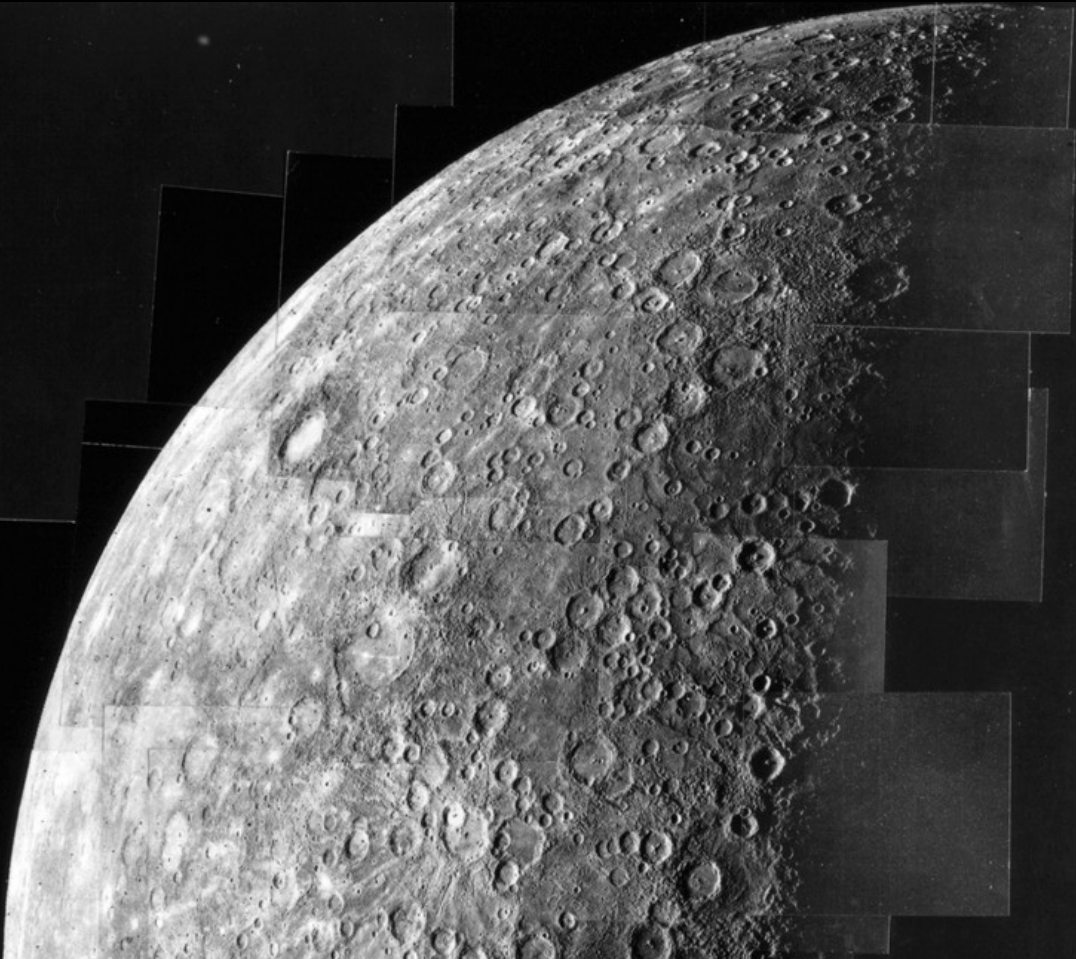


# RunRevPlanet Search

RunRevPlanet Search adds fast and flexible searching to your reference, help and content based applications. Find matching cards by searching on multiple terms with and, or and not conditions. Add powerful searching to your LiveCode applications for iOS, Linux, Mac or Windows.

## Guide and Reference



# RunRevPlanet Search

## Guide and Reference

Copyright (c) 2011 Scott McDonald PC Services  
All rights reserved.

February 2011 Edition



**Address:** Scott McDonald PC Services  
PO Box 139, Newtown, 2042  
New South Wales, Australia  
**Facsimile:** +612-9564-2347  
**Website:** <http://www.runrevplanet.com>  
**Email:** [runrevplanet@smpcs.server101.com](mailto:runrevplanet@smpcs.server101.com)

# Table of Contents

Introduction.....	3
Installation & Requirements.....	6
Conventions.....	7
Licensing Agreement.....	8
Coding a Search.....	10
Simple Search Demo.....	12
Advanced Search Demo.....	18
iOS Search Demo.....	23
Searches and Queries.....	24
Building an Index.....	27
Deploying Your Own Application.....	29
Error messages.....	31
Handler Reference.....	32

## About the Cover Image

The image on the cover of this RunRevPlanet Search Guide and Reference is a public domain image from the NSSDC (National Space Science Data Center) Photo Gallery. Many thanks to NASA and the NSSDC for making this image, and many more images, available to the public.

This mosaic of Mercury was taken by the Mariner 10 spacecraft during its approach on 29 March 1974. The mosaic consists of 18 images taken at 42 s intervals during a 13 minute period when the spacecraft was 200,000 km (about 6 hours prior to closest approach) from the planet. The resolution is about 2 km. The light-floored crater surrounded by darker material towards the edge is Lermontov, diameter 160 km. The original image m10\_aom\_18.tiff can be found at:  
[http://nssdc.gsfc.nasa.gov/imgcat/html/object\\_page/m10\\_aom\\_18.html](http://nssdc.gsfc.nasa.gov/imgcat/html/object_page/m10_aom_18.html)

## About RunRevPlanet

The RunRevPlanet website is an initiative of Scott McDonald PC Services. Our goal is to make RunRevPlanet one of the top websites dedicated to LiveCode and an invaluable source of components, controls, tools and resources for LiveCode developers. Visit us at:  
<http://www.runrevplanet.com>

RunRevPlanet is not officially affiliated with RunRev Ltd.

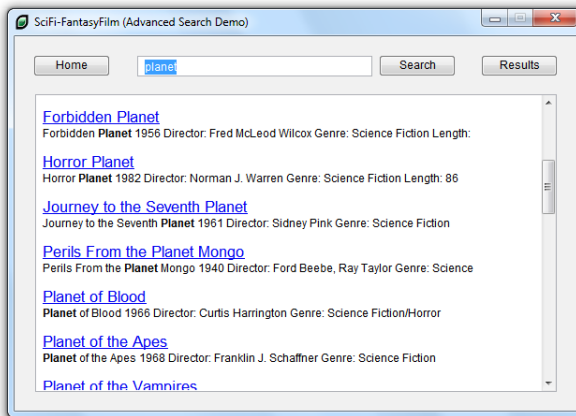
# Introduction

RunRevPlanet Search is a powerful search tool for your LiveCode applications. The RRP Search is written in 100% LiveCode script and does not use any platform specific libraries. This means you can add a powerful search ability to your cross platform application for iOS, Linux, Mac or Windows.

RRP Search is simple to use and requires only a few lines of script to add flexible searching to your LiveCode application. By using RRP Search (RRPS) you can focus on the content of your application and not be concerned with the complication of making the information easily accessible to your user.

