

About TURBU

TURBU is an RPG engine and game editor currently under development by Mason Wheeler. You can download the latest version and find news about the project at www.turbu-rpg.com

This help file was created with the free trial version of [HelpScribble](#).

Console help

The Console window is a debugging feature that allows you to examine the current state of the program and run script commands. It contains [data grids](#) and a [script input line](#). The data grids can be used to view the current switches and variables, and the script input line can enter RPG Script code for the script engine to run.

This help file was created with the free trial version of [HelpScribble](#).

Script input line

The input box in the [Console window](#) allows you to enter an [RPG Script](#) command and have the script engine run it. Simply type your command in and press the "Execute Script" button or hit <ENTER>.

Your command should end with a semicolon. This isn't 100% required, as the script engine will compensate for it, but it's good to get in the habit of doing it if you plan to write scripts in RPG Script, where they **will** be required.

This help file was created with the free trial version of [HelpScribble](#).

Console Data Grids

At the top of the [console window](#) is a tabbed panel containing two grids. The first grid shows the state of all switches, which can be true (on) or false (off). The grid in the second tab shows the project's variables, which each hold an integer.

The grids do not automatically update themselves with new values any time one of the switches or variables is changed. You can update the values for either grid by clicking the "Rescan" button below that grid. Also, loading the console by pressing <SPACE> from the main window while the console window is not open will immediately rescan both grids.

The grids only display the values of the current switches and variables; they cannot be used to input new values. Use the script input line for that.

This help file was created with the free trial version of [HelpScribble](#).

RPG Script

RPG Script is the scripting language for [TURBU](#) projects. It is built upon [RemObjects's PascalScript](#) framework, which in turn is based on the Object Pascal programming language. It's designed to be powerful, yet easy to use, and its capabilities are constantly being developed and added to, just as TURBU is.

[Object Pascal basics](#)

[RPG Script reference](#)

This help file was created with the free trial version of [HelpScribble](#).

Object Pascal basics

DISCLAIMER: This is not meant to be a comprehensive tutorial on writing Object Pascal code. There are plenty of those available on the Internet. What it does do is give you a bit of information to get you up to speed on the fundamentals.

The scripting system in TURBU is based on Object Pascal. Object Pascal is an all-purpose programming language that provides a good balance between ease of use and raw power. Writing programs or scripts in Object Pascal is based on using instructions to manipulate data. The data is set up as *variables*, which are a lot like variables in algebra: a letter or word that refers to a bit of information whose value is not always the same. In algebra, variables are used in place of numbers. In programming, however, a variable can represent any kind of information: a number, a sentence, an image, a sound, and so forth. There are six basic types of variables used in TURBU's script engine:

- Integer: A whole number, with nothing after the decimal point.
- Real: Any real number, including decimals.
- Boolean: A "switch" value which can be either [true](#) or [false](#).
- String: A group of letters. This can represent a word, a sentence, a paragraph, or almost any amount of text. If you use text in your code, to assign to a string variable or send to a routine that needs a string, (there'll be more on routines later,) you have to put the text inside 'single quotes' (not "double quotes"). So: *'Hello everybody!'* is good. *"Hello everybody!"* is bad, and so is *Hello everybody!* without quotes of any kind. **NOTE:** Make sure to only use ordinary, straight quotes, not the curly "smart quotes" that word processing programs like to create.
- Array: A group of variables of the same kind that are all grouped together. Picture a row of mailboxes, with each one only big enough to hold one envelope. Every mailbox is exactly like every other mailbox except for two things: their location, and what they contain. This is how arrays work; such a group of mailboxes would be referred to by programmers as an "array of envelopes", since it's not the boxes themselves that are important, but what they hold.
- Object: An object is a group of variables that are not all of the same kind but are all grouped together. Objects can be made up of integers, booleans, strings, arrays, other objects, or other data types that are beyond the scope of this tutorial. Objects also contain their own code for managing their data members.

Referring to variables

Variables are referenced by a name. A variable name is a single word or letter.

It can also contain numbers and the underscore character, if the programmer wishes to use them, but the first character in the name must be a letter. It's best to use descriptive names that are easy to read and understand. For example, the variables that make up a Hero object have names such as "name", "level", "exp", "hp", "maxHp" and so on. How you refer to a variable depends on where the variable is located.

A variable that stands on its own (that is not part of an array or an object) is referred to simply by its name.

A variable that is part of an array is referred to by the array's name, plus the specific variable's index in square brackets. For example, Switch #21 would be *switch[21]*, or hero #5 would be *hero[5]*.

A variable that is part of an object is referred to by the object's name, plus a dot (period), plus the variable's name. So the amount of EXP that hero #5 has would be found at *hero[5].exp* and the amount of money the party currently has is *party.money*.

Manipulating variables

Now that you know how to refer to variables by name, all that's left is to understand how to do things with them. There are two basic things that can be done with a variable: It can be manipulated directly with **operators**, or passed to a **routine** as input.

Operators:

Most of the operators should be familiar to anyone who took math classes in school. **+** adds two things together. **-** subtracts one value from another. ***** multiplies two values, and **/** divides. The word **mod** is also used as an operator. It produces the remainder when one number is divided by another. And parenthesis **()** can be used to enforce a certain order of operations.

There are two other important operators: **=** and **:=**. An equals sign is used to test variables to see if one is equal to the other. It produces a boolean (true or false) value, and is commonly used in **IF..THEN** statements, such as *if switch[7] = true then*. Equals does not set a variable equal to a certain value. Let me repeat that. *The = operator DOES NOT set a variable to a certain value in Object Pascal.* That's a very easy mistake for new Object Pascal programmers to make, especially if they've used BASIC or C/C++ in the past. To set a variable to a value, you use the assignment operator, colon-equals. For example: *variable[10] := party.money;*

There are five other comparison operators, in addition to **=**. They are: **<** (less than), **<=** (less than or equal to), **>** (greater than), **>=** (greater than or equal to), and **<>** (not equal to). These all give boolean results. For example, if **x** is an integer variable whose value is currently 5: **x > 4**: true. **x > 6**: false. **x > 5**: false. **x >= 5**: true. **x < 6**: true. **x <= 4**: false. **x = 5**: true. **x = 6**: false. **x <> 6**: true.

Routines:

A routine is a block of code that does a specific thing and runs when its name is "called" by the program code. Routines often require variables as input, and then do work on those variables. For example, the *showMessage* routine requires a string variable. It takes the text in that string and displays it in the in-game message box. So, *showMessage(hero[3].name);* would open the message box and output hero #3's name in it.

There are two basic types of routines, **procedures** and **functions**. A procedure simply takes its inputs, if any, and executes its code, while a function takes its inputs, executes its code, and produces a result, which is a variable of a certain type. The *showMessage* routine mentioned above is a procedure. There's a similar routine, however, called *inputNumber*, which takes an integer as input, describing how many digits the number to be input should have, then opens a message box with a number selector in it. When the user inputs a number, the *inputNumber* routine sends this number back to whatever code called it as its return value.

Function return values are handled as if the entire function was a variable. So, to call *inputNumber* to have it produce a 3-digit number and store that number in variable #38, you would write: *variable[38] := inputNumber(3);* Likewise, a routine that belongs to an object is referred to just like a variable that belongs to an object: by the object's name, a dot, and the routine's name. So, to equip hero #1 with item #28, (assuming that item #28 is a type of weapon or armor,) you would use: *hero[1].equip(28);*

End of line semicolon

You may have noticed that the examples of actual lines of code used above all end with a semicolon ; character. This is an essential part of Object Pascal syntax. The semicolon is necessary to tell the computer where each command ends and the one after it begins. There are a few very specific exceptions which we won't get into here, but as a general rule, it's essential to end every line of code with a semicolon.

That's it... for now

This tutorial has barely scratched the surface of Object Pascal or RPG Script. It will be expanded somewhat as the scripting feature becomes more important. If you really want to learn more about the language, run a web search for "Pascal tutorial" or "Delphi tutorial". More information about the variables, objects and routines available in RPG Script and the TURBU game engine can be found in the [RPG Script Reference](#) section.

This help file was created with the free trial version of [HelpScribble](#).

Explanation of Object Pascal types

[Back to RPG Script Reference](#)

Integer types: Several different types of integers are used in Object Pascal. They differ in two factors: how many bytes are used to store the data, and whether the number is *signed* (allowed to be either negative or positive) or *unsigned* (only zero and positive numbers). Types with more bytes can represent a wider range of numbers, but take up more memory. Likewise, unsigned types can represent numbers twice as large as signed types of the same byte size, by sacrificing the ability to represent negative numbers.

Byte: Unsigned, 1 byte. Range: 0..255

Shortint: Signed, 1 byte. Range: -127..128

Word: Unsigned, 2 bytes. Range: 0..65,535

Smallint: Signed, 2 bytes. Range: -32,768..32,767

Cardinal: Unsigned, 4 bytes. Range: 0..4,294,967,295

Integer: Signed, 4 bytes. Range: -2,147,483,648..2,147,483,647

Enumerated types: A set of names that each represent a number, but which is generally restricted to a far smaller set size than integer types provide. They are used by the programmer as a mnemonic device to make certain sets of data easier to remember and refer to. For example, instead of numbering a hero's slots as 1..5, they are represented by an enumerated type using the names of the slots. Enumerated types are declared using a list of names separated by commas, within parenthesis.

String: A sequence of letters, used to represent text. A *literal string* (actual string data, not a string variable) used in a script must be enclosed in 'single quotes', so the computer knows it's looking at a string and not a variable name or enumerated type identifier. "Double quotes" and curly "smart quotes" generated by word processors will not work. Only straight 'single quotes' are accepted by the script compiler.

Array: A group of variables of the same type, arranged in an ordered sequence. In Object Pascal, arrays are declared with the word "array," then an optional range declaration in square brackets, then "of <variable type>". So an array of ten **word** variables would be defined as: *array [1..10] of word*, and an array of **integer** variables with no predetermined length would simply be: *array of integer*.

Object: A group of variables which are not all of the same type, grouped together to form a larger, composite variable. Objects are used to represent a single item with many different attributes. Objects also contain routines for managing the variables that they contain. An object's routines are known as **methods**, and its variables are usually hidden from direct access by the object's programmer. Instead, the variables are accessed through **properties**, an interface that looks like variables to the person using the object, but which may actually contain routines to process information behind the scenes when reading or changing it. For example, the Hero object's "hp" property reads directly from the variable, but when a script changes its value, that value is first checked to make sure that it's valid (not negative and not higher than the hero's maximum HP), corrects the value if necessary, then assigns it to the real variable. Then it checks if the value it assigned is 0, and if so, it sets the "dead" condition on the hero. But all this happens behind the scenes; all the RPG Script writer has to do is change the property as if it were a normal variable.

This help file was created with the free trial version of [HelpScribble](#).

