

spell/400 - spell checker for the System i

SPELLCHECK

Spell/400 Integration Tool V2

Contents

1 Overview	4
1.1 The option for your customers	4
1.2 Do you need the SPELLCHECK tool?	4
1.2.1 Distribute Spell/400 to each of your customers	4
1.2.2 Check each installation for the Spell/400 commands and APIs	4
1.3 The case for using the SPELLCHECK tool	4
2 Installing the SPELLCHECK library	5
2.1 Checking the contents of the SPELLCHECK library	5
3 Programmer's overview	5
3.1 SPELLCHECK Objects	6
3.1.1 SPELLCHECK program	6
3.1.2 SPELLCHECK data area	6
3.1.3 SPELLCHECK library	6
3.2 A quick example in RPG	6
3.2.1 SPELLCHECK menu	7
3.2.2 The RPG Example	7
3.2.3 The RPG source code	8
3.2.4 How can I test what happens if Spell/400 isn't installed?	9
4 Configure SPELLCHECK options	10
4.1 Using the SPELLCFG program (Menu option 10)	10
4.1.1 The joblog warning	10
4.1.2 Show Warning if SPELL400 not installed	10
4.1.3 Replace warning with the text below	10
4.1.4 Instead of warning, call this program	10
5 Programming	11
5.1 Programming methods	11
5.1.1 Press Enter to continue (forces a spellcheck)	11
5.1.2 Press Fx to Spell Check	11
5.2 Before programming - trying it out on the command line	11
5.3 Before programming - V2 has another way	12
5.3.1 Why is this an advantage?	12
5.3.2 Notes on V2 method	12
5.3.3 Before programming - trying V2 out on the command line	13
5.4 CL programming	14
5.5 SPELLCHECK special functions	14

5.5.1	Specifying a control word	14
5.5.2	You can check if Spell/400 is installed	14
5.5.3	You can debug SPELLCHECK	15
5.6	FAQs	15
5.6.1	Does the window popup on every API?	15
5.6.2	If Spell/400 exists does SPELLCHECK do anything?	15
5.6.3	Explain job switches	16
6	Distributing SPELLCHECK	16
6.1	SPELLCHECK program and dataarea	16
6.1.1	Where does the SPELLCHECK data area go?	16
6.1.2	Security considerations	16
6.1.3	Can inhouse dictionaries be distributed?	17
6.2	Documentation for your client	17
6.2.1	Security implications	17
6.2.2	Can the window popup be stopped?	17
6.2.3	Can the window popup be tested?	17
6.2.4	Can the window popup be changed by the client?	17

1 Overview

The intention of the SPELLCHECK tool is to allow you to distribute your application with the spell checking abilities provided by Spell/400 without the need to distribute or support Spell/400 itself.

If the customer has Spell/400 installed then Spell Checking will work as you'd expect - however if Spell/400 isn't installed then your application will carry on as normal.

SPELLCHECK thus allows you to code Spell Checking into all your applications without fear that some sites will incur 'Command not found' errors. In fact it makes no difference to you whether they have Spell/400 installed, since your application works the same way at any site.

1.1 The option for your customers

Your customers have the option of downloading, evaluating and licensing Spell/400 entirely independently of your agreement with them. If they have already licensed Spell/400 for another purpose then your application will start using it.

If some of your customers do not wish to use Spell/400 then your application will silently skip the spell check.

1.2 Do you need the SPELLCHECK tool?

No, you don't. SPELLCHECK does not provide anything that you cannot already do within your CL or RPG programs. However by using SPELLCHECK you have most of the work done for you!

Without SPELLCHECK you'd have to decide which of the following suited you better:

1.2.1 Distribute Spell/400 to each of your customers

You may find it easier to standardize on Spell/400 for all your customers and make it part of your software distribution. If this is the case then you can ship the evaluation copy of Spell/400.

