unicode plugin for PowerPro: version .69 17 April 2009

A PLUGIN TO ALLOW USE OF UNICODE TEXT IN POWERPRO

Alan Campbell

No warranty of any kind, express or implied, is included with this software; use at your own risk Responsibility for damages (if any) to anyone resulting from the use of this software rests entirely with the user.

1.0 Overview

This PowerPro plugin allows you to manipulate unicode strings. You can get them in and out of the clipboard and in and out of files. You can use all the functions available in the file plugin, but the filenames can be unicode. There are equivalents of most of the string functions available in PowerPro for manipulating them.

What you *can't* do (and will never be able to do, unless Bruce rebuilds PowerPro for unicode) is use unicode strings in PowerPro bars, menus or dialogs.

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 bug in inputDialog service
- fixed bug in keys service (which crashed if *raw* parameter specified)
- added pdf version of documentation
- added version resource
- added a destroy alias for the release service
- added unicodePluginFunctions.txt to be used as a file menu, perhaps merged with pprofunctions.txt.

2.0 Requirements

Requires 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. They work in standard configuration.

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

The support utility encodeUnicode.exe (see Section 8.3) requires riched20.dll and may not work on win9x machines.

The to_binary service requires the binary plugin. You should have at least version 0.46. Get it here:
http://groups.yahoo.com/group/power-pro/files/Plug-ins_and_add-ons/0_Funny_Strings/

There are currently two variants of the binary plugin: one whose services need the name of a variable as an argument, so they can directly access the variable buffer's contents, and one that works with blocks of binary data via handles. The unicode plugin can use either. The latter is preferred.

Tested on W2000 sp4. On W2K it would appear that if you haven't specifically installed regional support for whatever unicode variant you want to use, you may not be able to rename files or manipulate shortcuts using the relevant unicode services. But if you're always using unicode, you probably have whatever regional support you need already installed.

Unlike W2K, XP doesn't seem to require specific regional support to be installed for unicode file names to work no matter what characters are used.

I'm unclear how much unicode support, if any, is available on win 9x machines. I found a note on a Microsoft site that said "…unicode support on Windows Me/98/95 requires Microsoft Layer for Unicode." If you're on such an OS and need unicode, you probably know more than I do.

2.1 Reporting Bugs, Requesting Enhancements

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: "UNICODE 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 unicode plugin you're using.

3.0 File list

plugins\unicode.dll
docs\unicodePluginReadme.rtf
docs\unicodePluginReadme.chm
docs\unicodePluginFunctions.txt

scripts\unicodePluginGeneralDemo.powerpro
scripts\unicodePluginStringDemo.powerpro
scripts\unicodePluginStringDemoDotSyntax.powerpro
scripts\unicodePluginClipboardDemo.powerpro
scripts\unicodePluginFileDemo.powerpro

scripts\unicodePluginKeysDemo.powerpro
scripts\keystrokes.txt

scripts\testAllFiles.powerpro
scripts\testWatchFolder.powerpro
scripts\testUnicode.lnk
scripts\testNotUnicode.txt
scripts\testUnicode.txt

scripts\encodeUnicode.powerpro

4.0 Installation

Copy unicode.dll from unicodePluginX.XX.zip archive into your PowerPro directory, or into its plugins subfolder.

All other files can go wherever you want, including .powerpro scripts.

unicodePluginFunctions.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

Thanks to Bruce for source code for his plugins and built-in Powerpro functions.

Ta to Andreas Mokros for help debugging UTF-8-oriented services, and to Sean () for suggestions and help testing the keys service.

Stole code from:

Rich Edit Control DOES NOT support UNICODE string? at:

http://www.msusenet.com/history/topic.php/1870572589-1.html

BASE 64 Decoding and Encoding Class 2003 at

http://www.codeguru.com/Cpp/Cpp/algorithms/article.php/c5099/

7.0 Testing

To run any of the unicodePlugin*.powerpro scripts:

Either:

- Put them in your <PowerPro configuration>\scripts directory
- Run scripts with a command (menu item?) in PowerPro like

  *Script RunFile unicodePluginTestScriptToSelf *or* .unicodePluginTestScript

Or:

- Put the scripts 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.

The scripts will output both to the Powerpro debug window and numerous messagebox dialogs (there being no other way to present unicode strings from within Powerpro).

The debug window is easier to understand if you check "Put debug window items at end of list box" in PowerPro's configuration, Advanced Setup.

None of the test scripts use the evaluate-expression operator "&" so are 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 scripts work embedded in the scripts itself.

Note that

- **testAllFiles.powerpro** and **testWatchFolder.powerpro** are not meant to be run on their own: they're used by unicodePluginFileDemo.powerpro.
- **encodeUnicode.powerpro** is primarily a helper script, but can be used as a test as well.
- **unicodePluginStringDemoDotSyntax.powerpro** illustrates the handle.service syntax introduced with PowerPro 4.4.07.
- **unicodePluginKeysDemo.powerpro** exercises the keys service. keystrokes.txt must be in the same folder as the script. You must be prepared to switch to an application that can accept unicode characters as input, e.g. a unicode-aware text editor. Word seems to know what to do with unicode on my system.

8.0 Usage

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

8.1 Handles and Unicode String Storage

Since Powerpro can't itself store and remember unicode strings, unicode plugin services that produce unicode strings return not the string itself, but a *handle* to one. The string itself is stored internally, within the plugin A handle is just a simple string beginning "u\x05" followed by a number from 3000 to 3255.

If you have PowerPro 4.4.07 or later, instead of the standard syntax:

```
local uString1 = unicode.new("aaa")
local uString2 = unicode.new("bbb")
local uString3 = unicode.new("ccc")
local uString = unicode.join(uString1, uString2, uString3)
```

you can use the syntax

```
local uString = uString1.join(uString2, uString3)
```

This only works if the first argument of the service (in old-style syntax) is a handle to a unicode string.