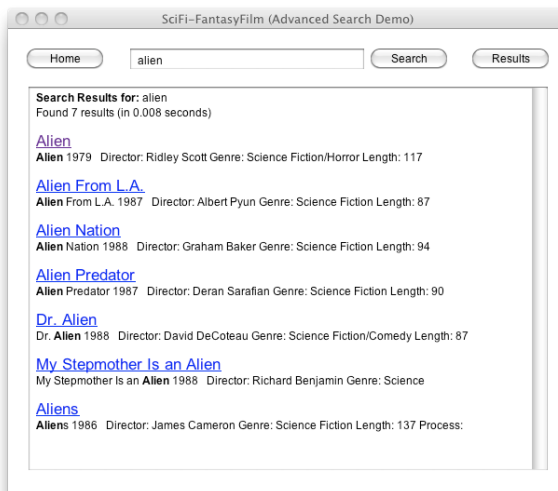
RRPS is fast because it uses an index that is built during development that becomes part of your application. This means your user does not need to wait long for results, even with stacks containing thousands of cards.

RRPS is flexible, unlike the find command in LiveCode, RRPS handles queries with words that can be combined with and, or and not conditions. Using RRPS is as easy as using a search engine on the web, and will be familiar to your users.



RRPS returns all matching cards as a single list, optionally context showing the context of each matching word. This makes it ideal for help engines and other applications where your user is searching for information and seeing all the possible cards helps her find what is being looked for.

As well as allowing flexible queries, RRPS provides more control over how words are indexed. RRPS does not use the default LiveCode definition for a word, instead you specify the delimiters used to determine what is considered a “word”. Unlike the LiveCode find command, RRPS handles new line and return character as you would expect.



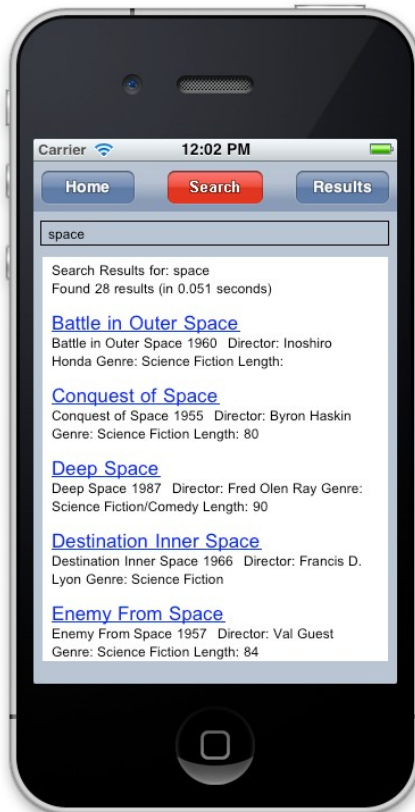
Just like Google search and other search engines, RRPS uses stop words. These are common words that do not need to be indexed and so speed up searches. For flexibility, RRPS lets you set the stop words, which you can add to or have no stop words at all.

Lastly, RRPS optionally shows the result of a query in a web search engine type of format with the link to each card that matches the query, and with text showing the context of each match.

RRPS can be used in stacks ranging from Help files, Reference works, eBooks, Course content, Educational stacks any application where flexible, fast access to the content will impress your users.

## Features of RRP Search:

- Build an index for fast searches at runtime
- Handling of stacks with thousands of cards without slowing down queries
- Handling of queries with an and condition
- Handling of queries with an or condition
- Handling of queries with an not condition
- Use of quote to specify queries with exact matching
- Set custom delimiters to define a word
- Use of stop words to speed up searches and minimise index size
- Returns all results with one query
- Search results can be returned in a format similar to a web search engine with a link and context
- Does not require using front or back scripts
- Mobile platform support for iOS
- 100% LiveCode script



# Installation & Requirements

To use RRPS you must have the following:

- Revolution 3.0, or newer, in the Studio or Enterprise edition.
- LiveCode 4.5, or newer.
- Using the iOS demo stack requires LiveCode 4.5.3, or newer.

RRPS may run with earlier versions of Revolution and LiveCode, but version 3.0 or higher is recommended. RRPS is distributed as a single ZIP archive containing these files:

```
rrpSearch.rev  
rrpSearch-Guide-Reference.pdf  
Readme.txt  
License.txt  
/Demos/Demo-Search-Advanced.rev  
/Demos/Demo-Search-Advanced-iOS.rev  
/Demos/Demo-Search-Simple.rev  
/Demos/Demo-Search-UseList.rev  
/Demos/SciFi-FantasyFilm.rev
```

## Installation

To install RRPS, unzip the files above into a folder of your choice.

- \* Do not unzip the files into your LiveCode program folder, instead put them into a convenient folder where you work in documents.

After unzipping the files, each time you start a new LiveCode application that uses the Search you make the rrpSearch.rev file a substack your application main stack. While keeping the original rrpSearch file intact.

# Conventions

The terms “rrpSearch stack”, “rrpSearch.rev” and “RRPS” may be used interchangeably in this guide depending on the context. Each of these names refers to RunRevPlanet Search.

The handlers in RRPS are all commands. Throughout this guide the term handler or command may be used interchangeably. In some, cases the commands are used like functions by using the result function.

The word LiveCode refers to the LiveCode IDE (Integrated Development Environment) and may be used interchangeably with the term IDE.

LiveCode script is shown in a fixed width font with the same formatting shown in the IDE Script Editor. An example of a script is shown below.

```
command SearchQuery
  local searchFor, context

  put field "Query" into searchFor
  call "rrpSearchQuery searchFor" of stack "rrpSearch"
  call "rrpSearchResultLinks" of stack "rrpSearch"
  put the result into context

  set the HTMLText of field "SearchResults" of ↵
    card "Results" to summary & context
  go to card "Results"
end SearchQuery
```

The ↵ symbol is used to indicate single lines of script that are too long to fit in the width here, and so are broken over two or more lines, but can be entered as a single line in the Script Editor. Scripts that are entered in the Message Box are also shown with a fixed width font but without any formatting such as below.

```
answer the cVersionNumber of stack "rrpSearch"
```

- 🔍 A paragraph with this symbol highlights a point that could be unexpected behaviour, or a potential problem that may not be obvious at first.

# Licensing Agreement

The RRP Search software and accompanying files and documentation are protected by Australian copyright law and also by international treaty provisions. Any use of this software in violation of copyright law or against the spirit of the terms of this agreement will be prosecuted.

RRP Search is Copyright (c) 2011 Scott McDonald PC Services, all rights reserved.

Scott McDonald PC services authorizes you to make archival copies of the software for the sole purpose of backup and protecting your investment from loss. Under no circumstances may you copy the software and documentation for the purpose of distribution to others. Under no conditions may you remove the copyright notices from the software or documentation.

You may use an unlicensed copy of the RRP Search to evaluate the RRP Search for an unlimited time. You may not build or distribute a standalone application that uses the RRP Search without first purchasing a License Key from Scott McDonald PC Services at the RunRevPlanet website, or without first purchasing an Unlock Code from an approved vendor.

You may distribute, without run-time fees or further licenses, your own executable applications based on the RRP Search and the demonstration files after you have purchased a License Key or Unlock Code. You may not distribute applications that use the RRP Search with your License Key or Unlock Code in an unencrypted stack file.

When distributing your own executable applications based on the RRP Search you must encrypt with a secure password any stack that includes your License Name and Key or Unlock Code. You may not distribute your License Key or Unlock Code in a separate file with your application, or through other media such as Internet, email or print.

The previous restrictions do not prohibit you from distributing your own source code or stacks that depend on the RRP Search. However others who receive your scripts need to download their own copy of the RRP Search and purchase a License Key or Unlock Code in order to write programs that use your own code.

The RRP Search may be used by one person on as many computer systems that person uses. We expect that group programming projects making use of the software will purchase a license for each member of the group. In such cases, volume discounts may apply to site licensing agreements.