Inheritance

Certain object classes have enough things in common that implementing the same properties and methods for each different one would be redundant. Instead, it's possible to declare all of the common elements in a *base class* and define the other classes as *descendant classes* (also called *subclasses* or *child classes*) of the base class. A descendant class inherits all of the elements of its parent class, plus any new properties or methods that are defined for it. In essence, a descendant class is a more specialized type of its base class. For example, the [TRpgCharacter](#) class contains a [Flash](#) method which can be used with any object that descends from TRpgCharacter.

This help file was created with the free trial version of [HelpScribble](#).

Default Property

If an object contains at least one array property, and the objects in that array are the main focus of the parent object, the programmer may set that array as the *default property* for the object. This means that the script writer can treat the parent as the array itself in scripts. For example, the *hero* array property is the default property for the [TRpgParty](#) class. This means that, for a TRpgParty object named "party," `party.hero[2]` and `party[2]` can be used interchangeably.

Related topics

[RPG Script Reference](#)

[Object Pascal basics](#)

[TRpgParty Properties](#)

This help file was created with the free trial version of [HelpScribble](#).

Array Element Zero

Due to the nature of Object Pascal, all dynamic arrays ([arrays](#) whose range is not determined before the program runs) start at element #0, not element #1. Actual objects used in-game start at #1, in keeping with the RPG Maker database arrangement. The RPG Script system reserves element 0 of each array as a "junkyard," redirecting invalid values to it. For example, if there were only seven heroes defined in the database, the line `hero[9].equip(10);` would instead equip hero[0]. This helps keep the script engine from bailing out or causing a system error.

Warning: Unlike objects defined from the database, element 0 objects are not guaranteed to be stable, and the values of their variables are not clearly defined at any given moment. Clever programmers could use element 0 for their own purposes, but they do so at their own risk. Accessing element 0 has the potential to cause unforeseen consequences, especially if it is done unintentionally. In the example given above, the [equip](#) routine would still work properly, *including removing the equipment item from the inventory*, but the item would be equipped to hero # 0, not to anyone defined in the database. This could make the item appear to have vanished if the programmer is not aware of this behavior.

[RPG Script Reference](#)

This help file was created with the free trial version of [HelpScribble](#).

Read-only Properties

Certain object [properties](#) are designated as read-only in RPG Script. This means that the script writer is allowed to examine their value, but attempting to change them will cause the RPG Script compiler to reject the script.

[RPG Script Reference](#)

This help file was created with the free trial version of [HelpScribble](#).

MIDI Lead-in Issue

Certain MIDI files, including many of the files distributed in the RPG Maker Runtime Package, begin with a certain amount of silence. RPG Maker's MIDI player skips the silence at the front of the file and begins playing at the first note immediately. Unfortunately, the [SDL_Mixer](#) library used by TURBU does not currently support this feature. This can cause timing problems. Most notably, such songs will not work properly with the [Inn routine](#). This issue has been reported to the SDL developers, and I've requested that the option to skip lead-in silence be added to SDL_Mixer. Until then, it will be necessary to work around this issue by either choosing MIDI files with no lead-in, using the [Wait routine](#) in scripts to compensate if music/action synchronization is important, or using other file formats.

This help file was created with the free trial version of [HelpScribble](#).

XYZ Transparency Issue

TURBU can import the XYZ image format used in many RPG Maker projects. However, due to technical limitations in the current graphics library, it imports slowly, and transparency is not preserved correctly. The current version of the Map Viewer uses a hack to fake proper transparency for Chipset and Charset graphics in XYZ format, but not other graphic types. Therefore, it is recommended to convert XYZ images to PNG format in RPG Maker.

This will be fixed in a later version, after the Map Viewer has been migrated to a more powerful graphics library.

This help file was created with the free trial version of [HelpScribble](#).

RPG Script reference

This section describes the variables, object types, and routines that have been written thus far for the [RPG Script](#) engine.

DISCLAIMER: RPG Script and the TURBU engine are both under development. Nothing here is guaranteed to remain the same in future releases as it is now, and several properties and routines described here have not yet been fully implemented.

Global variables:

The following variables are directly available from the scripting system:

Name	Type	Description
Switch	array of boolean	This array contains a group of boolean values which can be set to true or false in scripts throughout the game.
variable	array of integer	This array contains a group of integer values which can be set to any valid number in scripts throughout the game.
party		

[TRpgParty](#) This object refers to the currently active party. hero Array of [TRpgHero](#) This array contains a group of Hero objects, one for each Hero entry in the project's database. event Array of [TRpgEvent](#) This array refers to the various events on the map. thisEvent [TRpgEvent](#) Refers to the TRpgEvent object whose script is currently running. timer [TRpgTimer](#) This object refers to the in-game timer. menuEnabled boolean Flag controlling whether or not the user can access the system menu. image Array of [TRpgImage](#) This array contains a group of Image objects. Its length is fixed at 20 slots

Object types:

The following classes (types of objects) are used in the scripting system. Following the established Object Pascal conventions, all object types begin with a T. Objects contain variables, known as "properties," and routines, known as "methods." The page for each object below will show a list of all the properties and methods that each type of object in the RPG Script scripting system contains.

[TRpgHero](#): Object representing a hero.

[TRpgParty](#): Object representing the active party.

[TRpgEvent](#): Object representing an event on the map.

[TRpgTimer](#): Object representing the in-game timer.

[TRpgInventory](#): Object representing the party's inventory.

[TRpgVehicle](#): Object representing an in-game vehicle.

[TRpgImage](#): Object representing an on-screen image.

Global routines:

The following routines are available directly from the scripting system, without

being members of any class.

random: Generates a random number.
showMessage: Displays an in-game message box
messageOptions: Sets various options for the in-game message box.
setPortrait: Sets the current portrait (face graphic) to be displayed in the message box.
clearPortrait: Sets the message box to not display a portrait.
showChoice: Shows a choice in the message box.
inputNumber: Uses the message box to input a number from the user.
heldItems: Checks to see how many of a certain item the party has.
heroJoin: Adds a hero to the party.
heroLeave: Removes a hero from the party.
addItem: Adds items to the inventory.
removeItem: Removes items from the inventory.
addExp: Adds experience to heroes.
removeExp: Removes experience from heroes.
addLevels: Adds levels to heroes.
removeLevels: Removes levels from heroes.
setSystemMusic: Changes the system background music for various events.
setSystemSound: Changes the preset system SFX for various events.
setSkin: Changes the current system "skin" graphics set.
battle: Begins a battle. (Not yet implemented)
prepareStore: Sets up a store inventory for the shopping system to use.
shop: Opens the shopping system.
inn: Calls the inn. (Partially implemented)
buttonStart: Checks to see if the current event script was started by pressing the action button.
openMenu: Opens the system menu.
inputText: Allows the user to enter a text string. Commonly used for entering a custom hero name.
rideVehicle: Causes the party to board/leave a vehicle.
teleport: Teleports the current party to another location. (Partially implemented.)
memorizeLocation: Stores the current party's location in three variables. (Slightly bugged.)
teleportVehicle: Teleports a specified vehicle to another location.
teleportEvent: Teleports a specified event to another location on the map.
swapEvents: Exchanges the position of two events on the map.
getTerrainID: Returns the ID # of the map terrain at a certain square.
getEventID: Returns the ID # of the event at a certain square.
eraseScreen: Clears the screen, optionally using a system transition.
showScreen: Restores the screen after an erase, optionally using a system transition.
setTransition: Changes the default transition settings.
setScreenTone: Changes the overall color of the screen. (Partially implemented.)
flashScreen: Flashes a certain color over the screen.
lockScreen: Locks the current screen position.
unlockScreen: Unlocks the current screen position.
panScreen: Pans the screen from the current camera position.
panScreenTo: Pans the screen to a certain position.
returnScreen: Causes the screen to return to the current party.
setWeather: Sets the current weather effect.
increaseWeather: Increases the severity of the current weather effect.
decreaseWeather: Decreases the severity of the current weather effect.
newImage: Sets up a new on-screen image.
showBattleAnim: Displays a battle animation on-screen.
playMusic: Changes the background music.

fadeOutMusic: Causes the background music to gradually fade out.
memorizeBgm: Causes the system to remember the current background music.
playMemorizedBgm: Causes the system to play previously memorized background music.
playSound: Plays a sound effect.
prepareRoute: Sets up a new move script for characters to use.
waitUntilMoved: Pauses the script until all characters have finished their move scripts.
stopMoveScripts: Cancels all active move scripts.
wait: Pauses the script execution.
keyScan: Checks to see if any commands are being entered from the keyboard.
setBGImage: Changes the current background image.
deleteCharacter: Removes the character the current script is attached to from the map entirely.
callGlobalEvent: Causes a global event to run.
callEvent: Causes an event from the current map to run.

This help file was created with the free trial version of [HelpScribble](#).

TRpgHero Class

TRpgHero is a class that represents a hero. One TRpgHero object is automatically created per Hero entry in the database.

[Back to RPG Script Reference](#)
[Explanation of Object Pascal types](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgHero Properties

[Back to TRpgHero](#)

[Back to RPG Script Reference](#)

[Explanation of Object Pascal types](#)

Name	Type	Description
name	string	The hero's name
charClass	string	The hero's class (listed as "degree" in RPG Maker)
level	byte	The hero's current level
exp	integer	The hero's total EXP
hp	integer	The hero's current HP
mp	integer	The hero's current MP
maxHp	integer	The hero's maximum HP
maxMp	integer	The hero's maximum MP
stat	array[1..4] of smallint	Array containing the hero's primary statistics. They can be accessed through this array, or individually by name: attack, defense, mind, agility
attack	smallint	The hero's total attackpower. Corresponds to stat[1]
defense	smallint	The hero's total defensive power. Corresponds to stat[2]
mind	smallint	The hero's mental strength. Affects magical attack and defense. Corresponds to stat[3]
agility	smallint	The hero's quickness and dexterity. Affects the chance to hit and to dodge attacks. Corresponds to stat[4]
equipment	array[1..5] of word	Array of numbers referring to the ID of the items the hero currently has equipped. A 0 means an unequipped slot. Slots 1-5 correspond to: weapon, shield or second weapon, armor, helmet, miscellaneous item.
skill	array of boolean	Array of boolean values, one for each skill defined in the database, signifying whether the hero knows that skill or not.
condition	array of boolean	Array of boolean values, one for each condition defined in the database, signifying whether the hero is afflicted by that condition or not.

This help file was created with the free trial version of [HelpScribble](#).

TRpgHero Methods

[Back to TRpgHero](#)

[Back to RPG Script Reference](#)

[equip](#): Sets a hero's equipment.

[unequip](#): Unequips whatever item is in a certain slot.

[fullheal](#): Fully restores the hero.

[takeDamage](#): Damages the hero. Used to calculate damage taken in combat.

[setSprite](#): Changes the hero's current sprite (character graphics set).

[setPortrait](#): Changes the hero's current portrait (face graphic).

[inParty](#): Checks to see if the hero is in the current party.

[equipped](#): Checks to see if the hero has a certain item equipped.

This help file was created with the free trial version of [HelpScribble](#).