1.2.2 Check each installation for the Spell/400 commands and APIs

Alternatively, you could incorporate Spell/400 commands and APIs into your application and - provided that you used error checking - your application wouldn't have a problem if it couldn't find the Spell/400 library.

1.3 The case for using the SPELLCHECK tool

If you are unsure about whether the SPELLCHECK tool simply adds an extra layer of complication, here are the advantages:

1. You will not need to distribute Spell/400 (although you still can if you want to). This may be a factor for customers who have already licensed Spell/400 and wouldn't want it overwritten.
2. Incorporating a single SPELLCHECK program (and dataarea) is much easier for distributing and support.

3. You don't need to code the error checking (for Spell/400) in your software and thus have to support one program 'path' for sites with Spell/400 and another 'path' for those without. It also provides a clean cut-off point where your software support can refer any problem with SPELLCHECK (and the resulting calls to Spell/400) to us.
4. You have the option of displaying an information window if Spell/400 is not found - either the default window saying how they can install Spell/400 or a customized window that you have created. If such a popup window is displayed - you and your customer have control over whether it is displayed subsequently.
5. The SPELLCHECK API is almost identical to the Spell/400 APIs - they simply add a new parameter.

2 Installing the SPELLCHECK library

Simply run the EXE program and it will restore library SPELLCHECK. You can obtain the EXE from the www.spell400.com website.

2.1 Checking the contents of the SPELLCHECK library

To check the upload worked successfully:

```
WRKOBJPDM SPELLCHECK
```

You should see the following (or similar) objects:

Work with Objects Using PDM				
Library		SPELLCHECK		Position to
				Position to type
Opt	Object	Type	Attribute	Text
	SPELLCHECK	*PGM	CLLE	VAR Spell Check
	SPELLCFG	*PGM	CLP	Control options in the distributed SP
	EXAMPLECL	*PGM	CLP	Example of a CLP program
	EXAMPLERPG	*PGM	RPGLE	Example of a RPGIV program
	SPELLCFGDF	*FILE	DSPF	Control options in the distributed SP
	EXAMPLEDF	*FILE	DSPF	Example DSPF
	EXAMPLESRC	*FILE	PF-SRC	Examples source code

Bottom

3 Programmer's overview

SPELLCHECK is almost identical to the normal Spell/400 commands and APIs. The difference is that it works whether or not Spell/400 is installed.

If Spell/400 isn't installed, SPELLCHECK can:

- ▷ Log an entry in the joblog, but otherwise carry on working
- ▷ and /or display a popup window saying Spell Checking is not installed

- ▷ and/or run one of your application programs
- ▷ and/or return a monitorable error message

The popup window can be configured to say pretty much anything, giving your customer advice on how they can get the Spell Checking working.

3.1 SPELLCHECK Objects

3.1.1 SPELLCHECK program

The SPELLCHECK program is all that you need to distribute. It allows access to all the Spell/400 APIs and can be embedded in any HLL language.

You do not need to distribute the SPELLCHECK library, simply put the SPELLCHECK program somewhere in the library list of your application.

It can be renamed to keep with your naming standards.

It has some inbuilt controls to allow customisation for any particular site.

3.1.2 SPELLCHECK data area

Optionally, you can also distribute the SPELLCHECK data area - which contains options on changing the default behaviour of the SPELLCHECK program, specifically:

1. If Spell/400 isn't installed, should SPELLCHECK issue a warning? (Default is Yes)
2. If the warning is displayed, what does it say? (Default advises they look at <http://www.spell400.com>)
3. If Spell/400 isn't installed, should it call one of your programs? (Default is no)

If the data area cannot be found (in the library list) then the default behavior is assumed.

3.1.3 SPELLCHECK library

The SPELLCHECK library, and its contents, only need to exist on your development system. Other than the SPELLCHECK program (and dataarea) which are copied to your application library, none of the contents need to be distributed to your client systems.

3.2 A quick example in RPG

Too much theory and this is beginning to sound complicated... So here's an RPG example to show its simplicity.

