vars plugin for PowerPro: version .87: 7 May 2009

A POWERPRO PLUGIN TO SAVE SCRIPT VARIABLES

Alan Campbell

1.0 Overview

This PowerPro plugin allows script variables to be saved to an ini file, and recovered from it, allowing variable values to persist between invocations of a script.

You won’t find this plugin makes much sense if you're not a reasonably advanced script writer. You must for instance understand what it means to declare a variable global, static and local, and you might want to know a bit about maps and vectors.

1.1 This Document

There are three versions of this document, with the same content. There's an RTF file, which looks nice in Word but is something like four meg in size; and there's a compiled help (CHM) document, which is much smaller if somewhat uglier; and there's a pdf, with bookmarks for each section heading.

In my experiments I've found the RTF file doesn't display correctly in anything but Word (not Keynote, even Wordpad: you've think Microsoft could at least get their rtf engines consistent). So if you don’t have Word, better use the chm file.

All documents have extensive hyperlinks. The table of contents at the front of each document is a set of them.

The chm file has no index.

1.2 What’s New In This Version

- fixed a bug meaning ini config file not found on machines running non-English version of windows
- added varsPluginFunctions.txt to be used as a file menu, perhaps merged with pprofunctions.txt.
- added a pdf version of documentation

1.3 Terminology

In order to describe how the vars plugin works, I need some vocabulary that hasn’t been provided in the PowerPro documentation.

Variables can (and should) be declared storage class (local, static, or global) (see the PowerPro help section "Programming scripts with if, jump, variables, flags"). I'll refer to these as the possible ***storage class****es* a variable may belong to.

Variables are normally a way of storing a single value. I'll call these simple variable ***scalars***.

Variables may also name maps or vectors (created and manipulated using services of the map and vec plugins). I'll call such variables ***collections***.

A single script file may contain many procedures, each starting with a @label and ending with a quit statement. I'll call each of those executable sections a ***procedure***.

1.4 Uses

- You could use this plugin to save variables that need to persist between runs of PowerPro. I use it to retrieve a set of global variables (often treated as read-only constants) in my startup script.
- If you had variables that needed to be used in several scripts, you could use the vars plugin to restore variables at the beginning of each script and save them when they finished. You could achieve the same result a number of other ways:
  - Use global variables. That might be undesirable, as the variables would then be visible to all scripts, not just the few that needed them.
  - Use several labelled subscripts beginning with a @label and ending with quit, in the same file, and declare the shared variables static. Static variables appear to hold their values for any subscript within a file.

2.0 Requirements

PowerPro version 3.4 or later. Test scripts require at least 3.8.15. Edit them to get rid of ?c…c syntax to use with previous versions. The scripts work in standard configuration.

Those of you who are tetchy about MFC will be happy to know that, unlike the registry plugins, vars.dll *doesn't* require MFC support (e.g. MFC42.DLL).

The test script uses the ini plugin.

Tested on W2000 sp2 and NT4 sp3.

2.1 Related plugins

Since this plugin saves variables in ini files, you might want to do clever things directly to the variable-saving ini files with the ini plugin.

This plugin uses the map and vec plugins to manipulate..guess what…maps and vectors.

2.2 Reporting Bugs, Requesting Enhancements

This plugin is complicated. I have by no means tested every possible path through the code, so bugs are likely.

If you hit any problems with this plugin (or any of my plugins, for that matter), it'd be helpful if you reported them via the PowerPro forum (http://groups.yahoo.com/group/power-pro/) in a message with a clear subject line (maybe: "VARS PLUGIN: apparent error in:…."). I don't read everything in the forum, but I will see anything flagged with an obvious header. Please include a copy of the script causing problems, and state which version of PowerPro and of the vars plugin you're using.

3.0 File list

plugins\vars.dll
docs\varsPluginReadme.rtf
docs\varsPluginReadme.chm
docs\varsPluginFunctions.txt
scripts\varsPluginTestScript.powerpro
scripts\allProcs.ini
scripts\allProcsGlobalsPerProc.ini
scripts\varsPluginTestScript.ini
scripts\allProcsStaticsPerFile.ini
scripts\varsPluginTestScriptOutputProvided.log
scripts\vars.ini

4.0 Installation

Copy vars.dll from varsPluginX.XX.zip archive into your PowerPro directory, or into its Plugins subfolder. If you want to provide an initial configuration of the plugin (see Section 9.1), edit vars.ini and put it in the folder pointed to by pprofolder; or add its edited contents to plugins.ini in the same folder.

The **.**rtf documentation, **.**powerpro script and **.**log sample output can go wherever you want. If you want the test script varsPluginTestScript.powerpro (which demonstrates vars.save and vars.restore services) to work correctly, create a vars subfolder of <pprofolder>\scripts and put allProcs.ini, allProcsGlobalsPerProc.ini, varsPluginTestScript.ini, and allProcsStaticsPerFile.ini in it.

varsPluginFunctions.txt can be appended to pprofunctions.txt supplied in the PowerPro distro (or included in it, using

include <*path to*>\comPluginFunctions.txt)

It can then be accessed either as part of pprofunctions.txt or on its own as a file menu (e.g. using a hot key associated with

*keys {filemenu <*path to*>\pproFunctions.txt})

5.0 Uninstall

Remove all files listed in the above section ("3.0 File list") from wherever they went.