TRpgHero.equip

Class

[TRpgHero](#)

Syntax

```
procedure equip(id: word);
```

Description

The equip method attempts to equip a hero with an item of the type represented by **id** in the database. If that item does not exist, is not a piece of equipment, or can't be equipped by the current hero, nothing happens. Otherwise, the hero's currently equipped item for that slot (if any) is [unequipped](#), one of that item is removed from the inventory if it is present, or created if it is not found in the inventory, and equipped on the hero.

Related topics

[TRpgHero.unequip](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgHero.unequip

Class

[TRpgHero](#)

Syntax

```
procedure unequip(id: TSlotList);
```

Description

The unequip method removes the item that the hero has equipped in the slot specified by **id**, if any, and returns it to the inventory. Using the "[all](#)" value causes all currently equipped items to be unequipped.

Related topics

[THero.equip](#)

[TSlotList](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgHero.fullheal

Class

[TRpgHero](#)

Syntax

```
procedure fullheal;
```

Description

The fullheal method restores a hero's HP and MP to their maximum values and removes all conditions.

This help file was created with the free trial version of [HelpScribble](#).

TRpgHero.takeDamage

Class

[TRpgHero](#)

Syntax

```
function takeDamage(power: word; defense, mDefense, variance: byte): word;
```

Description

The takeDamage method deals damage to a hero based on **power** and **variance**. **Defense** and **mDefense** signify how much the hero is capable of using his *defense* and *mind* statistics to defend against this attack. The return value is how much damage the hero actually took.

This method is not yet fully implemented. It currently simply subtracts a number of hitpoints equal to **power** from the hero's *hp* property.

Related topics

[TRpgHero Properties](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgHero.setSprite

Class

[TRpgHero](#)

Syntax

```
procedure setSprite(filename: string; index: byte; translucent: boolean);
```

Description

This method changes the hero's current sprite (graphics set). It first checks to see if **index** falls within the acceptable range (1..8) and if a CharSet file is available with the name given by **filename**. If either test fails, nothing happens. Otherwise, it sets the hero's sprite to the appropriate sprite from that charset.

This help file was created with the free trial version of [HelpScribble](#).

TRpgHero.setPortrait

Class

[TRpgHero](#)

Syntax

```
procedure setPortrait( filename: string; index: byte);
```

Description

This method changes the hero's current portrait (face graphic). It first checks to see if **index** falls within the acceptable range (1..16) and if a FaceSet file is available with the name given by **filename**. If either test fails, nothing happens. Otherwise, it sets the hero's portrait to the appropriate portrait from that faceset.

This help file was created with the free trial version of [HelpScribble](#).

TRpgHero.inParty

Class

[TRpgHero](#)

Syntax

```
function inParty: boolean;
```

Description

This method returns **true** if the hero is in the current party, or **false** if he isn't.

This help file was created with the free trial version of [HelpScribble](#).

TRpgHero.equipped

Class

[TRpgHero](#)

Syntax

```
function equipped(id: word): boolean;
```

Description

This method returns **true** if the hero has item **#id** equipped, or **false** if he doesn't, or if **id** is not a valid item number from the database.

Related topics

[TRpgInventory.contains](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgCharacter Class

TRpgCharacter is a base class that represents any on-screen character or event. The [TRpgParty](#), [TRpgEvent](#) and [TRpgVehicle](#) classes inherit from TRpgCharacter.

[Back to RPG Script Reference](#)
[Explanation of Object Pascal types](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgCharacter Properties

[Back to TRpgCharacter](#)

[Back to RPG Script Reference](#)

[Explanation of Object Pascal types](#)

Name	Type	Description
screenX	integer	Determines the X coordinate of the character's position relative to the current screen position.
screenY	integer	Determines the Y coordinate of the character's position relative to the current screen position.
translucency	byte	Value from 0 (fully opaque) to 7 (almost completely translucent)

This help file was created with the free trial version of [HelpScribble](#).

TRpgCharacter Methods

[Back to TRpgImage](#)

[Back to RPG Script Reference](#)

flash: Causes the character to flash a certain color.

move: Causes the character to move onscreen.

This help file was created with the free trial version of [HelpScribble](#).

TRpgCharacter.flash

Class

[TRpgCharacter](#)

Syntax

procedure flash(r, g, b, power: byte; time: cardinal; wait: boolean);

Description

This method causes the character to flash a certain color. The **r**, **g**, and **b** parameters determine the color, and **power** is the opacity of the flash. These parameters accept any value from 0 (none) to 255 (full). The flash appears instantly, then gradually fades out over a period of **time** milliseconds.

Related topics

[FlashScreen routine](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgCharacter.move

Class

[TRpgCharacter](#)

Syntax

```
procedure move(frequency: byte; skip: boolean; route: word);
```

Description

This method causes the character to move on-screen, or perform other actions according to the movement script designated by the **route** parameter. The **frequency** parameter can be any number from 1 (long delay between steps) to 8 (no delay between steps). If **true** is passed to the **skip** parameter, the movement script will skip any moves that are blocked or invalid and continue to the next step. If **skip** is **false**, the character will wait until it is able to perform the movement before continuing.

Related topics

[waitUntilMoved routine](#)

[stopMoveScripts routine](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgParty Class

TRpgParty is a class that represents the active party. One TRpgParty object named *party* is automatically created during initialization. Descends from [TRpgCharacter](#).

[Back to RPG Script Reference](#)
[Explanation of Object Pascal types](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgParty Properties

[Back to TRpgParty](#)

[Back to RPG Script Reference](#)

[Explanation of Object Pascal types](#)

Name	Type	Description
hero	array[1..4] of	

[TRpgHero](#) Array containing the slots for the heroes which make up the party. This is the default property for TRpgParty. money integer The amount of cash the party is currently carrying. inventory [TRpgInventory](#) Object containing the party's inventory. levelNotify boolean Determines whether the [addExp](#), [removeExp](#), [addLevels](#) and [removeLevels](#) routines notify the user when they change a character's level. map word The ID number of the current map. x word The x-coordinate of the party's current location on the map. y word The y-coordinate of the party's current location on the map. facing byte A number designating which direction the party is facing. The direction values are: down: 2, left: 4, right: 6, up: 8. translucent boolean Determines whether the party is drawn as partially translucent or completely opaque.

This help file was created with the free trial version of [HelpScribble](#).

TRpgParty Methods

[Back to TRpgParty](#)

[Back to RPG Script Reference](#)

[addItem](#): Adds items to the inventory.

[removeItem](#): Removes items from the inventory

[addExp](#): Awards EXP to the party members.

[removeExp](#): Removes EXP from the party members.

[addLevels](#): Increases the party members' levels.

[removeLevels](#): Decreases the party members' levels.

[takeDamage](#): Damages all members of the party.

This help file was created with the free trial version of [HelpScribble](#).

TRpgParty.addItem

Class

[TRpgParty](#)

Syntax

```
procedure addItem(const id, number: word);
```

Description

The addItem method calls the [add](#) method of the party's [inventory](#) property.

Related topics

[TRpgParty.removeItem](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgParty.removeItem

Class

[TRpgParty](#)

Syntax

```
procedure removeItem(const id, number: word);
```

Description

The `removeItem` method attempts to remove a certain amount of an item from the inventory. It first checks to make sure that **id** refers to a valid item number from the database, and that the inventory contains at least one of that item. If not, nothing happens. Otherwise, it removes **number** number of item **#id** to the inventory. If **number** is more than the quantity of the item in the inventory, it removes all of that item.

Related topics

[TRpgParty.addItem](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgParty.addExp

Class

[TRpgParty](#)

Syntax

```
procedure addExp(const id: smallint; number: integer);
```

Description

The addExp method adds EXP to the party. If **id** is -1, it adds **number** exp to every current member of the party. Otherwise, it adds **number** EXP to the hero occupying slot **#id**. If this is enough to cause the hero to gain a level, his *level* property is adjusted accordingly.

If slot is specified (not -1) and that slot is invalid or empty, it gets redirected to [hero #0](#).

Related topics

[TRpgParty.removeExp](#)

[TRpgParty.addLevels](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgParty.removeExp

Class

[TRpgParty](#)

Syntax

```
procedure removeExp(const id: smallint; number: integer);
```

Description

The addExp method removes EXP from the party. If **id** is -1, it removes **number** exp from every current member of the party. If this would reduce their EXP below 0, their EXP stays at 0. Otherwise, it removes **number** EXP from the hero occupying slot **#id**. If this slot is invalid or empty, it gets redirected to [hero #0](#).

Related topics

[TRpgParty.addExp](#)

[TRpgParty.removeLevels](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgParty.addLevels

Class

[TRpgParty](#)

Syntax

```
procedure addLevels(const id: smallint; number: byte);
```

Description

The addLevels method adds levels to the party. If **id** is -1, it adds **number** exp to every current member of the party. Otherwise, it adds **number** levels to the hero occupying slot **#id**. If this would raise a character's level above the maximum level, it stays at the maximum level. The character's EXP is changed to reflect their new level, and any new skills earned that level are awarded.

If the specified slot is invalid or empty, it gets redirected to [hero #0](#).

This method is not yet fully implemented. Changing levels does not yet adjust a hero's statistics.

Related topics

[TRpgParty.addExp](#)

[TRpgParty.removeLevels](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgParty.removeLevels

Class

[TRpgParty](#)

Syntax

```
procedure removeLevels(const id: smallint; number: byte);
```

Description

The removeLevels method removes levels from the party. If **id** is -1, it takes **number** levels from every current member of the party. Otherwise, it adds **number** levels to the hero occupying slot **#id**. If this would reduce a character's level below 1, it stays at 1. The character's EXP is changed to reflect their new level, and any skills earned at the level(s) lost are removed.

If the specified slot is invalid or empty, it gets redirected to [hero #0](#).

This method is not yet fully implemented. Changing levels does not yet adjust a hero's statistics.

Related topics

[TRpgParty.removeExp](#)

[TRpgParty.addLevels](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgParty.takeDamage

Class

[TRpgParty](#)

Syntax

```
function takeDamage(power: word; defense, mDefense, variance: byte): word;
```

Description

The takeDamage method calls the [takeDamage](#) method for each hero currently in the party. It returns the return value of the takeDamage routine from the last hero who took damage.

Related topics

[TRpgHero.takeDamage](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgEvent Class

TRpgEvent is a class that represents an event on the map. One TRpgEvent object is automatically created per event on the current map. Descends from [TRpgCharacter](#).

[Back to RPG Script Reference](#)
[Explanation of Object Pascal types](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgEvent Properties

[Back to TRpgEvent](#)

[Back to RPG Script Reference](#)

[Explanation of Object Pascal types](#)

Name	Type	Description
map	word	Defines which map # the event is on.
x	word	The event's x-coordinate on the tile map.
y	word	The event's y-coordinate on the tile map.
facing	byte	A number designating which direction the event is facing. If the event is represented by a tile and not a character sprite, it is considered to be facing down by default. The direction values are: down: 2, left: 4, right: 6, up: 8.

NOTE: All TRpgEvent properties are currently [read-only](#). This will be changed in the future as more support for events is added.