3.2.1 SPELLCHECK menu

Starting from the main menu:

GO SPELLCHECK/SPELLCHECK

```

SPELLCHECK                               Spell/400 SPELLCHECK API Menu

Select one of the following:

    1. Display overview
   10. Setup VAR options
   11. Show 'No Spell Checker Installed' warning

Examples
   20. CL checks a string
   21. RPG screenscrapes and changes text onscreen

   30. Example source code
   31. Work with objects in SPELLCHECK

Selection or command
====>
Bottom

```

3.2.2 The RPG Example

Take option 21 - the RPG example. If you have Spell/400 you should see a normal Screen-Scrape (STRSPASCN) spellcheck:

```

                               Spell/400 SPELLCHECK API Demo

Enter text here: This text has several misatkes. It is spellchecked
                  with 'wrap' so that words are allowed to croos ov
                  er from one line to the next.
                  This pargraph has several more misatkes in and the
                  lines above shouldnt croos over into this pargrap
                  h.
                  This last pargraph is the last of the three. It ha
                  s a few seperate misatkes but even after chaning a
                  ll of these the words shouldnt croos over.

..... Spell Check .....
: misatkes -> mistakes      :
: spellchecked -> spell checked :
: croos -> cross           :
: pargraph -> paragraph    :
: shouldnt -> shouldn't   :
: seperate -> separate     :
:                          :
: <F3> <F5> <>>>         < OK > :
:.....: F11=Show in 27x132 size

```

Note: You might not be familiar with the *QUICKMIN popup window, which tries to fix all the words in a single go. The easiest way to accept all the words is to move the cursor one position left and press Enter (or press F19).

3.2.3 The RPG source code

The SPELLCHECK program is invoked like this:

```
0029.00 c      Callp SpellCheckScreen('STRSPASCN ' :
0030.00 c      4:20:13:69:Text:NbrRmn)
```

That's it.

3.2.3.1 SpellCheckScreen API The SpellCheckScreen is really just the SPELLCHECK program, as defined in the SPELLCHECK API source member:

```
0016.00 D      PR                      ExtPgm('*LIBL/SPELLCHECK')
0017.00 * VAR needs to specify the actual program to call
0018.00 D      Spell400Pgm              10A  const
0019.00 * Required parameters
0020.00 D      ScrapeStrRow              3P 0  const
0021.00 D      ScrapeStrCol             3P 0  const
0022.00 D      ScrapeEndRow             3P 0  const
0023.00 D      ScrapeEndCol             3P 0  const
0024.00 D      Text                      32000A options(*varsize)
0025.00 * Optional parameters
0026.00 D      NbrErrRmn                 5P 0  const options(*omit:*nopass)
0027.00 * Defaults will be used if the following aren't specified
0028.00 D      Format                     10A  const options(*omit:*nopass)
0029.00 D      Select                     10A  const options(*omit:*nopass)
0030.00 D      Overlay                   10A  const options(*omit:*nopass)
0031.00 D      WinRow                     3P 0  const options(*omit:*nopass)
0032.00 D      WinCol                     3P 0  const options(*omit:*nopass)
```

In fact this is identical to the normal Spell/400 SpellCheckScreen API (see SPELL400/EXAMPLES, APIILE) - with the exception of the first parameter "Spell400Pgm" which specifies the API we want;

```
0029.00 c      Callp SpellCheckScreen('STRSPASCN ' :
0030.00 c      4:20:13:69:Text:NbrRmn)
```

By comparison - this is the normal Spell/400 API, just missing the STRSPASCN parameter:

```
0029.00 c      Callp SpellCheckScreen (
0030.00 c      4:20:13:69:Text:NbrRmn)
```

All the other APIs follow the same format - they call SPELLCHECK with normal parameters, except the first parameter is the real API name. In the RPG program it also calls OvrSpaEnv like this:

```
0015.00 c      Callp OvrSpaEnv('OVRSPAENV ' : *omit:*omit:*omit:
0016.00 c      *omit:*QUICKMIN')
```

