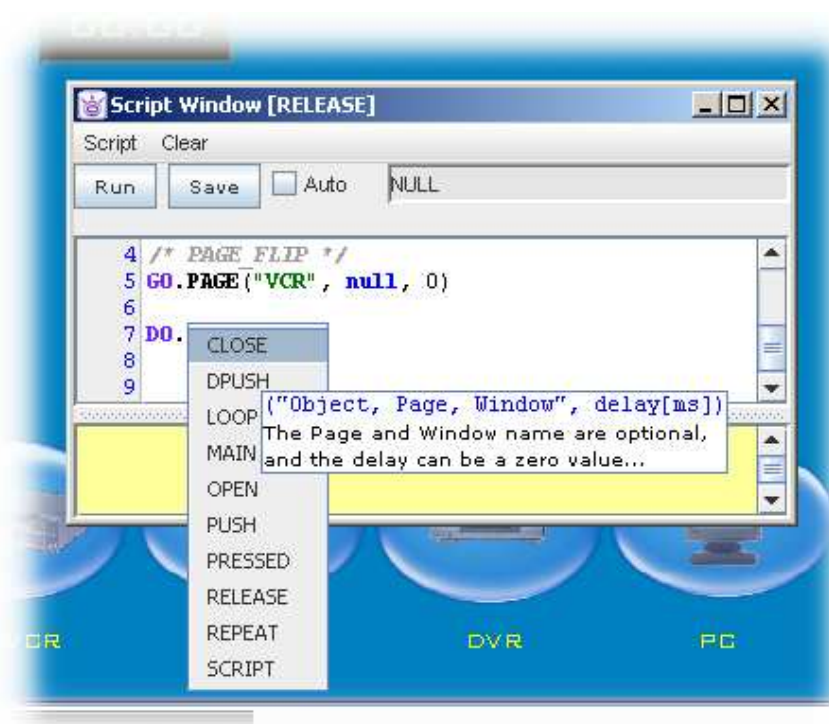


# xScript Language



*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("ISDNN0", 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