6.0 Acknowledgements

Idea from SGP, who tested intermediate versions.

7.0 Testing

The test script uses the ini plugin.

If you want to use the enclosed script, “varsPluginTestScript.powerpro” to exercise the plugin:

Then, either:

- Put varsPluginTestScript.powerpro in your <PowerPro configuration>\scripts directory

- Run varsPluginTestScript.powerpro with a command (menu item?) in PowerPro like

  *Script RunFile varsPluginTestScript.powerpro *or*
  .varsPluginTestScript

Or:

- Put varsPluginTestScript.powerpro anywhere and double click on it (as long as PowerPro is already running). This didn’t work on my machine until I manually set up an association with .powerpro, but if your PowerPro installation went correctly it should work for you.

varsPluginTestScript.powerpro will output both to the Debug window and a log file. By default that file is “varsPluginTestOutput.log” in same directory as your pcf file. You can edit the script to point it another path/file if you wish: it’s the first line of code in the script, and assigns a path/file name to the variable **vars_test_logfile**. If you assign the null string to **vars_test_logfile**, no log file will be created.

varsPluginTestScript.powerpro overwrites values in the [varsConfig] section of vars.ini if it exists, or plugins.ini settings if it doesn't. It will however attempt to restore whatever values it finds there (if any). If neither vars.ini or plugins.ini exist, it will create plugins.ini..

The varsPluginTestScript.powerpro script doesn’t use the evaluate-expression operator "&" so is not dependent on your choice for it. It uses the ?c…c syntax to avoid problems with your declared escape character, so you should have no problems whether that's ' or \.

There are further comments on how the varsPluginTestScript.powerpro script works embedded in the script itself.

The log file generated by varsPluginTestScript.powerpro can be compared with “varsPluginTestScriptOutputProvided.log” that comes with the distribution.

8.0 List of Services And General Notes on Usage

Ensure vars.dll is in your PowerPro installation directory, or in the plugins subfolder thereof.

There are numerous services in this plugin. They are

service	description	section
save_var	save one script variable to ini file	10.1
restore_var	recover one variable from ini file	10.2
save	save script variables to ini file	10.3
restore	recover variables from ini file	10.4
chars_to_hex_encode	specify which characters should be saved in \xNN format	10.5
error_dialog_on **error_dialog_off**	turn PowerPro script error dialog on and off	10.6
config	set location of configuration ini file	10.7
version	returns the plugin version number as four digit number, last two to be taken as right of decimal.	
unload	remove plugin from memory	

restore_var, restore, save_var and save always declare and use the variable **_t2mpn0g.**

These are described below, in section 10 and its subsections.

9.0 Writing scripts using the vars plugin

Call the appropriate plugin service as follows:

retval = **vars.service(**arguments if any**)**

Results, if any, are generally available as the returned value from the above expression, though there are alternatives**.**

After you are finished using the vars plugin in your script, you can if you wish unload it with

vars.unload

It's probably best done if you do not foresee the plugin being used again for a while.

All services return "OK" if all goes well, or a string beginning "ERROR: " if not.

There are services which affect the behaviour of the vars plugin (see sections 10.5 – 10.6). You can also customise the behaviour of plugin services by providing a configuration ini file (see section 9.1).

If you do an *Exec ChangeConfiguration, and the new and old pcf files are in different folders, you should unload this plugin before using any of its services with the new configuration. (only necessary if you use relative paths to specify the location of ini files).

9.1 The Configuration ini File

On the first call to any vars service, the plugin checks for a file called vars.ini in the folder pointed to by the PowerPro variable pprofolder (usually the folder in which the currently active pcf file is found). It then looks for a [varsConfig] section in that ini file. If that's not found, it looks for the same section in the file plugins.ini in the same folder. If either are found, it looks for the following values in the section:

key	possible values	default value	can also change using service	meaning
staticsSavedOncePerScript	as above	1	none	see Sec. 11.1
reqDeclarationBeforeRestore	as above	1	restore second param	see Sec. 10.2.1
varsBackupLocation	path to file/folder	scripts/vars	none	see Sec 11.1
varsBackupGlobalsLocation	file or null	null string	none	see Sec. 11.1
allowDupVariableDeclares	0 1 y n t f*	1	none	allow same variable with different storage classes. See below.
charsToHexEncode	any	+1-1F	chars_to_hex_encode	determines which characters are saved in \xNN format See Section 10.5 for details.
raiseErrors	0 1 y n t f*	1	error_dialog_on error_dialog_off	determines if errors in syntax format, service arguments etc cause powerpro to raise the script error dialog.