This help file was created with the free trial version of [HelpScribble](#).

TRpgEvent Methods

[TRpgEvent](#) does not currently have any methods defined.

This help file was created with the free trial version of [HelpScribble](#).

TRpgTimer Class

TRpgTimer is a class that represents the in-game timer. One TRpgTimer object named *timer* is automatically created during initialization.

[Back to RPG Script Reference](#)

[Explanation of Object Pascal types](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgTimer Properties

[Back to TRpgTimer](#)

[Back to RPG Script Reference](#)

[Explanation of Object Pascal types](#)

Name	Type	Description
time	integer	The number of seconds left on the timer.
visible	boolean	Whether the timer gets shown on the screen or not.
active	boolean	Whether the timer is currently counting down or not. This property is read-only. It gets changed when the appropriate methods are called.
inBattle	boolean	Determines whether or not the timer remains visible and active during battles. (This currently does nothing, as battles are not yet implemented.)

This help file was created with the free trial version of [HelpScribble](#).

TRpgTimer Methods

[Back to TRpgTimer](#)

[Back to RPG Script Reference](#)

go: Starts the timer.

start: Starts the timer and sets certain settings at the same time.

pause: Pauses the timer.

reset: Stops the timer and resets it.

This help file was created with the free trial version of [HelpScribble](#).

TRpgTimer.go

Class

[TRpgTimer](#)

Syntax

```
procedure go;
```

Description

The `go` method starts the timer counting down. It assumes that the *time* property has already been set.

Related topics

[TRpgTimer.start](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgTimer.start

Class

[TRpgTimer](#)

Syntax

```
procedure start(const visible, inBattle: boolean);
```

Description

The start method sets the timer's *visible* and *inBattle* properties to the specified values, then starts the timer counting down. It assumes that the *time* property has already been set.

Related topics

[TRpgTimer.go](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgTimer.pause

Class

[TRpgTimer](#)

Syntax

procedure pause;

Description

The pause method stops the timer in its countdown, but does not reset it.

Related topics

[TRpgTimer.go](#)

[TRpgTimer.reset](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgTimer.reset

Class

[TRpgTimer](#)

Syntax

```
procedure reset;
```

Description

The reset method stops the timer in its countdown, resets the *time* property to 0, and sets the *visible* property to false.

Related topics

[TRpgTimer.pause](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgInventory Class

TRpgInventory is a class that represents the party's inventory. One TRpgInventory object named *inventory* is automatically created as part of the [TRpgParty](#) initialization.

[Back to RPG Script Reference](#)
[Explanation of Object Pascal types](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgInventory Properties

[Back to TRpgInventory](#)

[Back to RPG Script Reference](#)

[Explanation of Object Pascal types](#)

[TRpgInventory](#) does not currently have any properties defined.

This help file was created with the free trial version of [HelpScribble](#).

TRpgInventory Methods

[Back to TRpgInventory](#)

[Back to RPG Script Reference](#)

[add](#): Adds items to the inventory.

[contains](#): Checks to see whether the inventory contains any of a certain item.

[remove](#): Removes items from the inventory.

This help file was created with the free trial version of [HelpScribble](#).

TRpgInventory.add

Class

[TRpgInventory](#)

Syntax

```
procedure add(const id, number: word);
```

Description

The add method attempts to add a certain amount of an item to the inventory. It first checks to make sure that **id** refers to a valid item number from the database. If not, nothing happens. Otherwise, it adds **number** number of item **#id** to the inventory. If this makes the current amount of this item greater than 99, the quantity is set back to 99.

Related topics

[TRpgInventory.remove](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgInventory.remove

Class

[TRpgInventory](#)

Syntax

procedure remove(const id, number: word)

Description

The remove method attempts to remove a certain amount of an item from the inventory. It first checks to make sure that **id** refers to a valid item number from the database, and that the inventory contains at least one of that item. If not, nothing happens. Otherwise, it removes **number** number of item **#id** to the inventory. If **number** is more than the quantity of the item in the inventory, it removes all of that item.

Related topics

[TRpgInventory.add](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgInventory.contains

Class

[TRpgInventory](#)

Syntax

```
function contains(id: word): boolean;
```

Description

The contains method searches through the inventory to determine whether it contains any of the item referred to by **id**. If **id** is not a valid item number from the database, or if the inventory does not contain any of the item, it returns **false**. If the inventory does contain at least one of the item, the return value is **true**.

Related topics

[TRpgHero.equipped](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgVehicle Class

TRpgVehicle is a class that represents the in-game vehicles. Three TRpgVehicle objects are automatically created during initialization: a boat, a ship and an airship. Descends from [TRpgCharacter](#).

[Back to RPG Script Reference](#)

[Explanation of Object Pascal types](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgVehicle Properties

[Back to TRpgVehicle](#)

[Back to RPG Script Reference](#)

[Explanation of Object Pascal types](#)

Name	Type	Description
map	smallint	The ID number of the map the vehicle's on.
x	word	The x-coordinate of the vehicle's current location on the map.
y	word	The y-coordinate of the vehicle's current location on the map.
facing	byte	A number designating which direction the vehicle is facing. The direction values are: down: 2, left: 4, right: 6, up: 8.
inUse	boolean	Shows whether the vehicle is currently active.

This help file was created with the free trial version of [HelpScribble](#).

TRpgVehicle Methods

[Back to TRpgVehicle](#)

[Back to RPG Script Reference](#)

[setSprite:](#) Changes the vehicle's current graphic.

This help file was created with the free trial version of [HelpScribble](#).

TRpgVehicle.setSprite

Class

[TRpgVehicle](#)

Syntax

```
procedure setSprite(filename: string; index: byte);
```

Description

This method changes the vehicle's current sprite (graphics set). It first checks to see if **index** falls within the acceptable range (1..8) and if a CharSet file is available with the name given by **filename**. If either test fails, nothing happens. Otherwise, it sets the vehicle's sprite to the appropriate sprite from that charset.

This help file was created with the free trial version of [HelpScribble](#).

TRpgImage Class

TRpgImage is a class that represents an on-screen image.

[Back to RPG Script Reference](#)

[Explanation of Object Pascal types](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgImage Properties

[Back to TRpgImage](#)

[Back to RPG Script Reference](#)

[Explanation of Object Pascal types](#)

Name	Type	Description
zoom	word	Percent value describing how large the image appears on-screen
timer	cardinal	Length of time that image transitions will take.

This help file was created with the free trial version of [HelpScribble](#).

TRpgImage Methods

[Back to TRpgImage](#)

[Back to RPG Script Reference](#)

[applyImageColors](#): Changes the shading of the image.

[applyImageEffect](#): Applies a graphical effect to the image.

[moveTo](#): Causes the image to move, resize itself, or change transparency.

This help file was created with the free trial version of [HelpScribble](#).

TRpgImage.applyImageColors

Class

[TRpgImage](#)

Syntax

```
procedure applyImageColors(r, g, b, sat: byte);
```

Description

This method changes the image's shading. The **r**, **g**, and **b** parameters affect red, green and blue color balance, and accept percentage values from 0-200. Any number higher than 200 will be treated as a 200. The **sat** parameter affects color saturation of the overall image, from 0 (completely gray) to 200 (exaggeratedly vibrant colors).

The values entered in this routine will be gradually changed over the period of time set in the object's **timer** property. If **timer** is currently at 0, or has not yet been set, the changes will take place immediately.

NOTE: This routine is not yet fully implemented. The **sat** parameter currently does nothing, due to technical limitations. Saturation changes will be implemented during the planned graphics engine rewrite.

This help file was created with the free trial version of [HelpScribble](#).

TRpgImage.applyImageEffect

Class

[TRpgImage](#)

Syntax

```
procedure applyImageEffect(which: TImageEffects; power: byte);
```

Description

This method applies a graphical effect to the image, specified by the **which** parameter. The **power** parameter gives the magnitude of the effect, from 0 to 10. Any value higher than 10 will be treated as a 10.

The `ie_rotate` effect will cause the image to spin continually on-screen, while passing `ie_wave` to **which** will cause the image to ripple.

The values entered in this routine will be gradually changed over the period of time set in the object's **timer** property. If **timer** is currently at 0, or has not yet been set, the changes will take place immediately. Passing `ie_none` to **which** will deactivate all graphical effects immediately. To deactivate an effect gradually, pass that effect with a **power** value of 0.

NOTE: It is possible to set both image effects at the same time. Due to technical limitations, only one can be displayed at the present time. This will be changed in the future. But for now, the ripple effect will take priority over the rotate effect.

Related topics

[TImageEffects Type](#)

This help file was created with the free trial version of [HelpScribble](#).

TRpgImage.moveTo

Class

[TRpgImage](#)

Syntax

```
procedure moveTo(x, y: integer; zoom, opacity: word);
```

Description

This method causes the image to move on-screen to the destination specified by **(x, y)**. It will also change the image's **zoom** property and **opacity** with the appropriate parameters.

The values entered in this routine will be gradually changed over the period of time set in the object's **timer** property. If **timer** is currently at 0, or has not yet been set, the changes will take place immediately.

This help file was created with the free trial version of [HelpScribble](#).

TRpgImage.erase

Class

[TRpgImage](#)

Syntax

procedure erase;

Description

This method erases an on-screen image. After calling it, the image's variable will no longer be valid.

NOTE: Calling **erase** on [image\[0\]](#) has no effect, to avoid potential memory corruption.

This help file was created with the free trial version of [HelpScribble](#).

TSlotList Type

Syntax

type TSlotList = (weapon, shield, armor, helmet, relic, all);

Description

An enumerated type that lists the various slots a hero can equip items in. The "shield" slot is also used for the second weapon if the hero can dual-wield; the name is simply for reference. "all" does not refer to a single slot; calling [unequip](#) with this parameter causes it to unequip all equipped items.

Related topics

[TRpgHero.unequip](#)

This help file was created with the free trial version of [HelpScribble](#).

TMboxLocation Type

Syntax

type TMboxLocation = (mb_top, mb_middle, mb_bottom);

Description

An [enumerated type](#) that lists the positions the message box can occupy on-screen.

Related topics

[MessageOptions routine](#)

This help file was created with the free trial version of [HelpScribble](#).

TBgmTypes Type

Syntax

```
type TBgmTypes = (bgmTitle, bgmBattle, bgmBossBattle, bgmVictory, bgmInn,
bgmBoat, bgmShip, bgmAirship, bgmGameOver);
```

Description

An [enumerated type](#) that lists the various sytem background music presets.

Related topics

[SetSytemMusic routine](#)

This help file was created with the free trial version of [HelpScribble](#).

TSfxTypes Type

Syntax

```
type TSfxTypes = (sfxCursor, sfxAccept, sfxCancel, sfxBuzzer, sfxBattleStart, sfxEscape, sfxEnemyAttack, sfxEnemyDamage, sfxAllyDamage, sfxEvade, sfxEnemyDies, sfxItemUsed);
```

Description

An [enumerated type](#) that lists the various sytem sound effect presets.

