Game Maker 4.1 (and above) Database Support vers 2.1

by Alasdair Forsythe (alasdairdf)

AdF Software
www.adfsoftware.vbhq.com
adfsoftware@vbhq.com

Introduction

Changes since last version

Scripts

Advanced searches

Special variables

Database format

Introduction:

A database is a group of data stored in a structured way. In programming it can be useful for looking up information quickly from a large source without having to go through a massive script of complicated 'if' statements. Also in this version you can fully customise your database files through advanced and easy to use scripts that enable you to change any information in your database.

Some examples of how this could be useful are:

Your own score table - you could have it store the name, level, score, difficulty, etc. and even let the players choose how many spaces there are.

AI lookups - you could have all your AI (artificial intelligence) in a database and get it to go to a certain cell that matches the criteria (criteria in table/row/column names), in the cell could be the command.

Specs - if you have [for instance] a lot of weapons with different specs then you could put them all into a database and make it look up the weapon that the player is using when you need to.

Level editors - you could make the level editor create databases, all the custom levels could be stored in one file and you'd beable to make it store what object to create, where, and what variables to declare/change to it.

Remember you can execute a string by using the execute_string(string) function (very useful).

Changes since last version:

Since version 2.0:

Some major bugs have been fixed, especially in the ins_table_direct script.
You can now give the number columns and rows in the ins_table_direct script.
A database_create script has been added.

Since version 1.0:

It's all been totally redone, the old tables are not compatible.
Several tables can now be stored in one file.
Encryption is supported.
The searches can be more advanced.
You can now lookup table/column/row names.
You can now change cells (you can change anything you can lookup).
You can now add additional tables/columns/rows from code.
You can pull any table out of the database file and store them in a 2D array for faster access.
And many more new functions are also included (including loading the array back into the file).

Scripts

Several scripts have been produced to enable you to edit the database file in almost any way, you only need to add the ones you're going to use. The encryption script won't be used by you for editing the databases but it's needed if the database file is going to be encrypted.
Even if you don't use encryption it may be required to create a blank script named *encryption* just so it knows it exists, to keep the computer happy.
Adv means it uses the advanced search (see advanced searches section).

lookup_direct:
format: lookup_direct(file_name,z,x,y,change_to)
field types: lookup_direct(string,adv,adv,adv,string)
The lookup_direct script enables you to look up any cell from any table in a database file.
z is the table, x is the column, y is the row.
Putting 0 for the x/y will return that row/column name.
Putting 0 for both the x and y will return the table name.
Don't include the last argument if you just want it to return the result of the cell, otherwise it'll change it to that.
The scripts are case sensitive.
If you wish to do something like add 1 to a cell you could put something like this in the last argument:
lookup_direct(file_name,z,x,y)+1
you can do very advanced commands with this.

encryption:
format: encryption(string,encrypt?)
field types: encryption(string,true/false)
True/1 will encrypt the string.
False/0 will decrypt it.
You could use this script for encryption on anything else you want encrypted as well.
It doesn't encrypt all the characters, it misses some out at certain intervals which makes it quicker and more confusing to manually decrypt.
It would have been better to actually use a table to check which characters changed into which but then you'd need to have this table and it couldn't have been encrypted.
If you wish to make your own encryption then make a decrypted database file, change the characters around in the script and encrypt it using the convert_direct script.

insert_direct:
format: insert_direct(file_name,z,pos,row/col,make/del)
field types: insert_direct(string,adv,real,true/false,true/false)
This script creates/deletes columns/rows.
True means row/create, false mean column/delete.
Pos is the position in which the column/row will be created/deleted.
The cells inserted are all blank so you'll need to use the lookup_direct script to change them.

ins_table_direct:
format: ins_table_direct(file_name,pos(z),make/del,cols,rows)
field types: ins_table_direct(string,real,true/false,real,real)
This inserts a blank cols*rows table, you'll need to edit the cells.

convert_direct:
format: convert_direct(file_name,encrypt?)
field types: convert_direct(string,true/false)
This script converts the whole database file between cryptions.
True encrypts the whole database, false decrypts it.
It may be useful to decrypt the whole file at the start of the game and encrypt it again at the end, then when you come to use the other scripts they'll run much faster as they won't have to decrypt each line they want to read on the spot.

array_table:
format: array_table(file_name,z,var,write)
field names: array_table(string,adv,string,true/false)
This script either puts a table into a 2D array or puts it back into a database.
var is the variable that it'll be put into/taken out of.
The variable it's put into with x and y and the end contain the total x and y, eg. if var was 'bob' then 'bobx' would contain the total cols and 'boby' would contain the total rows.
True means write it to the table, false means pull it out of the table.

array_search:
format: array_search(array_name,x,y,change_to)
field names: array_search(stirng,adv,adv,string)
This script is similar to lookup_direct except it's used for arrays.
There's no point using this script if you're only putting numbers for both x and y, eg. if you want x3, y5 in the variable 'bob' it's located at bob[3,5].

insert_array:
format: insert_array(array_name,pos,row/col,make/del)
field types: insert_array(string,real,true/false,true/false)
This script is similar to insert_direct except it's used for arrays.

database_create:
format: database_create(name/path,tables,cols,rows,encrypted?)
field types: database_create(string,real,real,real,true/false)
This script creates (or wipes) a blank database file with a certain number of tables, each with a certain number of columns and rows.