* Only the first non-whitespace character of the key's value is checked, no matter how many there are. So values "yes", "no", "true" or "false" will work. As usual the ini value is case-insensitive, so "Y" and "FALSE" are also valid.

† Only the first non-whitespace character of the key's value is checked, so "status", "data" and "none" will work.

If no vars.ini or plugins.ini files are found with a [varsConfig] section, the vars plugin will initially just use the compiled-in, default values (specified in the third column above) to configure itself.

If an ini file with a [varsConfig] section is processed, and there's a possible value that could be in ini but isn't, the vars plugin reverts to the default value for that value.

Once an ini file with a [varsConfig] section is processed, its values stay in force until the vars plugin unloaded or the config service is run.

If the first service called is one which itself changes the configuration (e.g. error_dialog_on), the configuration ini file will be found and evaluated *before* the service is applied.

If you want to change your initial configuration, you can use the ini plugin to do it. E.g.:

```
ini.set(pprofolder ++ "plugins.ini", "varsConfig",
        staticsSavedOncePerScript", "no")
```

After making your changes, either:

- unload the vars plugin. The next time you use one of its services, the new configuration will kick in. Or

- run vars.config**(<path to configuration ini file>)**.

If you do an *Exec ChangeConfiguration; and if the new and old pcf files are in different folders; and if there's a vars.ini or plugins.ini file in the new pprofolder, the configuration it specifies won’t take effect until you take one of the steps above..

allowDupVariableDeclares determines if variables restored or saved (and therefore named in the ini file) are allowed to have duplicate declarations, i.e. declared as different storage classes. If false, the following situations will generate error messages:

variable in ini file	not allowed to also have a declaration as
globals	static
static	global
local	static, global

10.0 The Services

Variables are saved to an ini file; either

- the one named in the last argument of the service call (which, if not an absolute path, is taken to be relative to the folder specified in pprofolder);

- or, if that argument is absent, the ini file determined by values in the vars.ini/plugins.ini configuration file: see below (Section 11.1: "Where Variables Are Saved to and Restored from: the ini Files")

The easiest services to use are save_var and restore_var. I suggest you try them first, and try them using only the required parameter (the name of a variable to save or restore).

For all services, if you want to control exactly where variables are stored, and how globals, statics and collections are stored, you need to read Section 11 and its subsections carefully.

10.0.1 The <*tag*> Argument

All services that save and restore variables have an optional <*tag*> argument. If it's not the final argument in a service's argument list, and you wish not to specify it, use the null string: a null <*tag*> argument means no <*tag*> argument.

Variables are saved separately for each combination of <*tag*> and script file. The plugin knows what script file you're invoking the service from: the <*tag*> is an arbitrary string you provide as an argument. The <*tag*> argument may be necessary because you might wish to:

- save a distinct set of variables for each section of code in a script file callable via a @label.

  I can't within my plugin determine the name, if any, of the @label by which a script was invoked. Hence <*tag*>. A good convention might be to always use the @label as the <*tag*>. Like this:

  ```
  @myLabel

  local myLocal
  static st_myStatic

  vars.restore("myLabel")
  .
  .
  vars.save("myLabel")
  quit
  ```

- save a distinct set of static variables for different states of a script file. Each state would be represented by a different <*tag*>: all procedures in the file would use the values of the static variables saved and restored under various <*tag*>s, and could therefore be regarded as "in a different state". This will only work if the key staticsSavedOncePerScript is 0, "n" or "no" in your configuration ini file.

If a script file has no labelled subsections, and if you have no need to save separate sets of scalar variable values, you might choose to invoke vars.save and vars.restore without a <*tag*> argument (or a null string in its place).

Details of specific services follow.

10.1 save_var

vars.save_var(<*variable_name*> [, <*tag*> [, <*ini_file*>]])

vars.save_var will save one variable in a script to an ini file. You can restore the saved variable using either vars.restore (Section 10.4) or vars.restore_var (see next section).

The <*variable_name*> parameter is not the variable itself; it's the name of the variable. So if you want to save a variable iX, call will be, at least

vars.save_var("iX")

not

vars.save_var(iX)

If an explicit <*ini_file*> isn't specified in the service call, the ini file determined according to the rules specified above in Section 11.1 ("Where Variables Are Saved to and Restored from: the ini Files") will be used.

If the variable <*variable_name*> does not exist at the point in a script at which **vars.save_var**(<*variable_name*>…) is invoked, the service will trigger an error message dialog or return an error message (see Section 10.6 to find out which).

See Section 10.0.1 regarding the <*tag*> parameter.

Variables containing floating point numbers will appear as gibberish in their ini file representation, but it's meaningful gibberish, which will get turned back into the correct number when you restore.

10.2 restore_var

vars.restore_var(<*variable_name*> [,<*tag*> [, <*recreate_variable*>
[, <*ini_file*>]]])

restore_var will restore one variable to a script, retrieved from an ini file.