Related topics

[SetSytemSound routine](#)

This help file was created with the free trial version of [HelpScribble](#).

TBattleResult Type

Syntax

```
type TBattleResult = (br_victory, br_escaped, br_defeated);
```

Description

An [enumerated type](#) that lists the ways a battle can end.

Related topics

[Battle routine](#)

This help file was created with the free trial version of [HelpScribble](#).

TTransitions Type

Syntax

```
type TTransitions = (trnDefault, trnFadeout, trnBlocks, trnBlockUp, trnBlockDn,
trnBlinds, trnStripeHiLo, trnStripeLR, trnOutIn, trnInOut, trnScrollUp,
trnScrollDn, trnScrollLeft, trnScrollRight, trnDivHiLow, trnDivLR,
trnDivQuarters, trnZoom, trnTwist, trnRipple, trnNone);
```

Description

An [enumerated type](#) that lists the various sytem transitions.

Related topics

[eraseScreen routine](#)

[showScreen routine](#)

[setTransition routine](#)

[TTransitionTypes Type](#)

This help file was created with the free trial version of [HelpScribble](#).

TTransitionTypes Type

Syntax

```
type TTransitionTypes = (trnMapEnter, trnMapExit, trnBattleStartErase,  
trnBattleStartShow, trnBattleEndErase, trnBattleEndShow);
```

Description

An [enumerated type](#) that lists the various times when system transitions are automatically used.

Related topics

[setTransition routine](#)

[TTransitions Type](#)

This help file was created with the free trial version of [HelpScribble](#).

TDirections Type

Syntax

```
type TDirections = (dir_up, dir_right, dir_down, dir_left);
```

Description

An [enumerated type](#) that represents the four cardinal directions.

Related topics

[panScreen routine](#)

This help file was created with the free trial version of [HelpScribble](#).

TWeatherEffects type

Syntax

```
type TWeatherEffects = (we_none, we_rain, we_snow);
```

Description

An [enumerated type](#) that represents the different weather effect patterns.

Related topics

[setWeather routine](#)

This help file was created with the free trial version of [HelpScribble](#).

TImageEffects Type

Syntax

```
type TImageEffects = (ie_none, ie_rotate, ie_wave);
```

Description

An [enumerated type](#) that represents the different graphical effects that can be applied to a [TRpgImage](#).

Related topics

[TRpgImage.applyImageEffect](#)

This help file was created with the free trial version of [HelpScribble](#).

Random routine

Generates a random number.

Syntax

```
function random(one, two: integer): integer;
```

Description

The random routine returns a random integer between the values of parameters **one** and **two**, inclusive. So, *random(5, 17)*; would return any integer value from 5 to 17.

This help file was created with the free trial version of [HelpScribble](#).

ButtonStart routine

Checks to see if the current event script was started by pressing the action button.

Syntax

```
function buttonStart: boolean;
```

Description

The buttonStart routine returns **true** if the current script was started by the user pressing the action button, or **false** if it was started in some other way.

This help file was created with the free trial version of [HelpScribble](#).

Wait routine

Pauses the script execution.

Syntax

```
procedure wait(time: cardinal);
```

Description

This routine causes the script's execution to pause for **time** milliseconds.

This help file was created with the free trial version of [HelpScribble](#).

DeleteCharacter routine

Removes the character the current script is attached to from the map entirely.

Syntax

```
procedure deleteCharacter(permanent: boolean);
```

Description

The deleteCharacter routine deletes the character running the current script from the map, preventing it from running any further scripts. If a script is in progress, it will be cut short. If `true` is passed to **permanent**, the character will be removed for the rest of the game; otherwise, it will load again if the party leaves the map and comes back to it.

This help file was created with the free trial version of [HelpScribble](#).

ShowMessage routine

Displays a message in the in-game message box.

Syntax

```
procedure showMessage(const msg: string);
```

Description

The showMessage routine displays the message box, with the message contained in **msg** in it. The script does not continue until the message box is closed.

Related Topics

[messageOptions routine](#)

[setPortrait routine](#)

[clearPortrait routine](#)

[showChoice routine](#)

[inputNumber routine](#)

This help file was created with the free trial version of [HelpScribble](#).

MessageOptions routine

Sets display options for the in-game message box.

Syntax

```
procedure messageOptions(const transparent: boolean; const position: TMboxLocation; const dontHideHero: boolean; const continueEvents: boolean);
```

Description

The messageOptions routine sets the options for the message box. If **transparent** is true, then the box will not be drawn on the screen; only the message will. The **position** parameter defines which third of the screen the message box displays in. If **dontHideHero** is true, the location in **position** will be temporarily overridden if the message box would cover up the current party. The **continueEvents** parameter is not yet implemented. When parallel events are implemented, if a message box opens while **continueEvents** is set to false, all parallel events currently running will be paused until the box closes.

Related Topics

[showMessage routine](#)

[setPortrait routine](#)

[clearPortrait routine](#)

This help file was created with the free trial version of [HelpScribble](#).

SetPortrait routine

Sets the portrait to be displayed in the in-game message box.

Syntax

```
procedure setPortrait(const filename: string; const index: byte; const rightside, flipped: boolean);
```

Description

The setPortrait routine sets the active portrait to be displayed when the message box is opened with [showMessage](#). It first tests to make sure that **filename** is a valid FaceSet graphic and that **index** falls within the proper range (1..16). If either of these tests fails, nothing happens. Otherwise, the message box is set to display the indicated portrait. If **rightside** is true, it will display on the right side of the message box, otherwise it displays on the left. If **flipped** is true, the portrait will be flipped horizontally.

Related Topics

[showMessage routine](#)

[messageOptions routine](#)

[clearPortrait routine](#)

This help file was created with the free trial version of [HelpScribble](#).

ClearPortrait routine

Turns off the portrait display feature of the in-game message box.

Syntax

```
procedure clearPortrait;
```

Description

The clearPortrait routine clears the portrait to be displayed when the message box is opened with [showMessage](#).

Related Topics

[showMessage routine](#)

[messageOptions routine](#)

[setPortrait routine](#)

This help file was created with the free trial version of [HelpScribble](#).

ShowChoice routine

Shows a choice in the message box.

Syntax

```
function showChoice(input: string; handler: byte): integer;
```

Description

The showChoice routine displays the message box and asks the user to select from the options presented in **input**. The return value is the number of the option that the player chose. The value of **handler** determines how the routine handles the user pressing the Cancel button. If **handler** is 0, then Cancel has no effect. 1..4 causes Cancel to return that value. 5 or higher causes Cancel to return a -1.

NOTE: This routine, in its current implementation, is designed to work around a few obscure quirks in RPG Maker's choice handler and still always translate properly. It uses strange characters and syntax, and it is not recommended for script writers to attempt to write their own showChoice calls until after the Project Importer is finished.

Related Topics

[showMessage routine](#)

[inputNumber routine](#)

This help file was created with the free trial version of [HelpScribble](#).

InputNumber routine

Uses the message box to input a number from the user.

Syntax

```
function inputNumber(const digits: byte): integer
```

Description

The inputNumber routine displays the message box and has the user input an integer with a number of digits equal to the **digits** parameter. The return value is the number that the user enters.

Related Topics

[showMessage routine](#)

[showChoice routine](#)

This help file was created with the free trial version of [HelpScribble](#).

Inn routine

Calls the inn.

Syntax

```
function inn(messageStyle: byte; cost: integer): boolean;
```

Description

The inn routine begins the Inn system. It asks if the party would like to rest at the inn, and presents a choice. If the party does not have at least **cost** gold on hand, the "Yes" option is grayed out. If the player chooses "Yes", all heroes are fully healed via the [fullheal](#) routine, and the function returns **true**. If the player chooses "No" or presses the cancel button, the function returns **false**.

NOTE: Certain MIDI files will not work correctly with this routine. See [MIDI Lead-in Issue](#) for details.

This help file was created with the free trial version of [HelpScribble](#).

HeldItems routine

Checks to see how many of a certain item the party has.

Syntax

```
function heldItems(const index: word; const equipped: boolean): word;
```

Description

The heldItems routine sees how many items of the type **index** refers to that the party currently has. If **equipped** is true, it counts how many are equipped by current party members; otherwise, it returns the number of the item in the inventory. If **index** is an invalid number, it returns 0.

Related Topics

[addItem routine](#)

[removeItem routine](#)

This help file was created with the free trial version of [HelpScribble](#).

HeroJoin routine

Adds a hero to the party.

Syntax

```
procedure heroJoin(const id: byte);
```

Description

The heroJoin attempts to add hero **#id** to the party. It does nothing if the hero is already in the party, the party is full, or **id** is not a valid hero record number.

Related Topics

[heroLeave routine](#)

This help file was created with the free trial version of [HelpScribble](#).

HeroLeave routine

Removes a hero from the party.

Syntax

```
procedure heroLeave(const id: byte);
```

Description

The heroJoin attempts to remove hero **#id** from the party. It does nothing if the hero is not in the party or if **id** is not a valid hero record number.

Related Topics

[heroLeave routine](#)

This help file was created with the free trial version of [HelpScribble](#).

AddItem routine

Adds items to the inventory.

Syntax

```
procedure addItem(const id, number: word);
```

Description

This procedure calls the [party.addItem](#) routine. It is included for convenience while writing scripts.

Related Topics

[heldItems routine](#)

[removeItem routine](#)

[TRpgParty.addItem](#)

This help file was created with the free trial version of [HelpScribble](#).

RemoveItem routine

Removes items from the inventory.

Syntax

```
procedure removeItem(const id, number: word);
```

Description

This procedure calls the [party.removeItem](#) routine. It is included for convenience while writing scripts.

Related Topics

[heldItems routine](#)

[removeItem routine](#)

[TRpgParty.addItem](#)

This help file was created with the free trial version of [HelpScribble](#).

AddExp routine

Adds experience to heroes.

Syntax

```
procedure addExp(const id: smallint; number: integer);
```

Description

The addExp routine adds EXP to heroes. If **id** is -1, it calls the [party.addExp](#) routine. Otherwise, it adds EXP to hero **#id**. If **id** is an invalid number, it gets redirected to [hero #0](#). If a character's EXP value goes above the maximum EXP value, it gets set to the maximum.

After adding EXP, if [party.levelNotify](#) is true, the routine shows a message if any heroes gained a level.

Related Topics

[removeExp routine](#)

[TRpgParty.addExp](#)

This help file was created with the free trial version of [HelpScribble](#).

RemoveExp routine

Removes experience from heroes.

Syntax

```
procedure removeExp(const id: smallint; number: integer);
```

Description

The removeExp routine removes EXP from heroes. If **id** is -1, it calls the [party.removeExp](#) routine. Otherwise, it removes EXP from hero #**id**. If **id** is an invalid number, it gets redirected to [hero #0](#). If a character's new EXP value drops below 0, it gets set to 0.

After subtracting EXP, if [party.levelNotify](#) is true, the routine shows a message if any heroes lost a level.