Advanced searches:

Many of the scripts support advanced searches, meaning you can use various different methods for finding the result you're looking for.

The first is a simple number it jumps to that table along, it's the quickest method. It doesn't matter wheather you put a real or a string for this as long as it just contains numbers. If you don't want it to jump and you just want to to compare each tab/col/rows names against it as a number you can trick the script by using something like "string('numb')" where numb is the number. As that's got letters in it'll think it's a string and compare it to the names of the tab/col/row but it'll be worked out when it comes to checking against it and it should all work smoothly.

The second method is to enter the strings 'true' or 'false', that'll make the script test the tab/col/row name as a command and see if they're true/false. It'll jump to the first one that works. Eg. you could call the col names

things like x>200 or image_single==2.

The third method is to put a variable, the variable name must be inside a string, eg. if you want to use what's inside the variable jeff then enter 'jeff' instead of just putting jeff. You'll see why you need to do this later on. This method checks the name of the tab/col/row as a string against what's in this variable as a string. It'll jump to the first one that works.

The forth method is similar to the third but it tests the name of the tab/col/row as a variable or action against what you've put. To get it to do this you just need '{tst}' at the start of the string. Eg. You could call the tab/col/row name jeff and enter '{tst}bob' and it'll test if what's in variable jeff is equal to variable bob, while in method 3 it'd test wheather variable bob is equal to the string 'jeff'.

The fifth method is similar to 3 & 4 but using a plain string instead of a variable. It can be used with and without '{tst}'. The format needs to be (if it were jeff) '"jeff"', it's got both speech and quote marks as it needs to be a string inside of a string, the first set are removed when the script is begun and the second set is needed so it's not checked against as a variable later on. If you do use {tst} it should look something like '{tst}"jeff"'. You need a set of ' inside a set of " (or the other way around).

Refer to the special variables area to see even more advanced and complecated searches.

Special variables:

After you use most of the scripts there will be certain variables that have been defined for the script's use that you may find useful.

encrypttq - Wheather the script is encrypted or not (true/false)
tottablesq - How many tables are in the script
maxxq - The number of columns in the table you were looking at
maxyq - The number of rows in the table you were looking at
pathz - The z position from where the result was found
pathx - The x position from where the result was found
pathy - The y position from where the result was found

For use in the arguments:
tempb is the variable that has the name of the tab/col/row/result stored in it you can use it in your searches. You can also use it to generate variables in the searches, eg. 'tempb+"x"' while the tab/col/row name is bob means it'd actually check against the variable bobx. That's one of the main reason why the searches are always required in a string.

Database format:

There's now a database creator program to create database files but if you wish to do it manually in notepad the format looks like this:

Encrypted (0/1 - 0=no, 1=yes. Everything past here will be encrypted if true)
Total number of tables
{1}z name 1{2}z names 2{3} z name 3{4} etc.
totx,toty (first table total cols and rows separated by a comma)
Column names (same format as z names)
Row names (same format as z names)
Row 1 (x's in format as z names)
Row 2 (x's in format as z names)
etc.

totx,toty (second table total cols and rows separated by a comma)
etc.

So here's an example of a working table:

```
0
3
{1}Table1{2}Second table{3}The third one{4}
2,2
{1}col1{2}col2{3}
{1}row1{2}row2{3}
{1}x1,y1{2}x2,y1{3}
{1}x1,y2{2}x2,y2{3}
1,1
{1}x>10{2}
{1}y>10{2}
{1}jeff=fred{2}
3,4
{1}jeff{2}bob{3}fred{4}
{1}bill{2}bernard{3}gertrude{4}alasdair{5}
{1}hey{2}people{3}how{4}
{1}good{2}that is{3}very{4}
{1}nice{2}of{3}you{4}
{1}to{2}say{3}so{4}
```

That would be a working uncrypted script with 3 tables. The tables are called 'Table1', 'Second table' and 'The third one'.

This is what the 3rd table actually looks like (without the cells displayed):

(0,0) The third one	(1,0) jeff	(2,0) bob	(3,0) fred
(0,1) bill	(1,1) hey	(2,1) people	(3.1) how
(0,2) bernard	(1,2) good	(2,2) that is	(3,2) very
(0,3) gertrude	(1,3) nice	(2,3) of	(3,3) you
(0,4) alasdair	(1,4) to	(2,4) say	(3,4) so

Back to top