<*variable_name*> is not the variable itself; it's the name of the variable: e.g. if you want to restore a variable iX, the service call will be, at least:

vars.restore_var("iX")

not

vars.restore_var(iX)

The variables is recovered from an ini file; either

- the one named in the fourth argument of the service call (which, if not an absolute path, is taken to be relative to the folder specified in pprofolder);
- or, if that argument is absent, the ini file determined by values in the vars.ini/plugins.ini configuration file: See Section 11.1: “Where Variables Are Saved to and Restored from: the ini Files”.

If the appropriate ini section/value(s) that defines the value of the variable <*variable_name*> doesn’t exist at the point in a script at which **vars.restore_var**(<*variable_name*>…) is invoked, the service will trigger an error message dialog or return an error message (see Section 10.6 to find out which).

See Section 10.0.1 regarding the <*tag*> parameter.

If you use a <*tag*>, it must match one you used when you called vars.save or vars.save_var.

10.2.1 The *<recreate_variable>* Parameter

The key **reqDeclarationBeforeRestore** in the vars.ini/plugins.ini configuration file (which defaults to 1 if not defined) and the second argument of the vars.restore_var service (<*recreate_variable*>s: defaults to 0 if not present) determine if variables must be declared (and collections created) before they are restored. **vars** will require variables be declared before restored if:

- **reqDeclarationBeforeRestore** is 1 and there is no second argument to vars.restore.
- there is a non-zero second argument to vars.restore, regardless of the value of **reqDeclarationBeforeRestore.**

Personally I think it might always be good practice to explicitly declare variables and create collections. Otherwise readers of your script (including you, in a year's time) will have to refer to the variable-storing ini file to see what variable does what.

If you cause the vars.restore service to declare variables, it will declare each to be of the storage class (as defined in the ini file) and create collections using the parameters stored there (see Section 11.5 "Storing Collections", below).

10.2.2 Restoring a Collection: Some Notes

When a collection is restored, I've assumed you want the vec or map to have all but *only* the values stored in the ini file. A static and global collection may have elements added by other scripts and procedures. So I have to clear any collection before restoring it. In Power-pro versions earlier than 4.1.03 the only way to do that is to destroy it and then create it again; if that's done the create parameters defined for each collection's value in the ini file (see Section 11.5 "Storing Collections" below) will be used. (In Power-pro versions 4.1.03 or later I can use the deleteall service to clear out a map or vec).

I'll assume you will not populate a local collection before you invoke restore, so a local collection will not be wiped before being restored.

10.3 save

> **vars.save**()
> **vars.save**([*<tag>* [, *<create_sections>* [, *<add_statics>* [, *<add_globals>* [, *<ini_file>*]]]]])

vars.save will save variables in a script to an ini file. You can restore the saved variables using vars.restore (see next section).

Unlike vars.save_var**,** **vars.save** will save *all* the variables with entries in an ini file – either

- the one named in the fifth argument of the service call (which, if not an absolute path, is taken to be relative to the folder specified in pprofolder);
- or, if that argument is absent, the ini file determined by values in the vars.ini/plugins.ini configuration file: See Section 11.1: “Where Variables Are Saved to and Restored from: the ini Files”.

See the Section 10.0.1 regarding the *<tag>* parameter.

10.3.1 Setting up an ini File: the Parameters *<create_sections>*, *<add_statics>*, *<add_globals>*

The second argument of **vars.save** (*<create_sections>*) if omitted is assumed zero. If non-zero, e.g.

> **vars.save**("myProc", 1)
> **vars.save**("", 1)

vars.save will generate the standard section names for scalars of each storage class and for collections, as dictated by the rules set by the configuration ini file keys **varsBackupLocation**, **varsBackupGlobalsLocation** and **staticsSavedOncePerScript** (see below, this section and one following).

You can then manually add the names of all the variable you want saved, with any or no value e.g.

```
myVar1=
myVar2=dummy
```

Or you can call **vars.save** with non-zero third (*<add_statics>*) and/or fourth (*<add_globals>*) arguments. Their permissible values are:

If argument is	then
omitted or 0	no variables of the relevant storage class are added.
1	*all* variables of the relevant storage class (including scalars, maps and vecs), defined at the point at which **vars.save** is called, will be added to the ini file
2	a PowerPro picklist will popup repeatedly offering a list of • all variables of the relevant storage class (including scalars, maps and vecs), • defined at the point **vars.save** has been called, • that have not yet been added to the ini file. To select a variable, click on it and click Ok, hit <enter>, or double click. The variable selected will be added to the ini file. Clicking Ok or hitting <enter> with no variable selected will select the first item on the list. To stop picking variable of the relevant storage class, click cancel or hit <esc>.
3	(Only legal if you have PowerPro 4.1.04 or later): As 2, but you can select multiple variables in the list by holding down <ctrl> or <shift> keys. Be careful of clicking Ok, or hitting <enter> with no variable selected: that will select all the variables in the list.