Because all the Procedures call the same SPELLCHECK program, but with a different list of parameters, it may make more sense to think of it in terms of normal pseudo RPG:

```
C      CALL  'SPELLCHECK'
C      PARM  'STRSPASCN'  API      10
C      PARM  4            TopRow   3 0
C      PARM  20           StrCol   3 0
C      PARM  13           BotRow   3 0
C      PARM  69           EndCol   3 0
```

And;

C	CALL	'SPELLCHECK'		
C	PARM	'OVRSPAENV'	API	10
C	PARM	'*SAME'	Omit1	80
C	PARM	'*SAME'	Omit2	80
C	PARM	'*SAME'	Omit3	10
C	PARM	'*SAME'	Omit4	10
C	PARM	'*QUICKMIN'	UIM	10

The PR definitions simply make the parameters easier to understand and use (especially with the ability to *OMIT).

You can find all the PR definitions in SPELLCHECK/EXAMPLESRC, SPELLCHECK - however it's just as easy to copy them from the main APIILE definitions as long as you add the first parameter.

3.2.4 How can I test what happens if Spell/400 isn't installed?

Before you go to the trouble of moving the SPELLCHECK program to a test system to see what happens, there is a way of simulating a system that doesn't have Spell/400.

Section 4 on the following page goes into the SPELLCHECK options in more detail, but if you just want to 'Simulate Spell/400 not installed' then :

- ▷ Take option 10 from the SPELLCHECK menu
- ▷ and change 'Simulate Spell/400 not installed' to *YES.

Now run the RPG example again, you should see:

```

                                Spell/400 SPELLCHECK API Demo

Enter text here:  This text has several misatkes. It is spellchecked
                  with 'wrap' so that words are allowed to croos ov
                  er from one line to the next.
                  This pargraph has several more misatkes in and the
                  .....
                  :                               Spell/400 Not Found                               :
                  :                               :                               :
                  : This application has been enabled to use the spell :
                  : correction abilities of Spell/400. There is more :
                  : information and a trial download from:           :
                  : http://www.spell400.com                          :
                  :                               :                               :
                  : Press any key to continue without spellchecking. :
                  :                               :                               :
                  :.....

Mistakes remaining . . . . :

Press enter to check spelling.

F3=Exit      F8=Display environment      F11=Show in 27x132 size

```

The window will pop up saying Spell/400 is not installed. To change this behavior you need to check 4 on the next page.

4 Configure SPELLCHECK options

4.1 Using the SPELLCFG program (Menu option 10)

```

BNB                      Setup Spell/400 VAR options

  Show warning if SPELL400 not installed . . . *ENABLE

  Replace warning with the text below . . . . *NO

  Warning text . . . . .
  _____
  This application has been enabled to use the spell
  correction abilities of Spell/400. There is more
  information and a trial download from:
  _____
  http://www.spell400.com
  _____
  Press Enter to continue without spell checking

  Instead of warning, call this program . . . *NO

  Simulate SPELL400 not installed . . . . . *NO

F3=Exit      F8=Restore defaults      F10=Spell check warning text

```

4.1.1 The joblog warning

SPELLCHECK always leaves a joblog *INFO message if there is a problem.

4.1.2 Show Warning if SPELL400 not installed

This allows the popup (and other options) to be shown if the API isn't found. There are three choices:

1. If *ENABLED then the popup will be shown
2. If *DISABLED then the popup will not be shown and all other options (like calling a program) are disabled as well.
3. You can also set this to *ONCEONLY - where it is *ENABLED for the first time the popup is called, and then changes to *DISABLED (in that job only) for subsequent attempts. Note that this uses job switch 3 (see 5.6.3 on page 16 for more information on job switches).

4.1.3 Replace warning with the text below

If set to *YES then the following text will be used in the popup window. You can change it to refer the customer to your own software support or sales staff. Use option 11 (call spellcheck *popup) to test how it looks.