This would fail:

```
local sString3 = "ccc"
local uString = sString3.join(uString2, uString3)
```

Furthermore, if you have PowerPro 4.4.11 or later, any handles created as intermediates in an expression (and not, therefore assigned to a variable or returned via quit) will be automatically released after the expression's evaluated, so this is fine:

```
uString = unicode.decode("EAQRBBIEEwQUBAAA")
unicode.messagebox("ok", uString.fill("zz"), "Cyrillic 3 chars + zz")
```

uString.fill("zz") in the second line allocates a handle to a Unicode string -- but that handle is automatically destroyed once the messagebox clears.

The plugin can store (and provide handles for) up to 256 unicode strings.

Because there's a limit, you may wish to use the release or releaseall services to let go unicode strings you no longer need.

Don't lose handles to unicode strings by e.g. overwriting it without first releasing the underlying string it points to. This is a bad idea.

```
local rcStruct = unicode.new("test")
rcStruct  = ""  ;; oops, lost the handle
```

So is this: unless you have PowerPro version 4.04.11 or later:

```
local sResult = unicode.join("clipboard is:" , unicode.clip_get)
```

If you have the later version, the handle from **unicode.clip_get** will be used, but immediately released once the expression it's part of is evaluated.

If you have earlier versions, the handle returned by **unicode.clip_get**, though it will work in context, will be lost after the expression is evaluated, so there'll be no way to release the unicode string.

If you have PowerPro version 4.4.05 or later, you don't have to use release on handles stored in local variables; they'll be released automatically when the local goes out of scope at the end of the routine: unless you override that automatic release by using the localcopy service.

For instance, in this context (assuming arg(1) carries a handle to a unicode string):

```
@aRoutine
local v
v=arg(1)
```

you probably don't want the unicode string destroyed when you routine exits, because, usually, the caller may wish to do more stuff with it. To prevent the automatic destruction of handles held in local variables, do this instead:

```
local v
v=dll.localcopy(arg(1))
```

Scripting tip: if you're going to use this facility, *don't* unload the unicode plugin at the end of your script. PowerPro will need the plugin dll loaded in memory to do it's thing with local handles.

If you do release a handle held in a local variable, best invalidate it by e.g. doing

```
myLocalVar = unicode.release(myLocalVar)
```

Unloading the unicode plugin will cause all stored unicode strings to be released.

unicode.file_open returns handles to files, which are incompatible with those returned by the standard PowerPro file plugin. You can use the filehandle.service syntax as you can with standard file plugin handles.

8.2 Making Unicode Strings

You can produce a unicode string (and get its handle) by

- explicitly creating it using new
- combining unicode and/or ordinary strings using join or append.
- copying an existing unicode string using copy
- applying a transformation to a unicode or ordinary string
- retrieving text from clipboard or file
- input it as an encoded literal:

8.3 How to Input Literal Unicode

You can't just type literal unicode into a Powerpro script, since Powerpro only understands ASCII. So, there's a support script (which uses the inputDialog and encode services): encodeUnicode.powerpro. Crank that up, type in your unicode string, hit the right button, and an encoded version of your unicode (base64, since you ask), is put on the clipboard. Paste that text in between quotes as an argument to unicode.decode, which will give you back a handle to a unicode string.

8.4 Using Unicode Strings

Okay, so the only thing Powerpro itself knows about unicode strings is their handles. It doesn't know how to use those handles (only the plugin does), so you can't meaningfully use a handle to a unicode string as an argument to a Powerpro function. In particular, that means you can't use unicode strings to set text in Powerpro bars or menus, or as arguments to other plugins.

The only way you can display a unicode string is to use unicode.messagebox. The debug window can't handle them. If you send a variable containing a handle to a unicode string to the debug window, all you'll see is something like "UNS__3213".

8.5 The Configuration Ini File

On the first call to any **unicode** service, the plugin checks for a file called unicode.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 [unicodeConfig] 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 modify using service	meaning
indexBase	0, 1	0	none	whether indexing starts at 0 or 1 get_bytes, set_bytes,
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 unicode.ini or plugin.ini files are found with a [unicodeConfig] section, the unicode 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 [unicodeConfig] section is processed, and there's a possible value that could be in ini but isn't, the **unicode plugin** reverts to the default value for that value.

Once an ini file with a [unicodeConfig] section is processed, its values stay in force until the **unicode 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", "unicodeConfig", "raiseErrors", "1")

After making your changes, either:

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

8.6 The Services

There are a lotta services in this plugin::

- services with no direct equivalent in Powerpro: to do with creation, destruction and conversion of unicode strings.
- clones of built-in Powerpro string functions**:** Services in this category usually have the same name as the Powerpro functions. The few exceptions are those where the built-in function name conflicts with a standard windows API function, which are altered to avoid the conflict: e.g. case, strcoll, stricol.

  To understand how a service works, go to

  winhlp32.exe -I stringfunc <path to powerpro.hlp>

  and look up the equivalent powerpro function, and read that in conjunction with the remarks against the service in question in the table below.

- clones of built-in Powerpro UI functions: just one, messagebox. See

  winhlp32.exe -I messagebox <path to powerpro.hlp>

- clones of clip plugin services**:** take name "clip_" plus the name of the clip plugin service.
  To understand how a service works, look up the equivalent clip service in clip.txt in the plugins folder and, and read that in conjunction with the remarks against the service in question in the table below.

  clones of file plugin services**:** take name "file_" plus the name of the file plugin service.

  To understand how a service works, look up the equivalent clip service in file.txt in the plugins folder, and read that in conjunction with the remarks against the service in question in the table below.

  clones on win plugin services: sendcopydata and keys

  file_open and file_writeall have a final, optional parameter that will cause the unicode byte-order marks (\xFF\xFE) to be prefixed to the file. If the parameter is missing, BOM is written.

The services are listed in the following tables. Arguments are listed for each service. The types for those arguments are described in the next section. There's a column in each table to indicate whether arguments are optional. If they are, the default value for that argument used if the argument is left out, is indicated in parenthesis.