The RRP Search will perform substantially in accordance with the accompanying documentation, and technical support by Scott McDonald PC Services will make commercially reasonable efforts to solve any problems associated with the RRP Search. If you encounter a bug or deficiency, we will require a problem report detailed enough to allow us to find and fix the problem.

In no event will Scott McDonald PC Services or anyone else who has been involved in the creation, development, production, or delivery of the RRP Search be liable for any direct, incidental or consequential damages, such as, but not limited to, loss of anticipated profits, benefits, use, or data resulting from the use of this software, or arising out of any breach of warranty.

By using this software you agree to the terms of this Licensing Agreement. If you do not agree, you should immediately erase all your copies of the RRP Search and apply for a refund if you have purchased a Licensing Key or Unlock Code. Scott McDonald PC Services provides web-based and email support for the RRP Search on an "as available" basis at no extra charge. When you send an email to technical support we will try to answer your support question within 48 hours.

Built with LiveCode. Portions (c)2000-2011 RunRev Ltd, All Rights Reserved Worldwide. You should review the License Agreement in your copy of LiveCode when building your own applications with LiveCode and the RRP Search.

All names of products and companies used in this Licensing Agreement, the RRP Search, or the documentation may be trademarks of their corresponding owners. Their use in this Licensing Agreement is intended to be in compliance with the respective guidelines and licenses.

# Coding a Search

To use RRPS you need to add some LiveCode to your application. A detailed description of the steps when coding is in the next two Tutorial chapters, but this chapter is a brief overview that outlines the basics.

There are two different aspects to coding RRPS, there is:

- The LiveCode that is executed to build the index, which is done during development and can normally be removed from your application.
- The LiveCode required to perform searches by the users of your application.

## Building the index

Firstly you need to initialize the stack. This is done with the following statement.

```
call "rrpSearchInitialize" of stack "rrpSearch"
```

This must be done only before building the index. Once the index is built you should only call this again if you need to re-build the index, because `rrpSearchInitialize` clears the index.

Then you must set the stack and cards to be included in the index.

```
call "rrpSearchIndexStack MyStack" of stack "rrpSearch"  
call "rrpSearchIgnoreCard MyStack,HomeCard" of stack "rrpSearch"
```

Here all of the cards in the “MyStack” stack, except for the card named “HomeCard” will be included in the index. Only the cards in the index are searchable by RRPS.

Lastly the index is built.

```
call "rrpSearchBuildIndex" of stack "rrpSearch"
```

This call may take a while to complete as the index is built if there are many cards (thousands) with a lot of text. The actual time required depends on the amount of content and the speed of your processor, but it could take tens of minutes in some cases.

The index is stored as a custom property array of the `rrpSearch` stack. Once the index is made, this code does not need to be executed again, and can be removed from your application if the searchable stack content is static.

## Doing a search

A search is done, by executing a query. A query is a single call.

```
call "rrpSearchQuery myQuery" of stack "rrpSearch"
```

In this example myQuery is a variable that holds a text string that specifies the query. The query string is text that RRPS uses to find matching cards that fulfil the search criteria. Details of query strings are in the Searches and Queries chapter.

All RRPS commands return value with the results function, so a simple example of displaying the list of matching cards after a query is shown below

```
call "rrpSearchQuery searchFor" of stack "rrpSearch"  
answer the result
```

The rrpSearchQuery returns a simple list of the name of the card and stack that matches the query, and the score which is the numbers of matches made on the card. This information can be process further in your application depending on the purpose of the search.

If you need to present the result in a way that is similar to a web search engine, which is familiar and easy to understand for many users, the rrpSearchResultLinks command returns a HTML string that is suitable for setting to the HTMLText property of a field.

```
call "rrpSearchResultLinks" of stack "rrpSearch"  
set the HTMLText of field "MyResults" to the result
```

This command is called after doing the query. The HTML includes links to simplify the showing of the card with each result. Adding a linkClicked handler to the field showing the contextual results allows a matching card to be shown with a mouse click.

```
on linkClicked pLink  
  call "rrpSearchGoLinkCard pLink" of stack "rrpSearch"  
end linkClicked
```

# Simple Search Demo

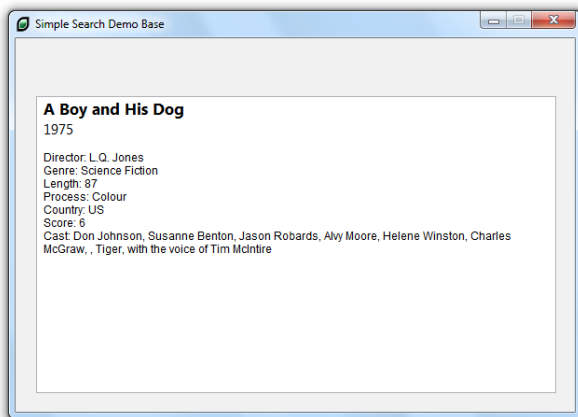
RRPS is a self-contained stack that you can use to add powerful searching to any LiveCode application. This chapter explains the process of adding a search facility to a simple stack. This demo is not a finished application, but shows the bare minimum to use RRPS. In the next chapter this simple demo is refined to make it easier to use.

Even though this demo is not a polished application, it is important to understand the basics of using RRPS before adding it to your own applications. It only takes a few minutes to work through this demo, and there is a completed copy of the stack named Demo-Search-Simple.rev in the Demos folder.

It is recommended that you carefully follow these steps the first time you use RRPS. Once you understand the process you are then ready to add RRPS to your own applications.

To properly illustrate the use of RRPS you need a stack with content for searching. For this demo there is a stack ready-made with 753 cards of information to use as a basis for this tutorial.

This stack is supplied to give you a starting point for this tutorial. The stack is called SciFi-FantasyFilm.rev and each card contains a field with the details of a science fiction or fantasy film between the years of 1940 and 1989. (This stack is not exhaustive and is a selection of notable films from that 50 year period.)



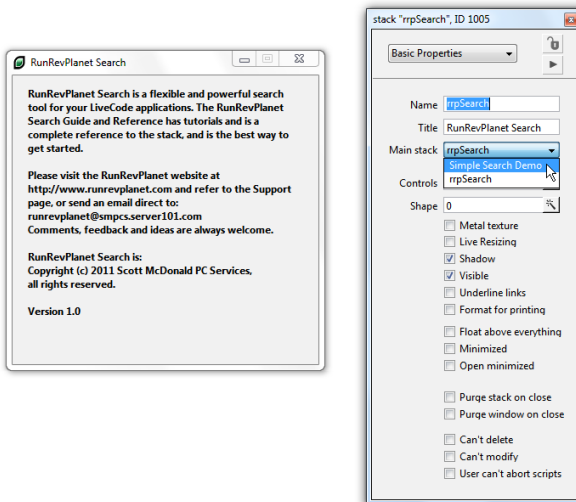
Choose the Open stack command in the File menu. Navigate to the Demos folder and select and open SciFi-FantasyFilm. In the open stack, you can see the first card with information about a film.

By pressing control-3 (command-3 on Mac), the keyboard shortcut for the Go Next command in the View menu, you can go to the next card. Repeatedly pressing the shortcut shows each card in the stack one after the other. In this tutorial, you add a new card to this stack with buttons to allow you to test RRPS.