The plugin cannot detect declared local variables: you have to add them manually.

In my experience an <*add_statics*> of 1 (adding all statics) and an <*add_globals*> is 2 (use a picklist) is quite a good combination, e.g.

```
vars.save("myLabelledProc", 1, 1, 2)
vars.save("", 1, 1, 2 )
```

YMMV.

Note that when variable names are added to an ini file they will always be in lower case (an artefact of how PowerPro records variable names): you may wish to edit variable name case in the ini file for clarity. Also note that if the plugin finds a static or global variable already entered in an ini file, it will not overwrite the value assigned to the variable.

If you only want to save a few variables, you might find that a few calls to vars.save_var(var_name) (see Section 10.1) might be the simplest thing option.

10.4 restore

vars.restore([*<tag>* [, <*recreate_variable*>s [, <*ini_file*>]]])

vars.restore reloads variables that were saved with vars.save.

You can't call **vars.restore** if you haven't previously called first created the ini file that stores variable values.

Unlike vars.restore_var**,** **vars.restore** will recover *all* the variables with entries in an ini file – either

- the one named in the third argument of the service call (which, if not an absolute path, is taken to be relative to the folder specified in pprofolder);
- or, if that argument is absent, the ini file determined by values in the vars.ini/plugins.ini configuration file: See Section 11.1: "Where Variables Are Saved to and Restored from: the ini Files".

Only variables named in the relevant sections of the ini file will be restored.

See Section 10.0.1 regarding the <*tag*> parameter.

If you use a <*tag*>, it must match one you used when you called vars.save or vars.save_var.

See Section 10.2.1 regarding the <*recreate_variable*> parameter.

See Section 10.2.2 ("Restoring a Collection…") if you want to do that thing..

10.5 chars_to_hex_encode()

vars.chars_to_hex_encode(<*character_selection*>)

Use this service to specify which characters should be saved in \xNN format. Initially which characters are encoded is determined by the vars.ini/plugins.ini key charsToHexEncode; if that's missing only characters under x20 encoded.

The format for the <*character_selection*> parameter and the charsToHexEncode key is:

<character_selection> := **rule**s separated by **whitespace**

rule := either a **+** or a **-**, followed by either a **range_rule** or a **char_as_hex**

range_rule := a **char_as_hex** followed immediately by a hyphen followed immediately by by a **char_as_hex**

char_as_hex := two **hex_digit**s

hex_digits := one of 0123456789ABCDEF

whitespace := any number of (but at least one) spaces and tabs

If a **rule** begins with a **+**, the following range or single character will be encoded in \xNN format. If the rule begins with a **-**, the following range or single character will be saved "as is".

So for instance the <*character_selection*> "-02-10 -80 +C0-FF" means: "save any character with a value in the range x02 to x10 as is; save x80 as is; save all characters with values above xC0 in \xNN format".

If a <*character_selection*> omits characters, those omitted characters continue to be dealt with on saves as determined by the default behaviour; the charsToHexEncode key; and any previous calls to **chars_to_hex_encode**.

If a <*character_selection*> includes characters, the rule for those characters is the same as the one already in force, that's not an error; the rule is just left in force.

10.6 error_dialog_on(), error_dialog_off()

Some vars services can result in errors. For instance, you might try to restore from a non-existent ini file, or a variable might be defined in an ini file that wasn't declared, and should have been. In addition to such errors setting a PowerPro variable or returning a value with a status message the prefixed "ERROR:" (see previous section); they will also trigger the standard PowerPro script error dialog, allowing you to cancel all running scripts.

Under some conditions you might not want vars plugin errors to be treated as scripting errors, and you would therefore not want to see the PowerPro script error dialog. If that's what you want (maybe because you're testing for the presence or absence of a section/key pair), invoke **vars.error_dialog_off()**, or make sure the **raiseErrors** key in the configuration ini file is false or 0. Invoke **vars.error_dialog_on()** to turn error dialogs back on after you turn them off.

If you unload the plugin, it's behaviour returns to the default in any subsequent call, i.e. errors on further vars service will cause the error dialog to pop up.

Invoking **vars.error_dialog_off()** only affects error dialogs appearing when a vars service call goes wrong. The normal PowerPro error dialog will appear if anything else goes wrong in PowerPro.

10.7 config

vars.config(<*name_of_ini_file*>)

specifies a configuration ini file, with format, section and keys as described in section 9.1

The ini file can either be given as an absolute path, or a path relative to the folder returned by the pprofolder variable (generally the folder containing the currently running powerpro configuration file).

Returns "OK" if file and found and there are no keys with illegal values, or a message beginning "ERROR:" if there is one.

If you unload and reload a plugin, it's behaviour returns to the default or to that defined by a default config ini file (see section 9.1).

.

11.0 save, restore, save_var, restore_var: The Details

The basic form of the save, restore, save_var and restore_var services are described above. But there's a lot of ugly detail if you want to stray away from the simple life.