4.1.4 Instead of warning, call this program

If a program name is entered here, and it is found in the library list at run time, it will be run instead of showing a window. No parameters are sent.

There is a special value of *EXCP which doesn't run a program but instead causes SPELLCHECK to 'fail' with an error message, which may be useful for monitoring.

5 Programming

5.1 Programming methods

There are two main methods of incorporating spell checking into programs;

1. using the normal field validation which happens whenever the user presses Enter
2. or using an optional Function Key to 'validate' the text.

5.1.1 Press Enter to continue (forces a spellcheck)

The normal method to force a spellcheck is to check the text after the user presses Enter, and just as all the fields are validated, the text is checked for spelling mistakes.

This method works well with SPELLCHECK since there is no apparent difference to the user whether they have Spell/400 or not, although it does mean that they will see the popup window every time they press Enter! You can disable the popup window after it has been shown once by changing the 'Show Warning' to *ONCEONLY.

You would probably want to use an indicator in the display file to monitor if the text has changed before checking for mistakes.

5.1.2 Press Fx to Spell Check

The second method leaves the option of Spell Checking to the user - which is started should they press a Function Key.

This is suitable for many sites where spell checking is left to the individual user. The popup window should probably be displayed each time.

5.2 Before programming - trying it out on the command line

Just to get the feel of the SPELLCHECK versatility, you can try it out on the command line;

```
call spellcheck dspspaenv = DSPSPAENV
call spellcheck (savspaenv test) = SAVSPAENV TEST
call spellcheck (ovrspaenv spelling) = OVRSPAENV SPELLING
call spellcheck (lodspaenv test) = LODSPAENV TEST
call spellcheck (dspspaenv *job) = DSPSPAENV *JOB
```

You can try any of these (and other APIs) whilst changing the popup to *ENABLE, *DISABLE and changing the 'Simulate Spell/400 not found' to *YES.

5.3 Before programming - V2 has another way

The V1 method works fine for OPM and ILE programs, but it has an annoying flaw for ILE procedures - the procedure definitions aren't exactly the same as the normal Spell/400 ones. Not a big flaw, but we thought it'd be easier if they were identical.

So we came up with the 2-pass system - the first call to SPELLCHECK identifies the API, and the second uses the same API definitions as the Spell/400 API. For instance, instead of:

```
0015.00 c      Callp OvrSpaEnv ('OVRSPAENV ' :*omit:*omit:*omit:
0016.00 c      *omit:*QUICKMIN')
```

You can now

```
0015.00 c      Callp OvrSpaEnv ('*OVRSPAENV')
0016.00 c      Callp OvrSpaEnv(*omit:*omit:*omit:*omit:*QUICKMIN')
```

5.3.1 Why is this an advantage?

Well if your application was tested using Spell/400 you'd use the normal API definitions:

```
0007.00 *=====
0008.00 *= SPELL CHECK APIS =====
0009.00 /COPY SPELL400/EXAMPLES,APIILE
0010.00 *=====
.....
0016.00 c      Callp OvrSpaEnv(*omit:*omit:*omit:*omit:*QUICKMIN')
```

And now, converting to using the SPELLCHECK tool, the COPY command has a minor change and the CALLP can be left as-is;

```
0007.00 *=====
0008.00 *= SPELL CHECK APIS =====
0009.00 /COPY SPELLCHECK/EXAMPLES,APIILE
0010.00 *=====
.....
0015.00 c      Callp SpellCheck ('*OvrSpaEnv')
0016.00 c      Callp OvrSpaEnv(*omit:*omit:*omit:*omit:*QUICKMIN')
```

The first, and main, advantage is that since the API PR definitions are identical to the Spell/400 APIs then there is no change needed to the CALLP statements.