To keep the original SciFi-FantasyFilm Stack unchanged you should first save it with a different name. Press control-k (command-k) to show the stack Property Inspector. In the Name field enter “Simple Search Demo” and change the Title if you want.

Choose the Save As command in the File menu and navigate to a convenient folder before clicking on the Save button. You can save it in any folder that is convenient for doing this tutorial, including the Demos folder. Now the rrpSearch stack must be made a substack of your Simple Search Demo mainstack.

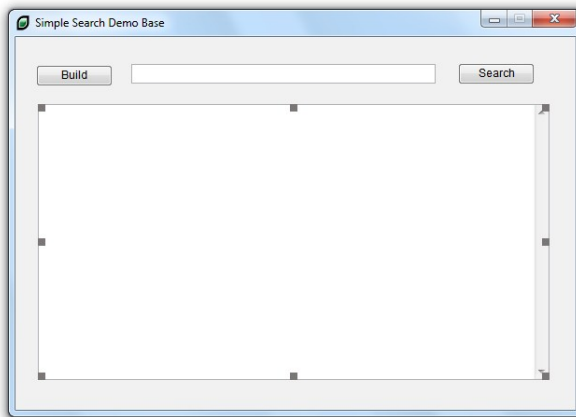
Choose the Open command in the file menu and navigate to the folder where you have installed RRPS. Select and open rrpSearch. Press control-K (command-k) to show the stack Property Inspector. In the Main stack drop down menu select “Simple Search Demo” as the main stack. (Assuming this is the name you gave the stack.)



Save this change, so click on the Simple Search Demo window again and press control-s (command-s) to save the stack. Next a new card is needed for putting search buttons on. Press control-n (command-n) to make the new card.

This new card needs two buttons and a field for entering the query. From the Tools palette drag a Push Button, and then another onto the card. Also drag a Text Entry Field onto the card. You can position them as shown below.

Using the Property Inspector set the names of the controls as “Build” and “Search” for the two buttons, and “Query” for the text field. Next one more field is created to show the results of the query. Since a query result can have multiple lines, drag a Scrolling Field from the Tools palette onto the card. The card should look similar to this.



Resize the last field and in the Property Inspector set the name to “SearchResults”. Now with the card set up with the required controls, some LiveCode is added to make the buttons function. For this tutorial LiveCode is added to the buttons as a way to get the demo up and running quickly.

RRPS uses an index that is created from the content of the cards that will be searched on. Right click on the Build button and from the contextual menu choose Edit Script. In the mouseUp handler add the four lines.

```
on mouseUp
  call "rrpSearchInitialize" of stack "rrpSearch"
  call "rrpSearchIndexThisStack" of stack "rrpSearch"
  call "rrpSearchIgnoreThisCard" of stack "rrpSearch"
  call "rrpSearchBuildIndex" of stack "rrpSearch"
end mouseUp
```

The call command is used because RRPS does not need to be made a back script. In a typical application the number of lines of LiveCode that refer to RRPS are small, and so it is not necessary to use up one of the slots available for putting a stack into the message path.

Here is a brief explanation of what these four calls do. Firstly, RRPS needs to be initialized with `rrpSearchInitialize` which resets all of the properties of the stack and clears any index that may have been previously built. The next two calls specify the cards to be indexed. The call to `rrpSearchIndexThisStack` adds all of the cards in this stack into a list of what is to be indexed. It is important to understand that this command does not actually create the index and only sets up a list of what will be indexed with a later call.

Because the card with the Search buttons and field for results is not part of the contents to be searched, the `rrpSearchIgnoreThisCard` command removes this card from the list of cards to index.

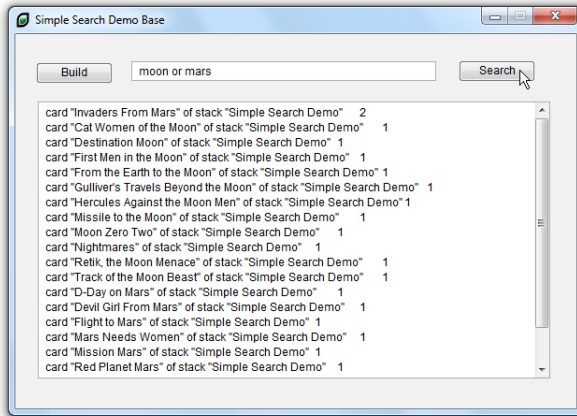
Lastly, the call to `rrpSearchBuildIndex` makes RRPS build the index for the cards you specified after initialization. In this case, this should only take a few seconds, although of course the time taken will vary depending on the amount of content and number of cards that are being indexed. In this case with less than 1000 cards it should only take a moment.

Right click on the Search button and from the contextual menu choose Edit Script. In the `mouseUp` handler add these lines.

```
on mouseUp
  local searchFor
  put field "Query" into searchFor
  call "rrpSearchQuery searchFor" of stack "rrpSearch"
  put the result into field "SearchResults"
end mouseUp
```

The first two lines of this script put the text from the text field into a variable for convenience, and then the third line calls `rrpSearchQuery` to execute the query and make a list of all the cards that match the query. After the call to `rrpSearchQuery` the result is put into the `SearchResults` field for display.

You can now test the demo with what has been done so far. But now is a good time to save the work done so press control-s (command-s) to do a save. Choose the Run (browse) tool from the Tools palette and then click on the Build button of your stack. Once the index is built, experiment with entering a query and clicking on the Search button to view the results.



The results from the `rrpSearchQuery` command are a list with one matching card per line. Each line is two items separated by the tab character. The first item is the name of the card and stack that matches the query, and the second item on the line is the score of the match. The score is the number of matches that were found on the card, so the minimum score is 1 with the number increasing with more matches. The highest score cards are listed first.

Here are four simple queries to try.

```
alien
alien or robot
alien not comedy
"james cameron" or "george lucas"
```

While you are trying these, remember that you need to click on the Search button after entering a query. This simple demo is not set to do the search when you press the enter key. Another point to note is that the `rrpSearchQuery` command only returns the query results, it does not include any mechanism to go to any of the matching cards. The tutorial in the next chapter explains one way to do this, but how the results are used is left to your application to handle.

## Conclusion

Here is a summary of the commands you have learned about in this chapter.

```
rrpSearchInitialize  
rrpSearchIndexThisStack  
rrpSearchIgnoreThisCard  
rrpSearchBuildIndex  
rrpSearchQuery
```

Here is a summary of what you have learned in this chapter.

- How to make RRPS a substack of your application
- How to initialise RRPS to clear the index
- How to include and ignore cards that are indexed
- How to build the index
- How to execute a query
- How to display the query results as a simple list

In the next chapter a more sophisticated demo is examined to learn more about the features of the grid.

- \* After you finish experimenting with this stack it should be closed and removed from memory before examining the demo in the next chapter. This is because the next demo stack also includes RRPS as a substack and having two copies of RRPS open at the same time will show a warning in the IDE.

# Advanced Search Demo

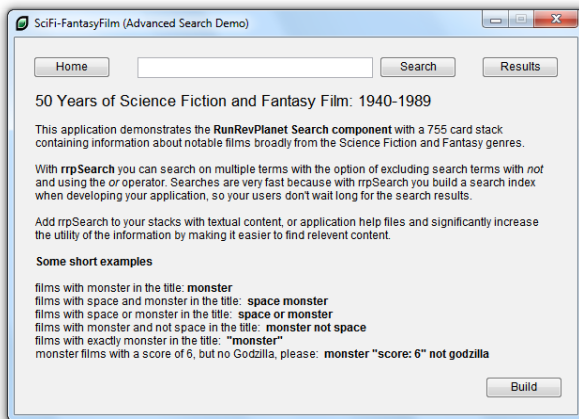
This tutorial refines the demo stack made in the previous chapter. This second demo stack is ready-made and the process of making it is not covered here. Instead the Demo-Search-Advanced.rev stack illustrates more advanced use of RRPS with a practical application to allow searching of the same set of cards about science fiction and fantasy film used in the previous chapter.