11.1 Where Variables Are Saved to and Restored from: the ini Files

If you're unfamiliar with how ini files work, see Appendix I: "The Structure of ini Files".

Variables will be saved to, and restored from, a location affected by the configuration ini file keys **varsBackupLocation**, **varsBackupGlobalsLocation** and **staticsSavedOncePerScript**.

You can also explicitly override the rules outlined below by providing an explicit name and location as the last ("*<ini_file>*") parameter of the save or restore services. See below.

varsBackupLocation can point either to a folder, or an ini file, in either case either absolute or relative to the location pointer to be pprofolder (generally the folder in which your running pcf file is located). If you don't set it, it defaults to <pprofolder>\scripts\vars. If the folder or ini file is specified with a relative path, it will be taken as relative to the folder specified by pprofolder.

If **varsBackupLocation** is a folder, the implication is that you want to back up variables that originate in each script file to an ini file with the same name as that file, located in the folder you specified. So if **varsBackupLocation** is "scripts\backup", and you execute vars.save from the script burble.powerpro, variables will be saved to <pprofolder>\scripts\backup\burble.ini.

If **varsBackupLocation** is an ini file, the implication is that you want to back up variables from *all* scripts to the same file. Variables originating from separate script files will be distinguished by section names that incorporate the associated script file/*<tag>*.

If you wish to have values of global variable stored in one file and one file only (and therefore avoiding the somewhat dubious idea of a global variable having many different values storied for many different scripts), set the **varsBackupGlobalsLocation** key to the name and location of an ini file. If the ini file is specified with a relative path, it will be taken as relative to the folder specified by pprofolder.

If **varsBackupLocation** is an ini file, **varsBackupGlobalsLocation** can point to the same file.

11.2 Explicit ini file parameters

If you specify an *<ini_file>* as a non-null-string final parameter to any vars service**,** that file will be used instead of that determined by **varsBackupLocation.** If the ini file is specified with a relative path, it will be taken as relative to the folder specified by pprofolder.

If you specify an *<ini_file>* as a non-null-string fifth parameter , note that even if the configuration ini file key **varsBackupGlobalsLocation** points to an ini file, global values will *not* be stored there, but in the ini file you name as a parameter to vars.save.

11.3 ini File sections

Regardless which ini file is used to save variables, for each script file + *<tag>* combination there will be separate sections for:

- local scalar variables
- static scalar variables
- global scalar variables
- the names of all non-scalar variables (i.e. vectors and maps)
- for each non-scalar variable, a section recording its values

Sections names are moderately complicated. See Appendix II: "Section Names in save/restore ini Files". Whatever you do, do *not* either modify or delete section names, unless you've really genned up on how they work and want to roll your own (e.g. to manage a change of configuration (see Section 11.7: "Caveats").

Now the important bit: restore will obviously only restore variables (named as keys in the appropriate section) it finds in the ini file. However, less obviously, save is also driven by the ini file contents: *only those variables that already have a key in an appropriate section of the ini file* will be saved (thus obviously overwriting the value of the key in question).

11.4 Storing Global and Static Scalars

One possible interpretation of what a global variable means suggests it makes some kind of sense to save values of global variables just once, in just one place, regardless of which script they're being saved from. If you want to do that, specify a non-null value for **varsBackupGlobalsLocation,** which must be a path to an ini file.

varsBackupGlobalsLocation and **varsBackupLocation** can point to the same ini file (which would be a common arrangement if you wanted all variable information from all scripts stored in one place).

Note that **varsBackupGlobalsLocation** is ignored if you specify an explicit ini file as the last parameter of a service call: in that case all variables, globals or otherwise, will be stored in the ini file explicitly specified by the parameter.

Or, you might chose to have **varsBackupLocation** point to a folder (implying separate ini files for each script and **varsBackupGlobalsLocation point** to the file in which globals are stored.

Even if you choose to storing the values of globals in just one location, you still have to specify the names of those globals you want to save and restore in the correct section for each script-name/proc-id (ending -global-scalars-]).

varsBackupGlobalsLocation is taken as null if you don't specify it, i.e. by default global values will be stored per script-name/proc-id, just like static and global variables.

staticsSavedOncePerScript (a Boolean) determines whether statics are stored once for an entire script file or separately for every *<tag>* within that file.

11.5 Storing Collections

Variables that are collections (vectors or maps) have to be handled differently. Once you've generated sections for a particular script/<tag>, there'll be a section (ending -collections-] in which to name all maps and vector, regardless of storage class (local, static, or global). Just add the names of the collection variables to that section, as:

```
[myScript-myProc1-collections-]
myVector1=blah
```

When you next run vars.save without a second argument, two things will change. A new section will be added to the ini file in which the values of that collection are stored. And the entry in the list of collection variables will have as value the name of that new section. E.g.

```
[myScript-myProc1-collections-]
myVector1=myScript-myProc1-local-vec-myVector1 10
....
[myScript-myProc1-local-vec-myVector1]
0=abra
1=ka
2=dab
3=ra
```

vars.save also puts a number after the entry in the list of collection variables. It's the detected size for the collection: vector.capacity or map.length (map.capacity in version 4.1.03 or later). It will be used by the plugin to recreate the vec or map if required, i.e. it will become an argument to vec.create or map.create. You can modify that size, and you can add any other arguments after it, separated by commas, that are valid arguments for the relevant create service. For maps, that would be *useCase*; for vecs: *growth, minsize*. See the relevant plugin documentation for details.