In the tables, the **service** column gives the name of the service and, *in italics*, any aliases for the same service. Aliases only work if you're using <handle>.alias syntax.

8.6.1 Service Argument Types

Each argument of each service has a type. The allowed argument types are:

type	which is
int	an integer, either as a literal or held in a variable
bool	an integer, but expected to be 1 or 0.
str	a normal powerpro string, either as a literal or held in a variable. The only arguments to unicode services that must be simple ASCII strings are things like the *keyword* argument of **change_case**, the *mode* argument of **file_open**, the *info* argument of **file_resolve**, or the *layout* argument of **messagebox**
us	a representation of a unicode (wide) string. This can be either: a handle to a wide string: must have been returned by a previous call to a unicode service and not yet released; recognised because meets criteria for a handle (which begins "UNS__" followed by a number from 3000 to 3255). Almost invariable held in a variable. a normal Powerpro string: If above criterion not met, assumed to be a plain ASCII string [1]
h_us	a handle to a wide string, which must have been returned by a previous call to a unicode service and not yet released.
enc	an encoded string, generated by the encode serice or the support utility encodeUnicode.exe. See Section 8.3. It's a base64-encoded version of a unicode string. Only used as an argument to unicode.decode.
fh	a file handle, returned by unicode.file_open.. Handles returned by the file plugin's file.open will not work.

[1] These will be converted to temporary unicode strings, which will be used as the real argument to the service, then, if not an argument to new, deleted.

Services without direct equivalents in Powerpro

service	arguments	type	req ?	returns	Powerpro equivalent	remarks
new	string	str	Y	h_us	*none*	makes a unicode string from a plain string
empty	size	int	Y	h_us	*none*	returns a handle to a WCHAR buffer able to accept up to *size* characters
decode	encoded	str (encoded)	Y	h_us	*none*	decodes *encoded* from base64. See Section 8.3
encode	target	h_us	Y	str	*none*	encodes *target* to base64. See Section 8.3
join	string1 string2 …	us us ...	Y N N	h_us	++	1-8 arguments joins unicode strings. details.
append	string1 string2 …	h_us us ...	Y N N	h_us	*none*	1-8 arguments appends string(s) onto another. details. modifies what the *string1* h_us points to and returns that handle
to_ascii *ascii*	string codepage	h_us int	Y N	str	*none*	convert unicode string to ascii. Uses default ANSI code page unless you provide *codepage*
to_binary	see details	see details	Y	details	*none*	convert a unicode string to a byteblock, usable by the binary plugin. details
from_utf8	string	str	Y	h_us	*none*	create a unicode string from UTF-8 encoded string. details.
from_mbcs	string	mbcs	Y	h_us	*none*	create a unicode string from MBCS encoded string
to_utf8	string	us	Y	str	*none*	create a UTF-8 string from a stored unicode string
release *destroy*	target	h_us	Y	-	*none*	release a handle
release_all					*none*	release all handles
localcopy	target	h_us	Y	h_us	*none*	makes a copy of a handle (typically passed into a routine as arg(n))

Services without direct equivalents in Powerpro						
service	arguments			returns	Powerpro equivalent	remarks
		type	req?			
						and copied to a local); explanation here.
version	-	-	-	int	*none*	the plugin version number is returned as four digits, an assumed decimal point before the last two.
set_base	base	int (0 or 1)	Y	-	*none*	sets base for get_bytes, set_bytes and some other services. Overrides config ini key indexBase
get_base	-			0 or 1	*none*	returns current base for indices
compare	string1 string2	us us	Y Y	int	< > ==	case insensitive compare

join, **append**: if one arg, just makes a new unicode string, unless it's a handle to a unicode string, in which case handle is simply copied

create_from_utf8: UTF8 if for the most part representable as an ANSI string. To the extent that's true, you can use this service. But if for instance your UTF-8 includes the NUL character (\x00), you're stuffed, because you can't enter that as part of a PowerPro string.

to_binary: arguments depend on which variant of the binary plugin you have. If it's the one that directly manipulates variables, provide the name of the variable as the first argument, a handle to the unicode string as the second. If it's the one that returns handles to binary strings, provide the handle to the unicode string as its one argument; **to_binary** will return a handle a binary string.

If you want to write a script in which **to_binary** will work for either variant of the binary plugin (only necessary if you want to write scripts for general distribution, not just for your own use), you need to test the last character returned by **binary.version**; that will be "h" for the variant that returns handles, and "v" for the variant that works directly with variable contents.

Equivalents of Powerpro functions that do UI things

service	arguments			returns	Powerpro equivalent	remarks
		type	req ?			
messagebox	layout message title	str us us	Y Y N	int	messagebox	*layout* can only be a plain (non-unicode) string.
inputDialog	format title useBOM	str us int (1/0)	Y Y N	int (1/0)	inputDialog	see below

inputDialog is complicated and hedged around by caveats.

- **format**: A plain (non-unicode) string made up of *variable=title* pairs. See PowerPro help for the details. the variables may contain plain ANSI strings, or handles to unicode strings. But:
  - if a *variable=title* pair is terminated by "??" and therefore defines a checkbox, the variable must *not* be a handle to a unicode string (since the only thing relevant about the variable's value is whether it's non-zero or not).
  - if a *variable=title* pair is terminated by "??" followed by "value|value2|value3…" and therefore defines a combobox, the values may contain plain ANSI strings, or handles to a unicode strings; but if the latter, they may not display correctly.[*]
- **title**: may contain a plain ANSI string, or a handle to a unicode string
- **useBOM**: if you're using a combobox, and your language needs the unicode byte order mark (BOM) in front of each string added, set this to 1.