This demo can be used as a basis for a complete application where you want to make content available to your users in a way that allows flexible searching of the cards. Such as a help system or other reference tool.

The demo in this chapter uses a group with the backgroundBehavior property set to true. If you are not familiar with the handling of groups it is recommended that you study the Groups & Backgrounds section in Chapter 4 of the *LiveCode Users Guide* to understand the details of the structure of this stack.

All the LiveCode scripts and handlers of the demo are in the stack script. With all the LiveCode in a single location of your stack it is easier to examine the working of the application.

Open the Demo-Search-Advanced.rev stack from the Demos folder. Before examining the LiveCode you can experiment with it by selecting the Run (Browse) tool in the IDE and then trying some queries. In this stack there is no need to explicitly click on the Search button; you can type a query and just press the Enter key.



Demo-Search-Advanced is based on the previous simple demo but adds an additional card the show in the results. There are also additional buttons, one to go into the “Home” which is the first card that includes the Build button, and another button to show the Results card. These buttons are grouped with the Search button and the Query text field, with the `backgroundBehavior` property of the group set to true. This allow a search to be done from any card and makes returning to the results of the search easier.

These improvements are not specific to the working of RRPS and are not discussed further in this chapter, instead the LiveCode for building the index and displaying the results with a link and context are explained.

Right click in a blank area of the Demo Search Advanced stack and choose the Edit Stack Script command from the contextual menu. Here you can examine all of the LiveCode handlers of the demo.

Below is the handler that builds the index.

```
command BuildIndex
  set the cursor to watch
  call "MakeReset" of stack "rrpSearch"
  call "rrpSearchInitialize" of stack "rrpSearch"
  call "rrpSearchSetIndexNumbers true" of stack "rrpSearch"
  call "rrpSearchIndexStack Demo-Search-Advanced" ↵
    of stack "rrpSearch"
  call "rrpSearchIgnoreCard Home,Demo-Search-Advanced" ↵
    of stack "rrpSearch"
  call "rrpSearchIgnoreCard Results,Demo-Search-Advanced" ↵
    of stack "rrpSearch"
  call "rrpSearchBuildIndex" of stack "rrpSearch"
  set the cursor to arrow
end BuildIndex
```

In this handler three new RRPS commands are introduced. By default, RRPS does not index numbers when building the index. In this application numbers are important, for example the year the film is made, or the score given to it. So the `rrpSearchSetIndexNumbers` command is called with a parameter of true to include numbers in the index.

Then `rrpSearchIndexStack` and `rrpSearchIgnoreCard` set the list of cards for indexing. These two commands are more flexible than those used in the simple demo, and allow you to specify stacks and cards that are not necessarily part of the current stack. The `rrpSearchIgnoreCard` command has two parameters, the first parameter is the name of the stack and the second parameter is the card in the stack that you want to have ignored.

The calls to `rrpSearchInitialize` and `rrpSearchBuildIndex` are exactly the same as in

the simple demo.

The next handler shown below actually performs the query and displays the results and is three times longer than the LiveCode from the simple demo.

```
command SearchQuery
  local searchFor, summary, context
  put field "Query" into searchFor
  call "rrpSearchQuery searchFor" of stack "rrpSearch"
  call "rrpSearchResultSummary" of stack "rrpSearch"
  put the result into summary
  call "rrpSearchResultLinks" of stack "rrpSearch"
  put the result into context
  set the HTMLText of field "SearchResults" ↵
    of card "Results" to summary & context
  go to card "Results"
  put searchFor into field "Query"
  put searchFor into field "Query" of card "Home"
end SearchQuery
```

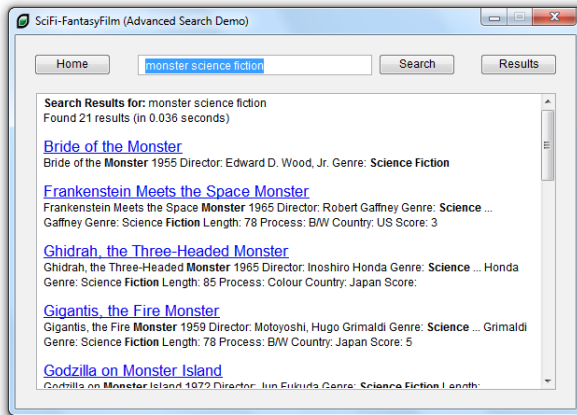
The longer length is partly because of the way the Query text field is in a background group, which requires extra work to make sure that the text of the query is visible on all cards. The other reason for the length is because this demo uses rrpSearchResultLinks to show the results as formatted HTML. The relevant lines after the call to rrpSearchQuery are:

```
call "rrpSearchResultSummary" of stack "rrpSearch"
put the result into summary
call "rrpSearchResultLinks" of stack "rrpSearch"
put the result into context
set the HTMLText of field "SearchResults" of card "Results" ↵
  to summary & context
```

Instead of using the result after calling rrpSearchQuery, two new commands are used to construct a HTML string ready for the SearchResult field. The rrpSearchResultSummary command returns the original query with details about the number of matches found and, optionally, the time taken for the query. This is summary is formatted with HTML and put into the local summary variable.

Then a call is made to rrpSearchResultLinks which returns the results that match the query as HTML string which are put into the local context variable. This variable is combined with the summary to set the HTMLText property of the SearchResults field of the results card.

By stepping through this handler and examining the buffer variable in the Variables pane of the Script Editor you can see the organisation of the HTML. The HTML links are anchored to the name of each matching card and the context of the matches are shown.



With a link as part of the contextual results, the links are used by the next handler.

```
on linkClicked pLink
  call "rrpSearchGoLinkCard pLink" of stack "rrpSearch"
  put field "Query" of card "Home" into field "Query"
end linkClicked
```

LiveCode sends the linkClicked message when a link is clicked in a text field. By handling this message, the pLink parameter is passed to the rrpSearchGoLinkCard command to go to the card. (The second line in this handler not directly related to RRPS and is to stop the query from apparently disappearing from the Search field in the background group.)

This completes the look at Demo-Search-Advanced which can be used as a starting point for your own content based stack with a more flexible search than using the standard find command in LiveCode.

## Conclusion

Here is a summary of the commands you have learned about in this chapter.

```
rrpSearchSetIndexNumbers  
rrpSearchIndexStack  
rrpSearchIgnoreCard  
rrpSearchResultSummary  
rrpSearchContextualResults  
rrpSearchGoLinkCard
```

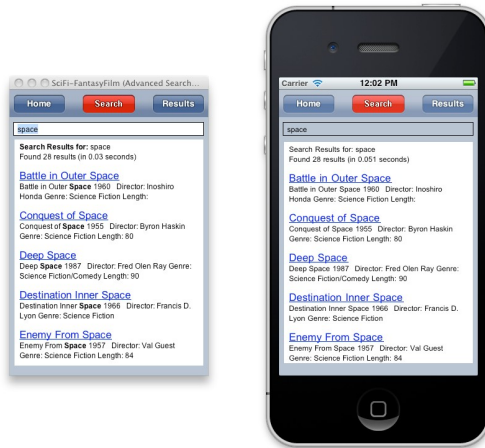
Here is a summary of what you have learned in this chapter.