Related Topics

[addExp routine](#)

[TRpgParty.removeExp](#)

This help file was created with the free trial version of [HelpScribble](#).

AddLevels routine

Adds levels to heroes.

Syntax

```
procedure addLevels(const id: smallint; number: integer);
```

Description

The addExp routine adds levels to heroes. If **id** is -1, it calls the [party.addLevels](#) routine. Otherwise, it adds EXP to hero #**id**. If **id** is an invalid number, it gets redirected to [hero #0](#). If the new total is greater than the character's maximum level, it gets set to that level.

After adding levels, if [party.levelNotify](#) is true, the routine shows a message indicating that the hero(es) gained a level.

Related Topics

[removeLevels routine](#)

[TRpgParty.addLevels](#)

This help file was created with the free trial version of [HelpScribble](#).

RemoveLevels routine

Removes levels from heroes.

Syntax

```
procedure addLevels(const id: smallint; number: integer);
```

Description

The addExp routine adds levels to heroes. If **id** is -1, it calls the [party.addLevels](#) routine. Otherwise, it adds EXP to hero #**id**. If **id** is an invalid number, it gets redirected to [hero #0](#). If the new level is less than 1, it gets set to 1.

After adding levels, if [party.levelNotify](#) is true, the routine shows a message indicating that the hero(es) lost a level.

Related Topics

[addLevels routine](#)

[TRpgParty.removeLevels](#)

This help file was created with the free trial version of [HelpScribble](#).

SetSystemMusic routine

Changes the system background music for various events.

Syntax

```
procedure setSystemMusic(const which: TBgmTypes; const newSong: string);
```

Description

The setSystemMusic routine changes a system BGM preset. The system music to be changed is specified by **which**, and **newsong** is the filename of the new system music to use.

Related Topics

[TBgmTypes type](#)

This help file was created with the free trial version of [HelpScribble](#).

SetSystemSound routine

Changes the preset system SFX for various events.

Syntax

```
procedure setSystemSound(const which: TSfxTypes; const newSound: string);
```

Description

The setSystemSound routine changes a system SFX preset. The system music to be changed is specified by **which**, and **newSound** is the filename of the new system sound to use.

Related Topics

[TSfxTypes type](#)

This help file was created with the free trial version of [HelpScribble](#).

SetSkin routine

Changes the current system "skin" graphics set.

Syntax

```
procedure setSkin(const name: string);
```

Description

The setSkin routine loads a new system "skin" graphic set. This graphic defines the look of various elements of the game, including menus, cursors and text boxes.

NOTE: setSkin will not run while a message box of any type is currently displayed. For technical reasons, the routine has to unregister the various graphical elements that make up a message box, then register the new ones, and while this is done very quickly, it is not quick enough to prevent the graphics engine from crashing by trying to display an image that's not registered properly. If setSkin is called while a message box is on-screen, it will wait and run once the box has closed.

This help file was created with the free trial version of [HelpScribble](#).

SetTransition routine

Changes the default transition settings.

Syntax

```
procedure setTransition(const which: TTransitionTypes; const newTransition: TTransitions);
```

Description

This routine changes the default system transition specified by **which** to the value passed to **newTransition**. It does nothing if `trnDefault` is passed.

Related topics

[eraseScreen routine](#)

[showScreen routine](#)

[TTransitions Type](#)

[TTransitionTypes Type](#)

This help file was created with the free trial version of [HelpScribble](#).

PlayMusic routine

Changes the background music.

Syntax

```
procedure playMusic(name: string; time, volume, tempo, balance: word);
```

Description

This routine changes the current background music to the song specified by **name**. No path or file extension is necessary; the engine will automatically search in the appropriate folders for background music, and check all valid music extensions. If the **time** parameter is greater than 0, the song will fade in over that many milliseconds. The **volume** parameter tells what percentage of the maximum volume the song will play at.

NOTE: The **tempo** and **balance** parameters are not currently implemented, due to technical limitations.

WARNING: Playback of certain MIDI files will be delayed by the [MIDI lead-in issue](#).

Related Topics

[fadeOutMusic routine](#)

[memorizeBgm routine](#)

This help file was created with the free trial version of [HelpScribble](#).

FadeOutMusic routine

Causes the background music to gradually fade out.

Syntax

```
procedure fadeOutMusic(time: word);
```

Description

This routine causes the current background music to fade to silence, over a period of **time** milliseconds. Passing 0 to **time** will immediately stop the background music.

Related Topics

[playMusic routine](#)

This help file was created with the free trial version of [HelpScribble](#).

MemorizeBgm routine

Causes the system to remember the current background music.

Syntax

```
procedure memorizeBgm;
```

Description

This routine stores the current BGM in the system's memory. This makes it easy to change to another song temporarily and then change back, without having to keep track of the current song.

Related Topics

[playMusic routine](#)

[playMemorizedBgm routine](#)

This help file was created with the free trial version of [HelpScribble](#).

PlayMemorizedBgm routine

Causes the system to play previously memorized background music.

Syntax

```
procedure playMemorizedBgm;
```

Description

This routine starts playing the BGM that was memorized by the most recent call to the [memorizeBgm routine](#). If `memorizeBgm` has not been called yet, it does nothing.

Related Topics

[playMusic routine](#)

[memorizeBgm routine](#)

This help file was created with the free trial version of [HelpScribble](#).

PlaySound routine

Plays a sound effect.

Syntax

```
procedure playSound(name: string; volume, tempo, balance: word);
```

Description

This routine plays the sound effect specified by **name**. No path or file extension is necessary; the engine will automatically search in the appropriate folders for sound effects, and check all valid sound extensions. The **volume** parameter tells what percentage of the maximum volume the song will play at, and the **balance** parameter defines left/right balance, from 0 (full left) to 100 (full right).

If **name** does not refer to a valid sound effect, no sound is played.

NOTE: The **tempo** parameter is not currently implemented, due to technical limitations.

This help file was created with the free trial version of [HelpScribble](#).

KeyScan routine

Checks to see if any commands are being entered from the keyboard.

Syntax

```
function keyScan(mask: word; wait: boolean): byte;
```

Description

The keyScan routine checks to see if any command keys are being pressed. This includes arrow keys as well as action buttons. The **mask** parameter determines which keys are checked for, according to the predefined key constants. If **true** is passed to **wait**, then the script will pause until the player presses a key, but if **wait** is **false**, it will scan the keyboard once and return a value based on whatever key is currently pressed.

Mask key constants:

NAME	VALUE
KS_DOWN:	down arrow 1
KS_UP:	up arrow 2
KS_LEFT:	left arrow 4
KS_RIGHT:	right arrow 8
KS_DIRS:	any arrow key 0xF
KS_ACTION:	action button (enter) 0x10
KS_CANCEL:	cancel button (escape) 0x20
KS_ALL:	all input keys. 0xFFFF

These names represent hexadecimal numbers, and can be combined with the word **or**. For example, to check for either the action button or the cancel button, but not the arrow keys, you would pass "**KS_ACTION or KS_CANCEL**" to **mask**. The **KS_DIRS** and **KS_ALL** variables include several numbers already combined, to simplify things.

Return values:

- 0: No key pressed.
- 1: Down arrow
- 2: Left arrow
- 3: Right arrow
- 4: Up arrow
- 5: Action button
- 6: Cancel button

NOTE: If **0** is passed to **mask**, there will be nothing to scan for, and the routine will immediately return a value of 0, even if **wait** is **true**. This is to avoid deadlocking the script in an infinite loop.

NOTE: KS_ALL does not check for <SPACE>, which opens the console window instead of sending input to the game itself.

NOTE: This routine automatically delays 10 milliseconds (1/100 of a second) before returning when **wait** is `false`. This is necessary to avoid a timing issue that could stall the program if keyScan is used in a loop. This is a very short delay and should not cause any problems, but any scripts that depend on very precise timing may need to take this delay into account.

This help file was created with the free trial version of [HelpScribble](#).

CallGlobalEvent routine

Causes a global event to run.

Syntax

```
procedure callGlobalEvent(id: word);
```

Description

The callGlobalEvent routine invokes event # **id** from the global event set. This routine also ends the current script. If **id** is not a valid event number, no event is called, but the current script still ends.

Related Topics

[callEvent routine](#)

This help file was created with the free trial version of [HelpScribble](#).

CallEvent routine

Causes an event from the current map to run.

Syntax

```
procedure callEvent(event, page: word);
```

Description

The callEvent routine invokes the event attached to the page specified by **page**, of the character # **event** from the current map. This routine also ends the current script. If either **event** or **page** is not a valid ID number, no event is called, but the current script still ends.

Related Topics

[callGlobalEvent routine](#)

This help file was created with the free trial version of [HelpScribble](#).

OpenMenu routine

Opens the system menu.

Syntax

```
procedure openMenu;
```

Description

The openMenu routine opens the system menu. It will work even if the global **menuEnabled** flag is set to false.

NOTE: Don't change the composition of the current party from the [console](#) while the menu is open. This can cause system errors.

This help file was created with the free trial version of [HelpScribble](#).

PrepareStore routine

Sets up a store inventory for the shop system to use.

Syntax

```
function prepareStore(merchandise: string): word;
```

Description

The PrepareStore function prepares an inventory for the shop system to use in its stores. The **merchandise** parameter is a string consisting of several numbers separated by spaces. Each number is the database ID number of an item to be sold in the shop. A typical merchandise string would look something like this:

5 7 28 13 20 21 12

Any number that is not a valid item ID is ignored. The function returns the ID number of the newly-created shop. It is the designer's responsibility to keep track of this number. The ID number might be different each time the game runs, but does not change after it has been initially created.

Related topics

[Shop routine](#)

This help file was created with the free trial version of [HelpScribble](#).

Shop routine

Opens the shopping system.

Syntax

```
function shop(style: word; messageStyle: byte; store: word): boolean;
```

Description

The shop routine begins the shopping system. The **style** parameter determines which type of shop will be opened: 0 is a buy/sell shop, 1 is a buy-only shop, and 2 is a sell-only shop. The **messageStyle** parameter refers to the set of shop messages to be displayed. The **store** parameter refers to a store inventory previously set up by the [prepareStore routine](#).

If an invalid number is passed to any of these parameters, it will be corrected to a default value: the lowest valid number for each case. A default [shop # 0](#) that sells nothing is created automatically during setup.

The routine returns **true** if anything has been successfully bought or sold, or **false** otherwise.

Related topics

[PrepareStore routine](#)

This help file was created with the free trial version of [HelpScribble](#).

InputText Routine

Allows the user to enter a text string.

Syntax

```
function inputText(start: string; heroId: word): string;
```

Description

This routine opens the Text Input menu, allowing the user to enter a string of text. This is commonly used for entering custom hero names, but can serve other purposes as well.

The **start** parameter contains an optional default string to be displayed. This can be used to provide the default name for a hero. The **heroId** parameter is the ID number of a hero whose name is being entered. This is used to display the hero's portrait. If **0** is passed to heroId, no portrait will be displayed. The function returns the string that the user enters.

