take this scope away.
- Right clicking on the topmost window will popup the menu where you select 'Run' to begin stepping through code if you are not already stepping.
- You can 'Tile' or 'Cascade' all open windows (every script that has ever been triggered will open a window) and you can access a list to open a particular window (or bring it to the front) from the 'Window' menu.
- The 'Save Script' (if enabled) will actually save the current script in the topmost window to the object related to that window (the title indicates the object name and the script type). You can edit in this window, but it would be a good programming practice to use the 'other' window for now.

RCS hopes to have better integration for this debugger in the next release, and eventually the RTE Script Window will replace this totally.

Script Window



The screenshot shows a window titled "Script Window [PUSH]". At the top, there are buttons for "Run", "Save", and "Update". The main area contains Java code with line numbers 1 through 13. The code defines variables for disc numbers, DVD paths, and folder names, and performs operations like getting text from an object, getting files, and replacing text. Below the code, the output is displayed on a yellow background, listing several movie titles and a timer object.

```
1  /* see RELEASE script for finishing logic */
2
3  var discNumbers;
4  var gDVD_PATH;
5  gDVD_PATH = GET.TEXT("DVD_Covers, Utility, DVD Master");
6
7  /* Get the name of the page from the button 'TITLE_' */
8  obj = GET.OBJECT("TITLE_");
9  var gDVD_FOLDER = obj.getText();
10 var gFolder = gDVD_PATH + "/" + gDVD_FOLDER;
11 var myFiles = GET.FILES(gFolder, false, false);
12 myFiles = REPLACE(myFiles, "-front", "");
13
```

XMen_1088
Xmen_2_1089
Last_Action_Hero_1367
Mission_Impossible_2_1076
Pulp_Fiction_257
java.util.Timer@109a0e50

The Script and Messages window is the most important window in the RTE. From this one window you will not only edit and save all of your scripts for all objects, but you will also receive feedback from the RTE in the Messages window. You set the script type from the drop down menu, and the script field will change to reflect any script that has been written for that object according to this setting. Very simple syntax highlighting and a number line has been integrated to help you in troubleshooting and debugging your script.

The buttons are:

Save - This will save the current text in the Script window to the current object (indicated in the Object field) according to the menu setting (the checkbox that is selected). The Messages field will provide feedback for status.

Run - You can use the Script field as a simple testing area for anything (just don't press Save). Anything that is in the Script field can be Run as a script when you press the Run button, and any errors will show up in the Messages field.

Auto - Selecting this button will allow automatic saving of every script (for the current type that is selected). If you have the 'RELEASE' type selected, then every script that is evaluated will be saved because everything is constantly being evaluated. This can be a great convenience, but it can be a little dangerous.

The suggestion is to use it sparingly (during simple editing, and specific 'multiple object' quick editing), and turn it off during 'real' debugging (like stepping), and cutting and pasting. Because this is a 'live' environment at all times, your 'scope' is constantly changing, and it is possible to wind up in an unexpected or 'unknown' state on rare occasions.

The Menu

The menu options are:

Push - Triggered when the object is pushed.

Release - Triggered when the object is released.

Repeat - Triggered when the object is held.

DPush - Triggered when the object is double-pushed.

Open - Triggered when an object is opened (this only works for pages at this time).

Close - Triggered when an object is closed (this only works for pages at this time).

Main - Every object has the capacity to hold large scripts and functions that you can call from other scripts. You can think of this as a placeholder for functions that you will call when you need them, but are relevant to the object.

Loop - Another 'placeholder' type script, but intended for managing any loop operations for the object. You 'call' this script in the same manner that you call your other scripts (DO.LOOP) and you can have the loop call itself with a delay to create your own custom thread (all calls are threaded). Make sure you create

a variable 'flag' or similar to control the loop in case you create an infinite loop.

The syntax for most calling and triggering is :

```
DO.PUSH("name, page, window", delay);
```

The first parameter is a string variable, and the only parameter you need is the 'name' (object name) if the object is on the current page. For any other advanced calling (i.e. calling objects on other pages or windows) you must use the other parameters.

Notes:

You can edit the name (real name) of the current object in the name field...press 'RETURN' to save.

Toolbar



The Toolbar is used for selecting the mode, and for selecting the type of object you need to create. When the 'Hand' is highlighted (green) you are in the 'Run' mode (objects will not be editable). When you have the arrow selected (the Edit mode) you can select objects for editing.

Special Features:

- Holding the 'Alt' key down before pressing a button will short-circuit the script parsing (i.e. your script will not trigger). This is useful for selecting an object without triggering the logic!
- Pressing the 'Run' tool (the Hand) will reset much of the runtime editing environment to default settings (like undo/redo)...even if it is already selected.