- How to include numbers in the index
- How to include and ignore cards that are indexed that are not in the current stack
- How to execute a query
- How to display the results of a query in a web search engine style
- How to use links in the results to go to a matching card
- How to make a stack that is a basis for a help system

# iOS Search Demo

This chapter introduces a demo stack based on the demo stack of the previous chapter. This third demo stack is ready-made and the process of making it is not covered here. Instead the Demo-Search-Advanced-iOS.rev stack illustrates use of RRPS in a mobile application with the same set of cards about science fiction and fantasy film, but in this designed for the Apple iOS platform.

- You must have LiveCode 4.5.3, or newer, with the Mobile Deployment add-on for iOS. The Apple XCode and iOS SDK must also be installed to test this demo on the iPhone simulator.



This stack has been modified to suit the look of the iOS platform, but the coding of the searching and the use of RRPS is unchanged.

The demo in this chapter uses the *Scroller Example.livecode* stack in the Mobile Examples folder from the Help, Example Stacks and Resources command of the LiveCode IDE. This stack should be studied to understand how the scrolling works. The bitmaps for the buttons were done with the revlet at <http://berndniggemann.on-rev.com/iphonebtnsrevlet/> by Bernd Niggemann.

## Conclusion

This stack illustrates that RRPS that works on the iOS mobile platform without extra coding, but changes (outside the scope of this manual) are required to give your stack the appropriate look and behaviour for the platform.

# Searches and Queries

RRPS uses a simple syntax to specify queries. A query is a text string that specifies what the words that are required to make a match, and optionally the words that are not required in the results and cards with those words should be omitted from the results.

Queries can use a combination of *and*, *or* and *not* logical operators to enable different types of searches to be done. The examples in this chapter use assume searches done with the Demo-Search-Advanced in the Demos folder, and so use a film based set of words.

## And

The *and* operator does not need to be explicitly put in a query. For example, this query:

```
world colour
```

lists all the cards that include the words “world” *and* “colour”. In other words, in the context of the information in the Demo-Search-Advanced stack, this lists films with world in the title and are in colour. RRPS allows quotes to to be more specific, to make sure the films were made with a colour process, (and not just have “colour” somewhere else in the field the search could be changed to:

```
world "process: colour"
```

This illustrates the use of quotes, which means the text inside the quotes must match exactly. So this query lists all the cards that include the word “world” *and* the phrase “process: colour”. Queries can have as many words to search on as you need, here is one more example following the theme of the above.

```
world "process: colour" fantasy
```

This query lists all the cards that include the words “world” and “fantasy” *and* the phrase “process: colour”. Except for words within quotes, the order of the search words is unimportant, the words can be matched anywhere within a card, even in different fields on the card.

## Or

The *or* operator must be explicitly put in a query. For example, this query:

```
star or war
```

lists all the cards that include the word “star” *or* “war” (or both). Again in the Demo-Search-Advanced stack, this will have matches in the titles of the films that have either, or both words. This example includes the film *Star Wars* in the results list because RRPS handles simple plurals in many cases. If the query was changed to:

```
"star" or "war"
```

The quotes force exact matching and so the results exclude for example *Star Wars*, *Battle Beyond the Stars* and some other results that no longer match exactly.

## Not

The *not* operator excludes from the results cards that have the word immediately following the not. For example:

```
not outer space
```

lists cards *without* the word “outer”, but do have “space”. This query illustrates how the not only affects the word that immediately follows it. To compare you can try the query:

```
space
```

to see the complete list of films that have “space” in the title. Or try:

```
outer space
```

to list the films with both words.

## More Examples

By experimenting with the queries you can see how RRPS works. Here are some examples with a description of the films shown from the Demo-Search-Advanced stack, with the query shown in bold on the next line

- films with monster in the title:  
**monster**
- films with space and monster in the title:  
**space monster**
- films with space or monster in the title:  
**space or monster**
- films with monster and not space in the title:  
**monster not space**
- films with exactly monster in the title:  
**"monster"**
- monster films with a score of 6, but no Godzilla, please:  
**monster "score: 6" not godzilla**

# Building an Index

The `rrpSearchBuildIndex` command of RRPS reads the text in the text fields of all of the cards to be indexed. This index is used by the `rrpSearchQuery` command when searching for cards that match a query. The index itself is a property of the `rrpSearch` stack.

This means that there can only be one index for each copy of RRPS. If you have an application that does require more than one index, for different types of searches, this can be done by making copies of the `rrpSearch` stack with different names. For example, you could copy the stack three times calling each one `rrpSearchA`, `rrpSearchB` and `rrpSearchC` respectively. By making each of these a substack you can have three indexes in the one application.

## Word Delimiters

RRPS parses each text field using a custom set of delimiters to determine the beginning and end of each word. The `rrpSearchWordDelimiters` command returns the list of delimiters used by RRPS. By default these are:

`;;,!?()[]{}<>@#%&^&*\|^`=+` as well as tab, space, quote, cr and lf`

You can change the word delimiters with the `rrpSearchSetWordDelimiters` command. This command accepts a single parameter which is a line of characters. Each character being a delimiter.

## Stop words

RRPS uses stop words to reduce the size of the index. Words in the stop word list are not indexed. The `rrpSearchStopWords` command returns the stop words as comma delimited items. These stop words are commonly used English words that often do not add to the quality of the results returned from a search. By default these are:

`a,all,am,an,and,any,are,as,at,be,but,can,did,do,does,for,from,had,has,have,here,how,i,if,in,is,it,no,not,of,on,or,so,that,the,then,there,this,to,too,up,use,what,when,where,who,why,you`

You can change the stop words with the `rrpSearchStopWords` or `rrpSearchAppendStopsWords` commands that accepts a comma delimited line of words. There should be no extra spaces between the items. The first command replaces the list or stop words, while the second adds words to the existing list.

- ☛ While stop words can significantly reduce the size of the index, when a large amount of text is being indexed they reduce the ability to search for specific phrases. For example, because “from” is a stop word you cannot search for it specifically. For some applications you may want to change the stop words, or set them to empty.

The `rrpSearchBuildIndex` command returns the list of words in the index with the frequency of each word. The list is available with the result function immediately after the call to `rrpSearchBuildIndex`. The list has two items on each line, delimited by a tab. The first item on each line is the word, and the second item is the number of times the word appears in the index.

## What is indexed?

These commands allow you to set the cards that are included in the index.

```
rrpSearchIndexThisStack
rrpSearchIndexStack
rrpSearchIgnoreThisCard
rrpSearchIgnoreCard
```

One or more of these must be called *after* `rrpSearchInitialize` and *before* `rrpSearchBuildIndex`, so RRPS has a list of cards to index. Once the cards to search are set, the contents of every text field on the cards are parsed to build the index. Refer to the next heading for making exceptions to this.

- ☛ In a large stack, or on a slow computer, this can take a while so be prepared to wait in such a case. Building the index may more than 10 minutes in some instances.

## Ignoring specific fields

Every text field on the cards being indexed are included in the index. Sometimes there may be fields that you do not want indexed. For example, this may be the case with fields that are part of a background group that appear on every card.

- To stop a field from being included in the index add a custom property named `rrpDontSearch` to the field and set its value to true.