Example

To set a name for hero #1:

```
hero[1].name := inputText(hero[1].name, 1);
```

This help file was created with the free trial version of [HelpScribble](#).

Battle routine

Begins a battle. (Not yet implemented.)

Syntax

```
function battle(which: word; allow_escape, first_strike: boolean): TBattleResult;
```

Description

The battle routine starts a battle. The battle functionality is not yet implemented; the current routine simply returns a result of [br_victory](#) without running a battle.

Related Topics

[TBattleResult type](#)

This help file was created with the free trial version of [HelpScribble](#).

MemorizeLocation routine

Stores the current party's location in three variables. (Currently bugged.)

Syntax

```
procedure memorizeLocation(var map, x, y: integer);
```

Description

This routine stores the location of the current party to three variables, denoting the current map and the party's X and Y coordinates.

Related topics

[Teleport routine](#)

This help file was created with the free trial version of [HelpScribble](#).

Teleport routine

Teleports the current party to another location. (Partially implemented.)

Syntax

```
procedure teleport(map, x, y: word);
```

Description

This routine teleports the current party to a new location, as specified by the map ID # and coordinates passed to the routine. It does nothing if the location is not on the map.

NOTE: Because multiple maps have not yet been implemented, this routine currently does nothing if the **map** parameter is not the same as the current map ID#.

WARNING: The routine does not check whether or not the destination square is valid terrain to walk on. It's possible to teleport the party to invalid terrain, and get stuck there.

Related topics

[MemorizeLocation routine](#)

[TeleportVehicle routine](#)

This help file was created with the free trial version of [HelpScribble](#).

TeleportVehicle routine

Teleports a specified vehicle to another location.

Syntax

```
procedure teleportVehicle(which: TRpgVehicle; map, x, y: integer);
```

Description

This routine teleports the specified vehicle to a new location, as specified by the map ID # and coordinates passed to the routine. It does nothing if the location is not within the map's boundaries. It also does nothing if the vehicle is currently carrying the active party and the destination is on a different map; the [teleport](#) routine should be used in this case.

WARNING: The routine does not check whether or not the destination square is valid terrain for the vehicle. It's possible to teleport the vehicle to invalid terrain, which could cause movement problems.

Related topics

[teleport routine](#)

This help file was created with the free trial version of [HelpScribble](#).

TeleportEvent routine

Teleports a specified event to another location on the map.

Syntax

```
procedure teleportEvent(which: TRpgEvent; x, y: integer);
```

Description

This routine teleports the event specified by **which** to a new location, specified by the coordinates passed to the routine. Events can only be teleported within the current map. If the location is not within the boundaries of the map, it does nothing.

Related topics

[Teleport routine](#)

[SwapEvents routine](#)

This help file was created with the free trial version of [HelpScribble](#).

SwapEvents routine

Exchanges the position of two events on the map.

Syntax

```
procedure swapEvents(first, second: TRpgEvent);
```

Description

This routine swaps the position of the events passed to **first** and **second**.

Related topics

[Teleport routine](#)

This help file was created with the free trial version of [HelpScribble](#).

RideVehicle routine

Causes the party to board/leave a vehicle.

Syntax

```
procedure rideVehicle;
```

Description

This routine causes the party to ride a vehicle in either the current square or the square immediately in front of the party, or to attempt to leave the vehicle if the party is already aboard a vehicle. If no vehicle is available, the routine does nothing.

Related topics

[Teleport routine](#)

This help file was created with the free trial version of [HelpScribble](#).

GetTerrainID routine

Returns the ID # of the map terrain at a certain square.

Syntax

```
function getTerrainID(x, y: word): word;
```

Description

This function is used to find the terrain type from the database of the lower tile on the map at grid position **x,y**. If **x,y** is not a valid location, the function returns a value of 0.

Related topics

[getEventID routine](#)

This help file was created with the free trial version of [HelpScribble](#).

GetEventID routine

Returns the ID # of the event at a certain square.

Syntax

```
function getEventID(x, y: word): word;
```

Description

This function is used to find the ID # of the event on the map at grid position **x,y**. If **x,y** is not a valid location, or if there is no event at that square, the function returns a value of 0.

Related topics

[getTerrainID routine](#)

This help file was created with the free trial version of [HelpScribble](#).

EraseScreen routine

Clears the screen, optionally using a system transition.

Syntax

```
procedure eraseScreen(whichTransition: TTransitions);
```

Description

This routine erases the screen, using the method specified by **whichTransition**.

If `trnDefault` is passed to **whichTransition**, the default map exit transition is used. The routine does nothing if the screen is already blank.

Related topics

[showScreen routine](#)

[setTransition routine](#)

[TTransitions Type](#)

This help file was created with the free trial version of [HelpScribble](#).

ShowScreen Routine

Restores the screen after an erase, optionally using a system transition.

Syntax

```
procedure showScreen(whichTransition: TTransitions);
```

Description

This routine restores the screen from the blank state set by the [eraseScreen](#) routine, using the method specified by **whichTransition**. If `trnDefault` is passed to **whichTransition**, the default map entrance transition is used. The routine does nothing if the screen is not already blank.

Related topics

[eraseScreen routine](#)

[setTransition routine](#)

[TTransitions Type](#)

This help file was created with the free trial version of [HelpScribble](#).

SetScreenTone routine

Changes the overall color of the screen.

Syntax

```
procedure setScreenTone(r, g, b, sat: byte; duration: cardinal; wait: boolean);
```

Description

This routine shades the entire map. The first four parameters represent percentages, and can be anywhere from 0-200. (Any value larger than 200 is treated as a 200). The **r**, **g**, and **b** parameters refer to the red, green and blue components of the shading. The **sat** parameter refers to the overall color saturation of the map. The transition will take place gradually, over a period of **duration** milliseconds. If **true** is passed to the **wait** parameter, the script's execution will pause until the transition is finished.

This routine only shades the map and the characters and events on it. Menus and message boxes aren't shaded.

NOTE: The Fade Out screen transition, as well as other script functions that may make use of fade-outs (such as the [Inn routine](#)) will fade from the current screen tone to black. The previous screen tone will be lost, and it is the designer's responsibility to restore it when the screen returns to normal.

NOTE: This routine is not yet fully implemented. The **sat** parameter currently does nothing, due to technical limitations. Saturation changes will be implemented during the planned graphics engine rewrite.

Related topics

[flashScreen routine](#)

This help file was created with the free trial version of [HelpScribble](#).

FlashScreen routine

Flashes the screen a certain color.

Syntax

```
procedure flashScreen(r, g, b, power: byte; duration: cardinal; wait: boolean);
```

Description

This routine causes a flash to appear, briefly shading the entire map, then fading. The first four parameters represent color values, and can be anywhere from 0-255. The **r**, **g**, and **b** parameters refer to the red, green and blue components of the flash. The **power** parameter refers to the initial opacity of the flash. The flash color will appear on-screen immediately, then gradually fade out (become more transparent) over a period of **duration** milliseconds. If **true** is passed to the **wait** parameter, the script's execution will pause until the flash has faded completely. This routine only flashes the map and the characters and events on it. Menus and message boxes aren't included.

NOTE: The routine uses different code than the [setScreenTone](#) routine uses. A screen tone and a flash can be on-screen at the same time. If so, the flash is drawn on top of the already-shaded screen.

Related topics

[setScreenTone routine](#)

[TRpgCharacter.flash](#)

This help file was created with the free trial version of [HelpScribble](#).

LockScreen routine

Locks the current screen position.

Syntax

```
procedure lockScreen;
```

Description

This routine causes the screen to stop scrolling to follow the party's movements. Teleporting will recenter the screen on the new location, but it will remain locked in place in the new position until [unlockScreen](#) is called. The screen can still be scrolled with the [panScreen](#) routine while it is locked.

Related topics

[unlockScreen routine](#)

[panScreen routine](#)

This help file was created with the free trial version of [HelpScribble](#).

UnlockScreen routine

Unlocks the current screen position.

Syntax

```
procedure unlockScreen;
```

Description

This routine causes the screen to go back to following the party as it moves, if it was previously locked by [lockScreen](#).

Related topics

[lockScreen routine](#)

[panScreen routine](#)

This help file was created with the free trial version of [HelpScribble](#).

PanScreen routine

Pans the screen from the current camera position.

Syntax

```
procedure panScreen(direction: TDirections; distance: word; speed: byte; wait: boolean);
```

Description

This routine causes the screen to move away from the current position, in the direction specified by **direction**, by **distance** number of squares. The **speed** parameter indicates the speed at which the screen scrolls, from 1 (slowest) to 6 (fastest). Passing 0 or any other invalid value causes the scroll speed to remain unchanged. If `true` is passed to the **wait** parameter, the script's execution will pause until the pan is complete.

Related topics

[TDirections Type](#)

[panScreenTo routine](#)

[returnScreen routine](#)

This help file was created with the free trial version of [HelpScribble](#).

PanScreenTo routine

Pans the screen to a certain position.

Syntax

```
procedure panScreenTo(x, y: word; speed: byte; wait: boolean);
```

Description

This routine causes the screen to move to recenter itself around the point specified by the coordinates **x** and **y**. The **speed** parameter indicates the speed at which the screen scrolls, from 1 (slowest) to 6 (fastest). Passing 0 or any other invalid value causes the scroll speed to remain unchanged. If **true** is passed to the **wait** parameter, the script's execution will pause until the pan is complete.

Related topics

[panScreen routine](#)

[returnScreen routine](#)

This help file was created with the free trial version of [HelpScribble](#).

ReturnScreen routine

Causes the screen to return to the current party.

Syntax

```
procedure returnScreen(speed: byte; wait: boolean);
```

Description

This routine causes the screen to move to recenter itself around the current party after having been moved away from it with the [panScreen](#) or [panScreenTo](#) routines. The **speed** parameter indicates the speed at which the screen scrolls, from 1 (slowest) to 6 (fastest). Passing 0 or any other invalid value causes the scroll speed to remain unchanged. If **true** is passed to the **wait** parameter, the script's execution will pause until the pan is complete.

Related topics

[panScreen routine](#)

[panScreenTo routine](#)

This help file was created with the free trial version of [HelpScribble](#).

SetWeather routine

Sets the current weather effect.

Syntax

```
procedure setWeather(effect: TWeatherEffects; severity: byte);
```

Description

This routine sets the state of the weather effect system. The **effect** parameter specifies [which effect to set](#), and **severity** describes how strong the weather should be, from 0 (none) to 10 (strong). Any value higher than 10 is treated as a 10. If [we_off](#) is passed to **effect**, then the weather is turned off and **severity** has no effect.

NOTE: Any weather effect will continue to display on the map, independently of any other changes. This includes teleport events that change the scene to a different map.

Related topics

[TWeatherEffects Type](#)

[increaseWeather routine](#)

[decreaseWeather routine](#)

This help file was created with the free trial version of [HelpScribble](#).

IncreaseWeather routine

Increases the severity of the current weather effect.