[*] http://www.harper.no/valery/PermaLink,guid,99b85fa3-104f-4a41-a28f-4786c68e77e4.aspx said: "One interesting (undocumented) thing that I found during test of Russian version is that [the] ComboBox [control]requires UNICODE BOM when adding UNICODE strings to them. Chinese, Japanese and Persian work without BOM, but Cyrillic doesn't – just show bricks instead of letters – weird…..if you don't have installed files for complex scripts, right-to-left languages or East Asian languages, then BOM is treated as invalid UNICODE character by ComboBox and you'll see something like "■" in front of every line"

Equivalents of Powerpro functions that manipulate strings

service	arguments			returns	Powerpro equivalent	remarks
		type	req ?			
length	target	h_us	Y	int	length	returns length of *target*
find (0-based) index (1-based)	searched target	us us	Y Y	int	index	returns index of of *target* within *searched*.
revfind (0-based) revindex (1-based)	searched target	us us	Y Y	int	revindex	returns index of last occurrence of *target* within *searched*
repeat	target count	us int	Y Y	h_us	repeat	returns unicode string formed by repeating first argument *count* times
remove	target count	us int	Y Y	h_us	remove	removes *count* characters from *target*
fill	target fill	us us	Y Y	h_us	fill	combines *target* and *fill* see Powerpro help re fill function
replacechars	target patt/from to	us us us	Y Y N	h_us	replacechars	see Powerpro help re replacechars function
translate	target from to	us us us	Y Y Y	h_us	translate	see Powerpro help re translate function

Equivalents of Powerpro functions that manipulate strings (continued)

service	arguments			returns	Powerpro equivalent	remarks
		type	req ?			
minimum *str_min* *min*	string1 string2	us us	Y Y	h_us	min	returns handle to minimum of two values; if the argument which is minimum is a literal, causes a new unicode string to be stored and handle to be generated
maximum *str_max* *max*	string1 string2	us us	Y Y	h_us	max	maximum of two values; see min
env	variable	us	Y	h_us	env	returns value of environment variable named by *variable*.
change_case *case*	target keyword	us str	Y Y	h_us	case	see Powerpro help re case function
slice (0-based) str_select, *select* (1-based)	target pos endpos	us int int	Y Y N	h_us	select	selects *pos* characters from *target*; or select *pos* through *endpos* characters. see Powerpro help re select function.
str_coll *strcoll* *coll*	string1 string2	us us	Y Y	int: 1: ws1 <ws2 0: ws1 =ws2 -1: ws1 <ws2	strcoll	compares two unicode strings using local collating sequence
str_icoll *stricoll* *icoll*	string1 string2	us us	Y Y	int: 1: ws1 <ws2 0: ws1 =ws2 -1: ws1 <ws2	stricoll	compares 2 unicode strings using local collating sequence, ignoring case of letters;
line	string	us	Y	h_us or int	line	returns the nth line, unless *n* == 0, in which case returns

Equivalents of Powerpro functions that manipulate strings (continued)

service	arguments			returns	Powerpro equivalent	remarks
		type	**req?**			
	n	int	Y			line count.
word	string n delimiters	us int us	Y Y N	h_us or int	word	returns the nth word, unless *n* == 0, in which case returns word count. *delimiters* default to space and tab if omitted
get_chars *get_char*	string pos posEnd type	h_us int int str	Y Y N N	h_us, str or int	assign from hu[pos, ep]	gets *pos*th to *posEnd*th characters in a unicode string; you can also use target[*pos*] syntax details
set_chars *set_char*	string pos posEnd *or* type value	h_us int str us str	Y Y Y N N	h_us supplied as first arg	assign to hu[pos, ep]	modifies an existing unicode string by setting *pos*th to *posEnd*th characters to *value*. You can also use *string*[*pos*] = *value* syntax. Always returns the *string* handle. details
default_get_set_type	type	str	Y	-	-	determines behaviour of set_chars, get_chars with no type arg details
squeeze	target runsOf	h_us h_us	Y N	h_us	squeeze	see Powerpro help re squeeze function
reverse	target	h_us	Y	h_us	reverse	see Powerpro help re reverse function
rotate	target n	h_us	Y Y	h_us	rotate	see Powerpro help re rotate function

Equivalents of Powerpro functions that manipulate strings (continued)

service	arguments			returns	Powerpro equivalent	remarks
		type	req ?			
	pre	int	N			
	min	usint	N			

get_chars: *pos* and *endPos* are based at 1 or 0 (depending on value of configuration ini key indexBase or last call to **set_base**; assumed value if absent depends on same); must be no greater than the length of the unicode string (determined by its length); if negative chooses a position at end of string (-1 is the last unicode character). If *endPos* is absent, one unicode character is retrieved. If *endPos* is present, unicode characters from *pos* to *endPos* are retrieved. *type* can begin with "u" to return a handle to a unicode string (the default if *type* is absent); "c" (character), in which case unicode characters are converted to ANSII; or "8" (number) in which case you get back the UTF-8 equivalent of the selected unicode characters.

You may wish to call **default_get_set_type** which will set the behaviour of get_chars (and set_chars) without a type argument from then on, until **default_get_set_type** is called again or the plugin is unloaded.

set_chars: *pos* and *endPos* are as for **get_chars**. The selected characters are replaced by *value*. *type* can only be present if *endPos* is absent (i.e. if setting a single character); if it begins with "u" (unicode), or is absent, *value* can be either a handle to a unicode string or a plain string; or "n" (number) in which the numeric value (0 to 65280 aka 0xFF) is used to set specified unicode character (which only works if you've selected exactly one byte with *pos* and possibly *endPos*). **default_get_set_type** also affects behaviour of set_chars if it's invoked without a *type* argument.