# Deploying Your Own Application

To deploy or distribute your own application with RRPS you must first purchase a License Key. This can be done from the RunRevPlanet.com website, or from other approved vendors. You can visit the RunRevPlanet website for full details, but in summary, a License Key can be purchased with a credit card and the key is emailed to you.

## Setting the license name and key

If you have purchased a License Key direct from RunRevPlanet, for RRPS to function fully in a standalone application, two extra lines must be added your scripts. These lines are normally where you initialize your application and may be in the openStack handler. Below is a sample of these two lines.

```
set the cLicenseName of stack "rrpSearch" to "Scott McDonald"  
set the cLicenseKey of stack "rrpSearch" to "XXX-XXX-XXX-XXX-XXX-XXX"
```

Here the two licensing properties are set. The first, cLicenseName is the name that is registered when you purchase the License Key. The cLicenseKey property is a special character code that is unique and emailed to you after the purchase is completed.

With these two lines in your applications that have a grid, you can make use of RRPS without any further payments or royalties.

- ☛ The License Key is for your use only and *must not* be made public or shared with others. This means that the script containing the key must be encrypted with a password. This requires setting the password property of the stack that sets these properties.

Setting these properties needs to be done only in the initialization of your application.

- ☛ When setting the cLicenceKey or cUnlockCode properties the code must be entered exactly as sent to you. For example, the hyphens are significant and must be included.

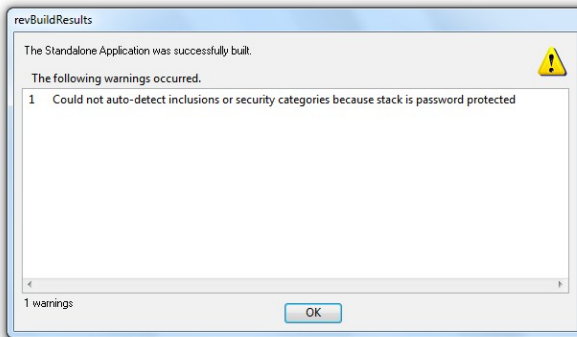
## Acknowledgement

While not required, acknowledgement of the use of the “RunRevPlanet Search” in the Readme file or the About box of your application is welcome.

## Search as a Substack

The RRPS stack should be distributed with your standalone application as a substack of your application. As a substack it becomes part of your mainstack file, but when making the standalone application you may get an error message.

When choosing the Save as Standalone Application command in the File menu the message shown below may appear. Depending on the version of LiveCode you are using, the wording or appearance of this warning may vary.



- ☛ To prevent this message in the General section of the Standalone Application Settings in the File menu, in the Advanced section the Select inclusions for the standalone application option must be selected. The stacks or script libraries required by your application must be selected manually.

# Error Messages

The handlers in RRPS can fail and report an error messages if the names of cards and stacks refer to objects that do not exist. The can occur when:

- Calling handlers to set the cards or stacks to index.
- Building the index.
- Executing a query

In each of these cases the most likely reason for the error is that an object name has been misspelt, or the name of the object has been changed after the index has been built.

In the first case, the solution is to check the parameters passed to these handlers:

```
rrpSearchIndexStack  
rrpSearchIndexCard  
rrpSearchIgnoreCard
```

In the second case, if object names have been changed then the solution is to call:

```
rrpSearchBuildIndex
```

to rebuild the index and update the names of the stacks and cards in the RRPS internal data structures. In general, once an index has been correctly built then RRPS should not generate errors at runtime, but if there is an error a warning dialogue is shown.

This warning can be suppressed with the `rrpSearchShowErrors` command with a parameter of `false`. Doing this is only recommended after you have fully tested and debugged your application. If this is done you can still check for errors with the `rrpSearchLastError` command.

# Handler Reference

This chapter documents all the public handlers in the rrpSearch stack. Each entry begins with the name of the handler and on the right whether it is a command or function. Next is the declaration of the handler as found in the rrpSearch.rev stack.

This is followed by a short description of the action of the handler, and if there are any parameters a list of them. Lastly, general comments about the handler are included.

- ☛ Some of these handlers are indicated as being functions, but all are declared as commands. When the first line of the entry indicates a function, this means the command returns a value with the result function.

---

## **cLicenseKey**

**property**

`cLicenseKey`

Set this property with a valid key to use RRPS in a standalone application.

Default: empty

### **Comment**

The `cLicenseKey` property must be set to a valid key, otherwise the `rrpSearchQuery` command will not work in a standalone application and an error message is shown. This property must be a key that is associated with the `cLicenseName` property. Refer to the *Deploying Your Own Application* chapter for more details.

---

**cLicenseName****property**`cLicenseName`

Set this property with the name used when the License Key was purchased to use RRPS in a standalone application.

Default: empty

**Comment**

The `cLicenseName` property must be set to the name associated with the key in the `cLicenseKey` property, otherwise the `rrpSearchQuery` command will not work in a standalone application and an error message is shown.

---

**cVersionNumber****property**`cVersionNumber`

Read this property to find out the version number of `rrpSearch`

Value: 1.0

**Comment**

The `cVersionNumber` property allows you to check the version of your copy of `rrpSearch`. This value is needed when requesting technical support.

---

**rrpSearchAppendStopWords****command**`command rrpSearchAppendStopWords pList`

Appends additional stop words to the words already in the stop word list.

**Parameters**

`pList`: a single line of text with the stop words delimited by commas with no extra spaces

**Comment**

This command adds words to the word stop list. If used, this command must be called before `rrpSearchBuildIndex`. To examine the stop words currently set in `rrpSearch` use the `rrpSearchStopWords` command. Refer to the *Building an Index* chapter for more details.

**command** `rrpSearchBuildIndex`

Builds the index that `rrpSearch` uses to find the results of a query.

**Returns**

the list of words in the index with the frequency of each word

**Comment**

This command is used when making your stack, and is not normally called by the user. This command needs to be called if you change the content of the cards or change the settings of the search, such as the stop words or delimiters. The list returned in the result function can be ignored, but is useful in analysing the index to determine suitable candidates for additional stop words. Refer to the *Building an Index* chapter for more details.

- ☛ This command puts text into the Message Box to indicate the progress of the building of the index.

**command** `rrpSearchGoLinkCard pLink`

Shows the card specified by `pLink`.

**Parameters**

`pLink`: the link parameter from the `linkClicked` handler

**Comment**

This command assumes the HTML link is in the format returned by the `rrpSearchResultLinks` command. This command would normally be called inside a `linkClicked` handler, where `pLink` is the same parameter passed to that handler.

---

**rrpSearchIgnoreCard****command****command** rrpSearchIgnoreCard pCardName,pStackName

Adds the specified card in the stack to a list of cards to ignore when the index is built.

**Parameters**

pCardName: the short name of the card

pStackName: the short name of the stack that contains the card

**Comment**

This command is called after rrpSearchIndexStack or rrpSearchIndexThisStack to exclude a specific card from the index. The two parameters are the short name for the card and stack respectively. If used, this command must be called before rrpSearchBuildIndex.

---

**rrpSearchIgnoreThisCard****command****command** rrpSearchIgnoreThisCard

Adds the current card to a list of cards to ignore when the index is built.

**Comment**

This command is called after rrpSearchIndexStack or rrpSearchIndexThisStack to exclude the current card from the list of cards to index. If used, this command must be called before rrpSearchBuildIndex.

---

**rrpSearchIndexCard****command****command** rrpSearchIndexCard pCardName,pStackName

Adds the specified card in the stack to a list of cards to index when the index is built.