5.3.2 Notes on V2 method

- ▷ The API name must be passed with an asterisk
- ▷ To change the API stored, just call SPELLCHECK again with the new API
- ▷ Once an API is called, the stored name is discarded and subsequent calls will need to specify the API again.
- ▷ SPELLCHECK tracks whether there is an API name 'stored' by switching on the second job switch. This may conflict with older applications that use job switches. Any call to SPELLCHECK whilst job switch 2 is '1' will use the stored API name

- ▷ To cancel the stored API, either;
 - call SPELLCHECK with a parameter of '*CANCEL' or
 - change job switch 2 to '0' (see 5.6.3 on page 16 for more information on job switches).
- ▷ The API to use on the subsequent call is 'stored' in data area QTEMP/SPELLCMD. This may be a security issue in that someone with access to the command line and CHGDTAARA command could alter the name of the program called.
- ▷ The RPGLE API definitions are stored in SPELLCHECK/EXAMPLES,APIILE - they are the same as the Spell/400 APIILE definitions except they call SPELLCHECK instead of the Spell/400 API.

5.3.3 Before programming - trying V2 out on the command line

Just to get the feel of the SPELLCHECK versatility, you can try it out on the command line;

```
call spellcheck
```

Should say a parameter is needed

```
call spellcheck *dspspaenv
```

Stores DSPSPAENV for the next call

```
call spellcheck
```

Will run the stored DSPSPAENV

```
call spellcheck
```

Should say a parameter is needed, i.e. the stored API can only be called once

```
call spellcheck *savspaenv
```

Stores SAVSPAENV

```
call spellcheck test
```

Runs the stored API SAVSPAENV with a single parameter of TEST

```
call spellcheck *strspascn
```

Stores STRSPASCN

```
call spellcheck *dspspaenv
```

Overwrites the store with DSPSPAENV

```
call spellcheck *job
```

Runs DSPSPAENV *JOB

You can try any of these (and other commands) whilst changing the popup to *ENABLE, *DISABLE and changing the 'Simulate Spell/400 not found' to *YES.

5.4 CL programming

The SPELLCHECK equivalent in CL would be something like:

```
CALL PGM(SPELLCHECK) PARM(OVRSPAENV '*SAME' '*SAME' '*SAME'
 '*SAME' '*QUICKMIN')
```

However it's not very self-documenting to specify all the omitted parameters with *SAME, so you have the alternative of specifying the actual command and then, on an error, displaying the popup window directly:

```
SPELL400/OVRSPAENV UIM(*QUICKMIN)
MONMSG CPF0000 EXEC(DO)
/* Command not found, show the popup warning */
CALL SPELLCHECK *DSPWRN
ENDDO
```

Note: You could use *POPUP instead of *DSPWRN, except *POPUP will always work whether SPELLCHECK is *ENABLED or *DISABLED - whereas *DSPWRN will only work if *ENABLED.

5.5 SPELLCHECK special functions

SPELLCHECK has a few control words that change its behavior. The control word is passed as the first parameter instead of an API name. The *DISABLE, *ENABLE and *DSPWRN are suitable for the customer and are documented 6.2.2 on page 17.

The following are more useful within a program.

5.5.1 Specifying a control word

- ▷ The control word is the first (and, at the moment, only) parameter.
- ▷ It must begin with an asterisk.
- ▷ It can be in lower or upper case.
- ▷ Only the first 3 letters (after the asterisk) are needed (except *DSPWRN which needs all 7 characters to distinguish between *DSP* Spell/400 APIs like *DSPSPAENV).

eg

```
*dis = *DIS, *DISABLE, *DISABLED, *disabled, *disblah
*dis <> DIS, DISABLE, DISABLED ...
```

5.5.2 You can check if Spell/400 is installed

```
call spellcheck (*CHECK')
```

If Spell/400 is found then a CPF9898 *COMP completion message is returned and your program can carry on.

If Spell/400 is not installed then a CPF9898 *EXCP Escape message is sent, which can be monitored.

The *CHECK processing is identical to checking for SPELL400/DSPSPAENV with LOG(*EXCP) except for the additional *COMP message. It will return the escape message if 'Simulate SPELL400 not installed' is set to *YES.