Equivalents of clip plugin services

service	arguments			returns	equivalent service	remarks
		type	req?			
clip_get	*(none)*			h_us	get	returns the text on the clipboard
clip_set	string cap	us bool	Y N (0)	-	set	sets the clipboard; if *cap* is present and set to 1, then this clip will be captured by PowerPro
clip_append	*(none)*	us	Y	-	append	appends text to the clipboard
clip_setpaste	*(none)*	us	Y	-	setpaste	sets text to the clipboard, then sends Ctrl-V to paste to foreground window
clip_copy	*(none)*				copy	see note
clip_cut	*(none)*				cut	see note
clip_paste	*(none)*				paste	see note
clip_clear	*(none)*			-	clear	see note
clip_length	*(none)*			int	length	returns length of clipboard treated as unicode string

clip_copy, **clip_cut**, **clip_paste**, **clip_clear**: all these are exactly the same as equivalent clip plugin service, so having them here is redundant. However, they're only a few lines of code apiece,

Equivalents of clip plugin services (continued)						
service	arguments	type	req?	returns	equivalent service	remarks
clip_fromFile	path	us	Y	-	fromFile	copies file at *path* to clipboard details.
clip_fromFileAppend	path	us	Y	-	fromFileAppend	appends file at *path* to clipboard does not support rich text format.
clip_toFile	path noErr prefix	us bool bool	Y N (0) N (1)	-	toFile	puts clipboard into file at *path* no error message if noErr is 1. if *prefix* is 1 or absent, the unicode BOM \xFF\xFE will be prefixed to file
clip_toFileAppend	path noErr	us int	Y N (0)	-	toFileAppend	appends clipboard to file at *path* no error message if noErr is 1 does not support rich text format.

clip_fromFile: if file extension is .clprtf, Bruce's code in clip.fromFile does weird stuff..
My code does exactly the same weird stuff, and makes no attempt to interpret the file contents as unicode.
If prefix not '0' or absent, the unicode byte-order mark (\xFF\xFE) is prefixed before clipboard contents

Equivalents of file plugin services						
service	arguments			returns	equivalent service	remarks
		type	req?			
file_open	fname mode format bom	us str str bool	Y Y N (u) N (1)	fh	open	opens a file from a unicode_string name and returns a file handle used subsequently to access the file. See note.
set_utf8_default	format	"a" or "u"	Y	nothing	none	whether the *format*s utf8, utf-8 and 8 are the same as utf8a or utf8u; see note.
file_readstring *readstring*	file	fh	Y	h_us	readstring	reads and returns next line (format determined by *format* specified in file_open which produced file handle) from file; the trailing \r and \n are removed; strips off leading unicode byte-order marks if present. See note
file_readline *readline*	file	fh	Y	h_us	readline	reads and returns next line (format determined by *format* specified in file_open which produced file handle) from file including any trailing \r or \n ; strips off leading unicode byte-order marks if present. See note
file_writeline *writeline*	file string	fh us/str	Y Y	0/1	writeline	writes a *string* (format determined by *format* specified in file_open which produced file handle) and following \r and \n

Equivalents of file plugin services (continued)						
service	arguments			returns	equivalent service	remarks
		type	req?			
file_writestring *writestring*	file string	fh us/str	Y Y	0/1	writestring	writes *string* (format determined by *format* specified in file_open which produced file handle) to file, but does not output an \r or \n
file_readall	path format	us str	Y N (u)	h_us if u/8 str if a	readall	reads entire file at *path* name into a single string; strips off leading unicode byte-order marks if present. See note
file_writeall	path string append format bom	us us/str str str bool	Y Y N N (u) N (1)	-	writeall	Writes a string to a file. See note.
file_setmaxline	max	int	Y	-	setmaxline	set maximum line buffer for subsequent calls to file_readline, file_readline; if n <264, 264 is used. If n>64k, 64k is used.
file_close *close*	file	fh	Y	-	close	closes the file and frees the file handle
file_closeall	*(none)*			-	closeall	closes all open file handles

Equivalents of file plugin services (continued)

service	arguments			returns	equivalent service	remarks
		type	req?			
file_eof *eof*	file	fh	Y	int	eof	returns 0 if last read did not encounter eof; non-zero otherwise
file_restart *restart*	file	fh	Y	-	restart	restarts reading or writing of a file from the beginning
file_delete	path	us	Y	-	delete	deletes a file; the *path* can contain wildcards.
file_deletenorecycle	path	us	Y	-	deletenorecycle	deletes without sending file to recycle bin the *path* can contain wildcards.
file_copy	path1 path2 noError	us us bool	Y Y N (0)	1 if success, 0 if not	copy	copies a file to a destination file or folder. See note
file_move	path1 path2 noError	us us bool	Y Y N (0)	1 if success, 0 if not	move	moves (renames) a file from *path1* to a destination file or folder specified by *path2*. See note.
file_isfolder	path	us	Y	int	isfolder	checks to see if *path* is a valid path to a folder the path may *not* contain wildcards
file_attrib	path	us	Y	str	attrib	returns a string of single letters indicating the *path*'s attributes.
file_type	path	us	Y	h_us	type	returns the type (extension) of *path* excluding the leading period
file_name	path	us	Y	h_us	name	returns the file name of *path*, excluding the folder and type
file_nametype	path	us	Y	h_us	nametype	returns the name and type from *path*

Equivalents of file plugin services (continued)

service	arguments			returns	equivalent service	remarks
		type	req?			
file_folder	path	us	Y	h_us	folder	returns the folder from *path* without the final \
file_size	path	us	Y	str	size	returns file size in bytes of file at *path*
file_size64	path	us	Y	str	size64	returns size as 64 bit integer; use int64 plugin to work with
file_ksize	path	us	Y	str	ksize	returns file size in kilobytes (size/1024)
file_lastmodified	path	us	Y	str	lastmodified	returns the last modified date and time of the file as a string of 12 digits : yyyymmddhhmmss.
file_getshortpath	path	us	Y		getshortpath	returns short path to file at *path*
file_setdate	path type date time	us str int int	Y N N N	-	setdate	sets date of a file at *path*. see note re *type* parameter for more details see file.txt doc re file.setdate
file_getdate	path type	us str	Y N	str	getdate	gets date of a file at *path*. see note re *type* parameter for more details see file.txt doc re file.getdate
file_watchfolder	path cmd maxwait keywords	h_us str int str	Y N N N		watchfolder	see note. for more details see file.txt doc re file.watchfolder
file_runwait	maxwait exePath params workDir howstart	int us us us str	Y Y N N N	int	runwait	for details see file.txt doc re file.runwait
file_runaswait	maxwait exePath params	int us us	Y Y N	int	runaswait	for details see file.txt doc re file.runaswait