**Parameters**

pCardName: the short name of the card

pStackName: the short name of the stack that contains the card

**Comment**

If only some cards in a stack need indexing with this command you can set only specific cards to be added to the list to index. The two parameters are the short name for the card and stack respectively. If used, this command must be called before rrpSearchBuildIndex.

---

**rrpSearchIndexStack****command****command** rrpSearchIndexStack pStackName

Adds all the cards in the stack pStackName to a list of cards to index when the index is built.

**Parameters**

pStackName: the short name of the stack

**Comment**

The parameter is the short name for the stack. After calling this command individual cards can be removed from the list of cards to index by calling rrpSearchIgnoreThisCard or rrpSearchIgnoreCard. If used, this command must be called before rrpSearchBuildIndex.

---

**rrpSearchIndexThisStack****command****command** `rrpSearchIndexThisStack`

Adds all the cards in the current stack to a list of cards to index when the index is built.

**Comment**

After calling this command individual cards can be removed from the list of cards to index by calling `rrpSearchIgnoreThisCard` or `rrpSearchIgnoreCard`. If used, this command must be called before `rrpSearchBuildIndex`.

---

**rrpSearchInitialize****command****command** `rrpSearchInitialize`

Initialises `rrpSearch` by setting all internal properties to default values and clears the index.

**Comment**

This command must be called once before calling `rrpSearchBuildIndex` to initialise `rrpSearch`. After calling this command you can call other commands such as `rrpSearchIndexStack` and `rrpSearchSetSopWords` to control how the index is built.

- ☛ Once the index is built this command should not be called again, unless you need to rebuild the index.

---

**rrpSearchLastError****function****command** `rrpSearchLastError`**Returns**

the last error message, if a runtime error has occurred in a `rrpSearch` handler

**Comment**

This command returns empty if there have been no errors since the last call to `rrpSearchLastError`. Calling `rrpSearchLastError` clears the error message.

---

**rrpSearchQuery****function****command** `rrpSearchQuery pQuery,pRawResults`**Returns**

a list of the cards that match pQuery. Each line in the list has two tab delimited items. The first item is the matching card and stack, and the second item is the number of matches on that card.

**Parameters**

pQuery: a string that specifies the words to be matched during a search  
pRawResults: empty, true or false

**Comment**

This command does the actual searching of the index to return the results of a query. This is the main command in rrpSearch. The format of pQuery is detailed in the *Searches and Queries* chapter. If pRawResults is empty or false, this command does extra processing to prepare the results as a HTML string that includes a link to the card and the context of each matching word. If your application does not call rrpSearchResultLinks, you can speed up rrpSearchQuery by setting pRawResults to true.

---

**rrpSearchQueryTime****function****command** `rrpSearchQueryTime`**Returns**

the number of milliseconds spent in the previous call to rrpSearchQuery

**Comment**

This command can be used when presenting the query results in your application.

---

**rrpSearchResultLinks****function****command** `rrpSearchResultLinks`**Returns**

the results of the previous search as a HTML string that includes a link to each matching card and the context of each match.

**Comment**

This command returns its value with the result function which should be called after this command. The contextual results are intended for use in the HTMLText

property of a field.

---

**rrpSearchResults****function**

**command** `rrpSearchResults`

**Returns**

the previous list of results returned by `rrpSearchQuery`

**Comment**

This command returns the same results as the last call of `rrpSearchQuery`.

---

**rrrpSearchResultSummary****function**

**command** `rrrpSearchResultSummary`

**Returns**

a string with the number of matching cards found, and optionally the time to process the query

**Comment**

This command is often combined with `rrpSearchResultLinks` to present the query results to the user. Call the `rrpSearchSetSummaryShowsTime` command with a parameter of `true` to include the time in the summary.

---

**rrpSearchStopWords****function**

**command** `rrpSearchStopWords`

**Returns**

the list of stop words as a comma delimited string

**Comment**

This command is needed to examine and possibly change the list of stop words used by `rrpSearch`. Stop words are words not included in the index. Normally, these are common words that can be ignored when searching because they do not add to the accuracy of a query. Refer to the *Building an Index* chapter for more details.

---

**rrpSearchSetIndexNumbers****command****command** `rrpSearchSetIndexNumbers pEnable`

Sets whether numbers are included in the index.

**Parameters**

pEnable: true or false

**Comment**

By default `rrpSearch` does not index numerical values. Calling this command with pEnable as true, before calling `rrpSearchBuildIndex` includes whole (integer) numbers in the index.

---

**rrpSearchSetLinkShowsContext****command****command** `rrpSearchSetLinkShowsContext pEnable`

Sets whether `rrpSearchResultLinks` includes the context of each matching word

**Parameters**

pEnable: true or false

**Comment**

By default `rrpSearchResultLinks` does show the context of each matching word. Calling this command with pEnable set to false only the links are included in the results. This makes the query execute faster.

---

**rrpSearchSetLinkShowsStack****command****command** `rrpSearchSetLinkShowsStack pEnable`

Sets whether `rrpSearchResultLinks` includes the stack name in the anchor text of the link.

**Parameters**

pEnable: true or false

**Comment**

By default `rrpSearchResultLinks` does not show the stack name in the anchor text of each link. Calling this command with pEnable set to true includes the stack name in the the links. This can be useful if more than one stack has been indexed.

---

**rrpSearchSetShowErrors****command****command** rrpSearchSetShowErrors pEnable

Sets whether commands that can fail at runtime show a dialogue box, or fail silently.

**Parameters**

pEnable: true or false

**Comment**

By default rrpSearch shows a dialogue if there is an error during execution. Set this to false to hide the warning. If this is done, the rrpSearchLastError can be called to check for an error. Refer to the *Error Messages* chapter for details. The most likely cause of a runtime error is an incorrect stack or card name, or the index has not been rebuilt after changing the names of the indexed objects.

---

**rrpSearchSetStopWords****command****command** rrpSearchSetStopWords pList

Sets the stop words which are words that are not included in the index.

**Parameters**

pList: a single line of text with the stop words delimited by commas with no extra spaces

**Comment**

If used, this command must be called before rrpSearchBuildIndex. To examine the stop words currently set in rrpSearch use the rrpSearchStopWords command. Refer to the *Building an Index* chapter for more details.

---

**rrpSearchSetSummaryShowsTime****command****command** rrpSearchSetSummaryShowsTime pEnable

Sets whether rrpSearchResultSummary includes the time taken for the query to execute.

**Parameters**

pEnable: true or false

**Comment**

By default rrpSearchResultSummary does not show the time taken for the previous query. Calling this command with pEnable set to true includes the number of seconds in the summary.

---

**rrpSearchSetWordDelimiters****command****command** rrpSearchSetWordDelimiters pList

Sets the characters that terminate each word.

**Parameters**

pList: a single line of text with the characters used to separate words in rrpSearch

**Comment**

The pList parameter is a string without any delimiters. If used, this command must be called before rrpSearchBuildIndex. To examine the word delimiters currently set in rrpSearch use the rrpSearchWordDelimiters command. Setting the word delimiters gives you control over what is considered a word in rrpSearch. Refer to the *Building an Index* chapter for more details.

**command** rrpSearchWordDelimiters

**Returns**

the characters that rrpSearch uses to delimit each word in the index

**Comment**

The word delimiters are the characters that indicate the end of each word. The contents of the text fields are parsed one character at a time when building the index using the word delimiters from this command as the terminating characters for each word.