The entire string found after the first blank character in the collection's key's value will be taken as the set of arguments to the create command for map or vec.

So for instance you might edit the above to tell the plugin restore service that the vector myVector1 should be created with initial size 20, and should be allowed to grow:

```
[myScript-myProc1-collections-]
myVector1=myScript-myProc1-local-vec-myVector1 20,1
```

When a collection is saved, the relevant ini section is deleted (to eliminate collection elements that no longer exist) and recreated. Therefore it will appear to move to the end of the ini file (where the win ini-manipulating API always appends new sections).

11.6 How Special Characters are Stored

Strings may include stuff that can't directly be saved to an ini file with about a bit of messing about.

- **\n, \r, \t, \\:** are stored as the appropriate byte as part of variable strings. If you output a newline or carriage return character to a text file, you get a text string spanning several lines, which won’t work in an ini file. So the bytes corresponding to \n, \r and \t are converted back to "\n", "\r" and "\t" before a string is written to an ini file by vars.save, and converted back again when read in by vars.restore.

  If you want to use a backslash before n, r, x or t without it being treated as an escape character, use \\, then the letter.

  A backslash before anything but n, r, t, x or \ have no special significance and are just treated as literal backslashes.

- **\x:** \xNN is treated as the character corresponding to the hex digits NN. There must be exactly two hex characters after \x

  When saving a string any character with a hex value less than 0x20 will be saved in \xNN format. You can override the riule determining which chacters are saved as \xNN using either the vars.ini/plugins.ini key charsToHexEncode or the chars_to_hex_encode service.

- **leading and trailing white space**: is normally trimmed off by the API commands I use to write and read ini files. Workaround is to enclose a string in quotes; any blanks within the quotes string are preserved. So you'll see all saved strings in an ini file generated by vars.save are quoted. The quotes are stripped off by vars.restore.

11.7 Caveats

You can mess things up nicely in all kinds of amusing ways.

In PowerPro you are allowed variables with different storage classes with the same name, e.g. in a given procedure the variable blah could by defined as static, and also already have been declared as global somewhere else. The vars plugin may not behave as expected in such circumstances. In particular, when you are saving a variable it may be detected as global but the value actually stored will be the current static or local value.

Avoid duplicate variable names: They are bad scripting practice anyway.

Don't delete or change ini section names in the files used to store variable values. You have been warned.

If you change a variable's name or storage class in a script, you have of course to modify any and all ini file sections that refer to that variable. If you delete a variable from a script, ditto.

If you change the name of a script file, you will have to change the names of all sections of the ini file that are used by that file.

If you try to backup and restore variables from two scripts with the same name (have to be in different folders, won't they?), you can only do it without collision if you either

- use disjoint sets of <*tag*>s in the separate files, or
- always use the explicit-ini-file-name argument every time you call save and restore from at least all-but-one of the identically-named script files.

If you change the vars.ini/plugins.ini configuration keys that govern save/restore, you'll have to modify the structure of the variable backup ini files. See Appendix III: "How to Reorganise Backup ini Files If You Change Configuration".

11.8 Where Variables Are Saved from and Restored to: the Scripts

A single script file may contain many procedures, each starting with a label and ending with a quit statement.

Variables in a procedure can be declared local, static or global, or have previously been declared global. Two variables with the same name can, in any procedure, be both global and static, or global and local. Best to avoid such multiple definitions if you can: I haven't tested every possible combination of multiple definition and save/restore service calls, but its possible some such might not work right.

Normally variables can only be restored into a script in which all variables to be restored have been declared (but see the next section for an exception); so on entry to a procedure, first declare everything static, global or local, and create any vectors or maps that have been saved; then invoke vars.restore. Then do your stuff, then invoke vars.save with the same <*tag*> (if any) that you used when you restored.

12.0 Change History

- fixed a bug meaning ini config file not found on machines running non-English version of windows
- added varsPluginFunctions.txt to be used as a file menu, perhaps merged with pprofunctions.txt.

0.84: 26 July 2007

- Fixed a bug that resulted in failure to save large numbers of global or static variables (with a symptom of complaining about an undeclared variable, one with a truncated name of a genuine variable name).
- the local variable **_t2mpn0g** is now used any time you call any save or restore service.

0.80: 30 November 2006

- fixed a bug which left the local variable _t2mpn0g undeclared before use in some circumstances
- added version service.
- Changed, hopefully improved documentation. Among other things the arguments I called <*proc_id*>s in previous versions of docs I now call <*tag*>s, and are explained more thoroughly.