Equivalents of file plugin services (continued)

service	arguments			returns	equivalent service	remarks
		type	req?			
file_runcallback	workDir howstart maxwait cmd params workDir howstart	us str int str us us us str	N N Y Y Y N N N	-	runcallback	suggest you don't execute unicode.unload in the *cmd* script. Powerpro may not like it. for details see file.txt doc re file.runcallback
file_allfiles	path cmd bSubfolders bPump bUseHidden	us str bool int bool	Y N N (0) N (0) N (0)	int	allfiles	see note. for more details see file.txt doc re file.allfiles
file_listfiles	path bSubfolders bPump	us bool int	Y N (0) N (0)	str	listfiles	see note for details see file.txt doc re file.listfiles
file_version	path	us	Y	str	version	returns file version info as four blank separated numbers
file_runas	username password app args start_folder	us us us us us	Y Y Y N N	1/0	runas	runs *app* as *username* logging on with *password*. *username* may be a local user, or may be of form user@domain. *app* runs with *username*'s registry profile and environment loaded. returns 1 if succeeds, 0 if fails
file_createshortcut **not yet working**	target fpathlnk desc	us us us	Y Y N	1/0	createshortcut	creates a .lnk file shortcut to another file for details see file.txt doc re file.createshortcut service

Equivalents of file plugin services (continued)						
service	arguments			returns	equivalent service	remarks
		type	req?			
	params iconpath iconindex	us us int	N N N			
file_resolve **not yet working**	path info	us str	Y N		resolve	resolves shortcut at *path* and returns specified information for details see file.txt doc re file.resolve

Equivalents of win plugin services						
service	arguments			returns	Powerpro equivalent	remarks
		type	req ?			
sendcopydata	string target dwData	us captionlist int	Y Y N	1/0	win. sendcopydata	sends string held in *string* to target window using WM_COPYDATA details.
keys	string raw	us bool (0/1)	Y N	-	win.keys, *Keys	sends keys. details.

sendcopydata: Haven't tested, I don't ASAIK an application that responds to WM_COPYDATA messages containing unicode strings.

keys: All the {}codes defined for the *Keys command work, with the exception of:

- obsolete codes: {cmdsep}, {param}, {clip}, {var}
- method (speed) codes: {fast}, {slow} and {sinp}. In the current version of unicode.keys, only SendInput is used; I don't think any other method is possible.
- {to *folder*} and {filemenu}.

These omitted {} codes will be accepted as valid input to unicode.keys, but will be silently ignored.

The sticky modifiers{alt}, {at}, {shift}, {sh}, {ctrl}, {co}, {win} and {wi} are implemented, as are % (for Alt), ^ (for Ctrl) and + (for Shift); however in your application they may have odd or no effects when combined with unicode characters.

{file *path* [, file_type]} sends the contents of a file at *path* as keystrokes; use the same file format as documented for *keys (including the rules for comments). file_type isn't required if the file is a UTF-16 file.

The *raw* parameter, if absent, assumed 0; if present and 1, string is sent without parsing the special characters {}%^+

Notes on file services:

file_open: Minimally, you may wish only to use file_open to open a file whose path or file name needs to be specified in unicode.

mode may be "r" (opens the file to be read); "w" (opens the file to be written; will be overwritten if it exists); or "a" (opens the file to have new information written after existing information). If omitted, "r" is assumed.

format (case-insensitive) parameter specifies the format of file contents, and determines what the *bom* (case-insensitive) parameter means:

this word	or this letter	means format is	read ops return	***bom* = 1 or absent**		***bom* = 0**	
				mode = "w"	**mode = "r"**	**mode = "w"**	**mode = "r"**
utf16 utf-16 unicode	u	UTF-16	h_us	start file with \xFF\xFE	discard leading \xFF\xFE if present	no BOM added	no BOM discarded
utf8 utf-8	8	UTF-8	h_us *	start file with \xEF\xBB\xBF	discard leading \xEF\xBB\xBF if present	no BOM added	no BOM discarded
utf8u utf-8u	8u	UTF-8	h_us				
utf8a utf-8a	8a	UTF-8	str				
mbcs	m	ANSI multibyte character stream	h_us				
ascii	a	plain ascii	str	*bom* ignored			
text	t	plain ascii	plain ascii				

You should set *bom* to zero if the ultimate consumer(s) of the unicode file you are writing won't recognise a byte-order mark, or if you know the unicode file you wish to read has no byte-order mark.

* By default, utf8 is the same as utf8u. You can change that option with the **set_utf8_default** service, which takes either "a" ("utf" then means "return read results from UTF-8 files as ascii strings" i.e. same as utf8a) or "u" ("utf" then means "return read results as handles to utf strings" i.e. same as utf8u) as an option. The option remains in force until you unload the plugin or call **set_utf8_default** again.

file_readstring, **file_readline**: The maximum line length starts at 4K ***bytes*** (therefore 2K unicode characters) but can be set as large as 64K bytes with unicode_file.setmaxline.

If the file_open used to obtain the handle of the file being read was called with format UTF-8, UTF-16 or MBCS, an h_us return will be returned; if ascii format was specified, a plain string will returned

file_readall: content taken to be UTF-16, unless *format* specified. *format* words and letters are as for file_open. What gets returns as for **file_readstring**, **file_readline** (see above).

file_writeall: writes the provided string to a file in UTF-16, unless *format* specified. *format* words and letters are as for file_open. The file is created at *path.*

If *append* is "a", "A" or 1, appends rather than creating new file.