Syntax

```
procedure increaseWeather;
```

Description

This routine increases the strength of the active weather effect by 1, up to a maximum value of 10. It has no effect if the weather is currently turned off.

Related topics

[setWeather routine](#)

[decreaseWeather routine](#)

This help file was created with the free trial version of [HelpScribble](#).

DecreaseWeather routine

Decreases the severity of the current weather effect.

Syntax

```
procedure decreaseWeather;
```

Description

This routine decreases the strength of the active weather effect by 1, down to a minimum value of 0. It has no effect if the weather is currently turned off.

Related topics

[setWeather routine](#)

[increaseWeather routine](#)

This help file was created with the free trial version of [HelpScribble](#).

NewImage Routine

Sets up a new on-screen image.

Syntax

```
function newImage(name: string; x, y: integer; zoom, transparency: word; pinned, mask: boolean):  
TRpgImage;
```

Description

This routine creates a new on-screen image, which is loaded from the file named in the **name** parameter. No path or file extension is necessary; the engine will automatically search in the appropriate folders for on-screen images, and check all valid image extensions.

The image is centered around the point (**x**, **y**) on the screen. The **zoom** parameter denotes what percentage of the image's original size it will be displayed as on-screen, and **transparency** is also a percent value, from 0 (completely opaque) to 100 (completely transparent and invisible.) If **true** is passed to **pinned**, the image will be "pinned to the camera" and always remain in the same position on-screen. Otherwise, it will be located on the map, and move on-screen when the map scrolls. The **mask** parameter specifies whether or not to use a "mask color" for image transparency.

If **name** is not a valid filename, the image created will be invisible, with a size of 0x0.

Related topics

[TRpgImage Class](#)

This help file was created with the free trial version of [HelpScribble](#).

SetBGImage routine

Changes the current background image.

Syntax

```
procedure setBGImage(name: string; scrollX, scrollY: shortint; autoX, autoY: boolean);
```

Description

This routine sets a new background image, which is loaded from the file named in the **name** parameter. No path or file extension is necessary; the engine will automatically search in the appropriate folders for background images, and check all valid image extensions.

If the **scrollX** and **scrollY** parameters are not 0, the background image will continually be in motion. A positive **scrollX** causes it to scroll to the right; negative **scrollX** makes it scroll to the left. Likewise, positive **scrollY** causes downward movement, and negative **scrollY** will make the image scroll upwards.

If **true** is passed to the **autoX** or **autoY** parameters, their scrolling in the corresponding direction will automatically be 1/2 of the camera's movement distance. This overrides the values of **scrollX** and **scrollY**. Auto-scrolling can be used to create a "parallax" effect.

This help file was created with the free trial version of [HelpScribble](#).

ShowBattleAnim routine

Displays a battle animation on-screen.

Syntax

```
procedure showBattleAnim(which: word; target: TRpgCharacter; wait, fullscreen: boolean);
```

Description

This routine plays a battle animation from the database. The **which** parameter indicates the ID number of the animation to play, and **target** specifies the target of the animation. If **true** is passed to **wait**, the script's execution will pause until the animation is finished, and if **fullscreen** is **true**, the animation will center itself at the center of the screen instead of on the target character, although a valid **target** parameter is still necessary for proper script execution.

If **which** is not a valid animation ID, **target** is not a valid character, or the graphics filename listed in the database for the animation does not refer to a valid image file, no animation will play.

This help file was created with the free trial version of [HelpScribble](#).

PrepareRoute routine

Sets up a new move script for characters to use.

Syntax

```
function prepareRoute(route: string; loop: boolean): word;
```

Description

The `prepareRoute` routine creates a new move script, according to the parameters passed to **route**. If `true` is passed to **loop**, the script will loop continually; if **loop** is `false`, it will only run one time. The routine returns an ID number for the new move script, which can be passed to [TRpgCharacter.move](#). If the route entered is identical to an existing move script, it returns that route's ID number instead of creating a duplicate.

The **route** string is made up of numbers separated by spaces. Each number represents either a movement command or, in a few specific cases, extra data for the previous movement command. The format is currently based on RPG Maker's somewhat convoluted system, and it is not recommended to attempt to write route strings manually. The system will be simplified in a later version.

Related topics

[stopMoveScripts routine](#)
[TRpgCharacter.move](#)

This help file was created with the free trial version of [HelpScribble](#).

WaitUntilMoved routine

Pauses the script until all characters have finished their move scripts.

Syntax

```
procedure waitUntilMoved;
```

Description

This routine pauses the current script until all characters on the map with assigned move scripts have completed the move script at least once. Characters that move by their default movement parameters are not included.

NOTE: If another script calls [stopMoveScripts](#) while waitUntilMoved is waiting, the waiting will immediately end, as all assigned move scripts will be canceled.

WARNING: If any character is running a move script with the **ignore** parameter set to `false`, it is possible for the waitUntilMoved routine to deadlock a script indefinitely if the character runs into an obstacle that does not move.

Related topics

[stopMoveScripts routine](#)

[TRpgCharacter.move](#)

This help file was created with the free trial version of [HelpScribble](#).

StopMoveScripts routine

Cancels all active move scripts.

Syntax

```
procedure stopMoveScripts;
```

Description

This routine stops all assigned move scripts for all characters on the current map. It does not affect the characters' default movement, though.

Related topics

[waitUntilMoved routine](#)

[TRpgCharacter.move](#)

This help file was created with the free trial version of [HelpScribble](#).

HelpScribble - The Complete Help Authoring Tool

This help file was created with the free evaluation version of [HelpScribble](#). **This ad and the little footers below each topic will be automatically removed when the author recompiles this help file with the full version of HelpScribble.** HelpScribble can be [purchased securely online](#) and is covered by a risk-free 90-day unconditional money-back [guarantee](#).

About HelpScribble

HelpScribble is a full-featured, easy-to-use help authoring tool for creating help files from start to finish. You can create [WinHelp \(.hlp\) files](#), [HTML Help \(.chm\) files](#), a [printed manual](#) and [online documentation \(on a web site\)](#) all from the same HelpScribble project.

If you have previously used another help authoring tool, you can reuse your work by importing the HPJ+RTF files created with the other tool or by a decompiler.

You can use the help files you make with HelpScribble to provide [context-sensitive help with your Window applications](#), no matter which development tool you use. You can also use HelpScribble to create stand-alone [portable documents](#).

Write Perfect Help Files for Your Software

You need HelpScribble to create a top quality help file for your software. Windows users expect to receive detailed documentation at a quick press of the F1 button. You will sell more copies of your software if a thoughtful help file flattens the learning curve for prospects trying the evaluation version of your product.

No matter which development tool you are using, you can always use a help file created with HelpScribble to provide context-sensitive help for your application.

If you know how to use a word processor, you can build a help file with HelpScribble. Creating new topics and adding links to them cannot be easier. No experience required.

[Read more...](#)

Portable Documents with Hyperlinks

With HelpScribble you can easily create hyperlinked documents that can be read on any computer running Windows.

The WinHelp files that you create with HelpScribble are a collection of linked pages, just like a web site. But unlike a web site, you only need to distribute one file. This file can be read on any Windows computer. No other software is needed.

You need absolutely no programming experience to use HelpScribble. Many word processors and HTML editors are more complicated than HelpScribble.

[Read more...](#)

The Complete Help Authoring Tool

HelpScribble is a full-featured stand-alone help authoring tool. It is not an add-on for MS Word like so many other help writing products. You can create the entire help file right within HelpScribble.

With HelpScribble you can even provide your documentation in several forms, all from the same source. You can add context-sensitive help to your application, print a manual and provide online documentation on your web site.

WinHelp

WinHelp is the help file format supported by all versions of Windows. It is fast, compact and supported by Windows 3.x, NT 3.x, 95, 98, ME, NT4, 2000 and XP. All Windows users are familiar with it. With any Windows development tool you can use WinHelp files to add context-sensitive help to your applications.

With HelpScribble you have all the functionality you need to build a complete help file, including table of contents, index, browse sequences, etc. With HelpScribble's SHG (image map) editor, you can make your pictures (screen shots) say more than a thousand words.

[Read more...](#)

Printed Manual

HTML Help

With Windows 98, Microsoft introduced a new help file format called HTML Help. Using HelpScribble, you can build classic WinHelp .hlp files and new HTML Help .chm files from the same source. You decide whether you want maximum compatibility with older computers, or if you want to offer a state of the art help system.

[Read more...](#)

Web Help

If you want to provide online documentation, you can create one large HTML file or a collection of HTML files from your HelpScribble project and upload those to your web site. Your customers will have instant access to the latest documentation at any time.

Reading from paper is a lot easier than reading from a screen. A printed manual or, for downloadable software, a printable manual in PDF format, adds greatly to the value of your product.

You can use your HelpScribble project as the basis for your printed manual. You can export your help text to one large RTF or HTML file that you or your DTP professional can turn into a finished manual using a word processor or DTP program.

[Read more...](#)

[Read more...](#)

Cross-Platform Help

If you are developing platforms such as Linux, where there is no standard help format, you can provide your documentation as HTML files. The user can easily view those files in a browser. If you use Kylix, you can even easily integrate them into your application.

[Read more...](#)

Number 1 Help Tool for Delphi & C++Builder

You can use HelpScribble no matter which development tool you use. Even if you use no development tool at all. (Stand-alone help files can be very useful.) But if you are a Delphi or C++Builder developer, you will certainly appreciate the way HelpScribble integrates with your favorite development tool.

HelpScribble's HelpContext property editor takes most of the tedious job of assigning Topic IDs to HelpContext properties out of your hands. Adding context-sensitive help to your application could not be easier. Simply click on the corresponding controls and help topics to link them.

Delphi developers like HelpScribble so much that it won the Delphi Informant Readers Choice awards three years in a row. In 1999, 2000 and 2001, HelpScribble was chosen the Best Help Authoring Package, beating well-known tools such as RoboHelp and ForeHelp.

[Read more...](#)

Do Not Take My Word For It

I could rattle on all day about how great HelpScribble is. But do not take my word for it!

Find out [what the people say about HelpScribble](#) before you decide to [buy](#).

Chris Kryza

"Just wanted to let you folks know that you have created a very, very, very nice product! The last time I had to generate a context sensitive help file I used Doc-To-Help and I remember it took me weeks to get it right. With HelpScribble I was able to put together a complete help system in under 2 days and I then created a web-help deployment of same in seconds. I used your demo version to create the entire project and then purchased your product online after I saw how good it all looked. Great stuff!"

Roj Ash

"I just felt I had to compliment you on this product. I have no experience of producing Help files, and having evaluated a number of products, I was beginning to believe that a mere programmer had no chance whatever!

"I am not only making good progress in producing the Help for our product (which is usually used by ****very**** inexperienced users), but I'm actually enjoying it!

"Thanks for a great product."

Try the Free Evaluation Version Today

HelpScribble is an excellent help authoring tool. But do not take my word for it! Take a look at the [self-running online demos](#) or download the [free evaluation version](#) and see for yourself.

Convinced? [Buy HelpScribble now!](#)