0.78: 5 October 2005

- Fixed error reporting bug in vars.save

0.77: 25 August 2005

- Fixed handling of map keys containing \n

0.75: 25 August 2005

- In saved strings, added handling for the \xNN (hex coded characters) and now allow the escape character \ to itself be escaped (so \\ is a literal \). See section 11.6.
- Added vars.ini/plugins.ini key charsToHexEncode and associated service chars_to_hex_encode

0.70: 30 May 2005

- Added hyperlinks to documentation, and added doc in HTML help format; tidied up and corrected documentation.

0.63: 9 April 2005

- fixed a bug in detection of globals and statics in save_var service.

0.62: 7 April 2005

- added save_var and restore_var services (see Sections 10.1, 10.2)
- changed rules regarding integration of the vars.ini/config.ini key **varsBackupGlobalsLocation** when there's an explicit *<ini_file>* parameter to a service call: see Section 11.1.
- Fixed a bug affecting use of explicit ini file as last argument to restore service.

0.60: 27 February 2005

- Fixed some bugs

0.57: 15 December 2004

- picklist code used when you use the save service and want to pick existing statics or globals now allows multiple selection if you're using a version PowerPro 4.1.04 or later.

0.56: 13 December 2004

- Fixed bugs in vars.save when creating new ini file.

0.55: 12 December 2004

- Fixed a few bugs in the picklist code used when you use the save service and want to pick existing statics or globals to add to a (newish) ini file. Now no limit on number of globals, and globals added will disappear from the picklist.

- When you use the save service and want to pick existing statics or globals to add to a (newish) ini file, the picklists will now exclude relevant variables already in the ini file
- New var.ini/plugins.ini configuration key **allowDupVariableDeclares**; if false reports multiple declaration of same variable as error.

0.50: 9 December 2004

- first version

Appendix I: The Structure of ini Files

For those of you unfamiliar with ini files.

An ini file must have this internal structure:

```
        [section name]
        key_name= value
        another key_name= another value
        ; a disabled key name= an old value
( ... followed by more sections containing more name=value lines ... )
```

- Any lines starting with a semicolon are regarded as comments and cannot be read
  or written to by this plugin.
- Section names and key names are not case sensitive and may contain spaces.
- Section names must be unique.
- Key names within each section must be unique.

Appendix II: Section Names in save/restore ini Files

In the table, italic text is variable, non-italic fixed for category

In section names, hyphens are used to prevent collision between set section names and those (holding collection values) that include the name of a (map or vector) variable.

In the next table, prefix is:	
when all variables stored in a single ini file:	*script_name*-*<tag>*
when variables for each script file held in a separate ini file:	*<tag>*

Section Name Formats				
to record				**section name becomes**
type	**storage class**	**names or values**	**location**	
scalar	local	names values		[*prefix*-local-scalars-]
scalar	static	names		[*prefix*-static-scalars-]
scalar	static	values	stored per <*tag*>	[*prefix*-static-scalars-]
scalar	static	values	stored per file	[static-scalars-]
scalar	global	names		[*prefix*-global-scalars-]
scalar	global	values	stored per <*tag*>	[*prefix*-global-scalars-]
scalar	global	values	stored in one place	[-global-scalars-]
collection	local static global	names		[*prefix*-collections-]
collection	local	values		[*prefix*-local-vec-*vector_name*] [*prefix*-local-map-*map_name*]
collection	static	values	stored per <*tag*>	[*prefix*-static-vec-*vector_name*] [*prefix*-static-map-*map_name*]
collection	static	values	stored per file	[*script_name*-static-vec-*vector_name*] [*script_name*-static-map-*map_name*]
collection	global	values	stored per <*tag*>	[*prefix*-global-vec-*vector_name*] [*prefix*-global-map-*map_name*]
collection	global	values	stored in one place	[-global-vec-*vector_name*] [-global-map-*map_name*]

Appendix III: How to Reorganise Backup ini Files If You Change Configuration

if ini config key	is changed:	you must
varsBackupLocation	from folder to ini file	merge the individual, one-per script ini files, first prefixing each section name with script_file-
	from ini file to folder	separate sections relating to each script file into separate files,, first removing script_file- from each section name
varsBackupGlobalsLocation	null string to ini file	move [<file_name>-<tag>-global-scalars-] sections to ini file for globals and rename [-global-scalars-] move [<file_name>-<tag>-global-<coll_type>-<coll_name>] sections to ini file for globals and rename [-global-<coll_type>-<coll_name>]
	ini file to null string	reverse above procedure
staticsSavedOncePerScript	0 to 1	rename [<file_name>-<tag>-static-scalars-] sections in each ini file to [<file_name>-static-scalars-] rename [<file_name>-<tag>-static-<coll_type>-<collection name>] sections to [<file_name>-<coll_type>-<collection name>]
	1 to 0	reverse above procedure