If *bom* is 1 or absent and *format* is ‘u’, the BOM \xFF\xFE is prefixed to file before *string.*
If *bom* is 1 or absent and *format* is ‘8’, the BOM \xEF\xBB\xBF is prefixed to file before *string.*

file_copy, file_move: The second path can be either a folder name or a full file name. If third argument is present and starts with digit "1", no error messages will be produced.

file_setdate, **file_getdate** *type* parameter: the date set or fetched is:

last modified if *type* omitted or *type* = "m" or "M"

created if *type* = "c" or "C"

file_allfiles: For each file found in folder, _file_ is set to a h_us which can be used in the script *cmd*. The script *cmd* or subsequent scripts must release that handle. The bar (|) syntax, available in file.allfiles, can't be used in this service.

file_watchfolder: When there's a relevant change in the watched folder, _file_ is set to an h_us, and *cmd*(,) executes with same h_us as argument. The script *cmd* or subsequent scripts must release that handle. I suggest you don't execute unicode.unload in the *cmd* script. Powerpro may not like it.

file_listfiles: returns a string which is list of *handles* to all the file names (treated as unicode strings) in the folder given by *path*. This won't work if there more than 256 files to get handles for, because of the plugin limit on the number of handles that can be held. After calling **file_listfiles** you should release all the handles returned (which you can recover using the line function: nb not unicode.line, just the plain PowerPro line function).

9.0 Restrictions

10.0 Using unicode Plugin from Other Plugins

The unicode.dll exports various functions that can be used to support unicode data in other plugins.

You get at those functions via a PPUNICODESERVICES which you can either copy from here:

```
struct PPUNICODESERVICES
{
 BOOL      (* bFreeSpace)        ();
 WCHAR*    (* get)          (IN LPSTR);
 BOOL      (* add)          (IN WCHAR*, LPSTR, PPROSERVICES*);
 BOOL      (* remove)       (IN LPSTR);
 BOOL      (* modify)       (IN WCHAR*, LPSTR, PPROSERVICES*);
 BOOL      (* addmake)      (IN WCHAR*, LPSTR, PPROSERVICES*);
 INT       (* getsize)      (IN LPSTR);
};
```

or get from handleCollection.h, which is in the unicode plugin archive under the folder callingFromOtherPlugins, along with a sample application.

10.1 Exported Functions

These five functions are exported from unicode.dll for use in other plugins, as members of the UNICODESERVICES struct:

BOOL bFreeSpace()

Call this function to see if there'sfree space in the handle collection held by the unicode plugin.

WCHAR* get(IN LPSTR pszHandle)

Call this function to use retrieve a Unicode string from handle *pszHandle.*

Returns the NULL pointer if the handle is invalid.

BOOL add(IN WCHAR* pwszString, OUT LPSTR pszHandle, IN PPROSERVICES* ppsv)

Call this function to add a unicode string to the collection held by the unicode plugin.

pwszString: the unicode string to add.

pszHandle: handle returned if unicode string added successfully.

ppsv: the PPROSERVICES* pointer passed into every service

Returns 1/0 if successful/failure. *pwszString* is assumed to be on the heap and will be deallocated by a call to free().

BOOL remove(IN LPSTR pszHandle)

Call this function to remove a Unicode string with previously issued handle *pszHandle.*

BOOL modify(IN WCHAR* pwszString, INOUT LPSTR pszHandle, IN PPROSERVICES* ppsv);

Call this function to modify an existing Unicode string with previously issued handle *pszHandle.*

If *pszHandle* refers to the null Unicode string, a new handle is issued and stored in *pszHandle*.

If the existing unicode string pointed to by *pszHandle* is the same size or larger than *pwszString*, it's copied into the existing string buffer. Otherwise *pwszString* is duplicated on the heap and that duplicate becomes the stored string buffer.

BOOL addmake(IN WCHAR* pwszString, OUT LPSTR pszHandle, IN PPROSERVICES* ppsv)

As add, above, but *pwszString* is not assumed to be on the heap; it's duplicated on the heap, and the result added, and deallocated by a call to free().

INT getsize(IN LPSTR pszHandle)

gets size of buffer in wide characters (usually same size as unicode string) associated with previously issued handle *pszHandle.* unicode.empty() creates an empty buffer of the specified length (plus one).

You access all the above via an exported function

struct PPUNICODESERVICES* getUnicodeServices()

which you can use to get a pointer to the PPUNICODESERVICES struct you need to do everything else.

10.2 Using the Exported Functions

Two choices: static or dynamic linking

If you want to link statically, you need handleCollection.h from the callingFromOtherPlugins folder of the unicode plugin archive, and in VC++ you need to link with unicode.lib. The bad news: unicode.dll will have to be on your path, i.e. the folder <PowerPro Installation folder>\plugins will have to be added to the PATH environmental variable.

Probably easier to add a bit of code to your plugin DllMain that finds unicode.dll, sets up a function pointer to getUnicodeServices(), then calls it to get a pointer to the PPUNICODESERVICES struct.

You can just copy relevant code from samplePlugin.c in sampleDependentPlugin.zip, which has sample code and projects to make a simple plugin that uses unicode.dll. See the next section.

10.3 Sample Plugin Using unicode.dll Functions

samplePlugin.zip in the unicode distribution includes the following files.

samplePlugin.c
PowerProDefines.h: you must include these two files to compile the sample plugin. The source file has the code necessary to dynamically load and link to the functions listed above.

samplePlugin.dsp
samplePlugin.dsw: A VC6++ project including above header and source file. You can probably get it to run in VC5++ just by editing the workspace file, and in VC7++, which should convert the files to its own format.

samplePlugin.dev
Makefile.win: A Dev-Cpp/mingw project file and generated mingw make file.

sample.powerpro: You have to move the dll generated by one of the above projects to your plugins folder before you run this script.

The enclosed projects will generate a plugin, sampleUnicodeClientPlugin.dll in the project folder. If you want to use a debugger, you'll have to alter the project to create the dll in the <PowerProFolder>\plugins folder.

11.0 Possible Enhancements

It would be very easy to increase the maximum number of unicode strings that can be dealt with at once.

Might be able to fix file_createshortcut and file_resolve, which don't work at the moment. Working on it.

Might be able to provide hooks from other plugins (ini? reg? net? pop?) into this one, so you could perhaps read unicode out on an ini file or specify a url to one of the the net plugin webpage services. Let me know via forum if there's any demand.

12.0 Change History

.69:

- fixed bug in inputDialog service
- fixed bug in keys service (which crashed if *raw* parameter specified)
- added pdf version of documentation
- added version resource
- added a destroy alias for the release service
- added unicodePluginFunctions.txt to be used as a file menu, perhaps merged with pprofunctions.txt.

.67:

- added to_utf8 service

.65:

- added set_base, get_base and configuration ini key indexBase, which control whether indexing starts at 0 or 1 in set_chars and get_chars
- added modify() and addmake() members to set of exported functions for use by other plugins, and corrected documentation for others.
- fixed an obscure bug in the set of exported functions which might have resulted in a ommory leak when unicode plugin interacted with other plugins.
- I added slice, find and revfind 0-based services a few version ago, but forgot to update all the documentation. Mea culpa.

.63:

- added second "raw" parameter to keys service.

.61:

- fixed spaces in sendkeys service.

.60:

- Fixed get_char and set_char (and equivalent using []s); they were broke; also they now take up to two position arguments, as well as a type argument, and…
- set_char name changed to *set_chars* (but still has set_char as an alias).
- get_char name changed to get*_chars* (but still has get_char as an alias).
- Fixed file_readall so it don’t crash on files larger than a few meg.
- Added *type* argument for get_chars, set_chars, and a set_default_get_type service to allow setting behaviour of set_chars and get_chars when invoked without a type argument (e.g. via the target[*position*] syntax)
- Added slice, find and revfind – 0-based equivalents of select, index and revindex
- Fixed case("title", str) where str ends in blanks
- Added optional third argument for delimiters to word service

- Added squeeze, reverse and rotate services

.58:

- added sendcopydata service.
- added keys service, providing most of the options available with the *Keys command

.56:

- added **empty** service, allowing creating of empty buffer to receive a unicode string
- added the to_binary service
- Fixed a bug that caused inputdialog and messagebox to misbehave
- added aliases to various services, e.g. str_select for select, ascii for to_ascii. Aliases show up in the tables of services as italicised names

.54:

- renamed create_from_utf8 to **from_utf8**
- renamed convert_to_ascii to **to_ascii**
- added from_mbcs
- changed parameters for file_open, **file_readstring**, file_readline, **file_writeline**, **file_writestring**: now type of file content specified in second parameter of file_open, not in second parameter of handle-dependent file_read* and file_write* services.
- type of file content (format) for file_open, file_readall and file_writeall can be utf8; modifier to utf8 determines if read operations produce plain ascii strings or handles to Unicode strings; the service set_utf8_default determines what's mean when you specify just "utf8" without a modifier
- mbcs now supported for all file reading and writing services.
- the format parameter of file_readall and file_writeall can now be words as well as letters (and are exactly the same as the valid format parameters for file_open)
- handle.service syntax now supported for file handles returned by file_open as well as for unicode string handles
- fixed bug which caused unicode.release_all to crash PowerPro
- fixed bug in inputdialog causing crash on second call
- clarified that file_setmaxline sets number of *bytes* available for a file_readstring or file_readline, not characters.

.52:

- Change the order of arguments in change_case to allow handle.service syntax to work
- If you have PowerPro version 4.04.11 or later, handles returned by a service but neither assigned nor returned via quit (i.e., used as intermediate results in an expression) will be automatically released once the expression is evaluated

.50:

- Support for handle.service syntax for handles to unicode strings

.48:

- Bruce (will change)/(has changed) the mechanism for the automatic destruction of handles (see here) held in local variables as of PowerPro 4.4.07. This version of the unicode plugin is compatible with 4.4.07 or earlier.

.44:

- added version service.
- test scripts now run from any folder, not just <PowerProInstallation>\scripts folder.
- handles to unicode strings now are prefixed "u\x05".
- handles assigned to local variables will automatically be released when local goes out of scope, unless you've used the new localcopy service to override.

.42:

- added release_all service

.40:

Playing catch-up with Bruce, reflecting changes in PowerPro 4.3.06:

- added file_size64
- scripts run from file_allfiles now respect script cancel if error

.39:

Playing catch-up with Bruce, reflecting changes in PowerPro 4.3.04a:

- added **file_runaswait**
- fixed **file_watchfolder** to allow rename in watched folder
- added no error flag for **file_move**, **file_copy**

.37:

- added service **create_from_utf8** that returns a handle to a UTF-16 string.
- added parameters to **file_open** allowing specification of whether file content is UTF-16 or UTF-8
- combined **file_readstringu** and **file_readstringa** services, and added a parameter to resulting **file_readstring** to allow specification of file content type: ANSI or unicode.
- combined **file_writestringu** and **file_writestringa** services, and added a parameter to resulting **file_writestring** to allow specification of file content type: ANSI or unicode.
- combined **file_readlineu** and **file_readlinea** services, and added a parameter to resulting **file_readline** to allow specification of file content type: ANSI or unicode.
- combined **file_ writelineu** and **file_ writelinea** services, and added a parameter to resulting **file_ writeline** to allow specification of file content type: ANSI or unicode.
- combined **file_readallu** and **file_readalla** services, and added a parameter to resulting **file_readall** to allow specification of file content type: ANSI, UTF-16 or UTF-8.
- combined **file_ writeallu** and **file_ writealla** services, and added a parameter to resulting **file_ writeall** to allow specification of file content type: ANSI or UTF-16.

- removed **encodeUnicode.exe** from distribution; its function is now implemented by the encodeUnicode.powerpro script

.35:

- changed name of encoded service to **decode**
- added **encode** service.
- added **inputDialog** service

.33:

- Added **file_runas** service

.31:

- Added **file_version** service

.30:

- First version