5.5.3 You can debug SPELLCHECK

This may be of help in using SPELLCHECK but is primarily there so that any problems can be copy/pasted into an email for software support.

```
call spellcheck (*DEBUG')
```

*DEBUG dumps the variables into the joblog, eg:

```
6>> call spellcheck dspspaenv
SPELLCHECK DEBUG @ 200109 165610.
SPELLCHECK DEBUG API/Pl: DSPSPAENV.
SPELLCHECK DEBUG PARM1 : DSPSPAENV.
SPELLCHECK DEBUG STATUS: *ENABLE.
SPELLCHECK DEBUG SIMFLR: *NO.
SPELLCHECK DEBUG OVRPGM: *NO.
SPELLCHECK DEBUG JOBSWS: 00010000.
SPELLCHECK DEBUG STDAPT: DSPSPAENV.
SPELLCHECK DEBUG USETXT: *NO.
```

The first time this is run the joblog will be displayed with DSPJOBLOG. Subsequent calls to SPELLCHECK will leave the same information in the joblog but will not display it.

SPELLCHECK 'remembers' the DEBUG dump for every call because of job switch 4. To stop DEBUG either signoff or change job switch 4 (see 5.6.3 on the next page for more information on job switches).

You can also start the debug by setting job switch 4 to '1' which has the advantage of starting it 'silently' - i.e without running DSPJOBLOG.

5.6 FAQs

5.6.1 Does the window popup on every API?

No, it is not displayed when Spell/400 commands such as OVRSPAENV are used - only when the spell checking is invoked. To see what happens, try this:

```
call spellcheck ('OVRSPAENV')
call spellcheck ('STRSPASCN')
```

You will see that both log a failure in the joblog, but only the second shows the popup window.

5.6.2 If Spell/400 exists does SPELLCHECK do anything?

Not really, for instance:

- ▷ If *ENABLED and Spell/400 exists then spell checking will work as normal - ie. no popup window

- ▷ If *DISABLED and Spell/400 exists then spell checking will still work as normal, and still no popup window.

If Spell/400 exists but the license has expired

- ▷ If *ENABLED then spell checking will be ignored - there will be no popup window
- ▷ If *DISABLED then spell checking will still be ignored - there will be no popup window

5.6.3 Explain job switches

Job switches are an archaic way of tracking changes in the job. Their advantages are that they are extremely quick and easy to program.

There are 8 job switches which can individually be either 0 or 1. By default they are all 0, unless a program changes them to 1.

In CL you can use DSPJOB, RTVJOB and CHGJOB to display, retrieve and change the job switches. In RPG they are mapped to indicators INU1-INU8, and can be used as any other indicator. For instance to switch job switch 4 on use CHGJOB SWS(xxx1xxxx) or in RPG set *INU4 to 1.

SPELLCHECK uses job switches 2, 3 and 4.

Job switches are in so little use there is little chance of conflict with your application - however you should be aware that they are used and ensure their use does not conflict with your applications.

6 Distributing SPELLCHECK

6.1 SPELLCHECK program and dataarea

Only the SPELLCHECK program is needed, and it should be placed somewhere in the application library list. If it finds SPELL400 it will automatically make that a product library (the second product library - just in case your application becomes the main product library).

No other objects (SPELLCHECK library, SPELLCFG objects etc) are needed.

The SPELLCHECK program can be renamed to fit in with your naming standards.

6.1.1 Where does the SPELLCHECK data area go?

The dataarea can go anywhere in the library list, which means you can have different dataareas for different applications.

The SPELLCHECK program might have to create a dataarea if it cannot find one, and it will create it in the current library.

6.1.2 Security considerations

The SPELLCHECK program can be made to call the program identified in the SPELLCHECK dataarea, so that both should be made PUBLIC(*USE).

Ideally, by distributing the SPELLCHECK *dataarea, you can control the owner is. In contrast, the dataarea could be created by an end-user, who would then have the authority to change the program called (provided they also have access to a command line and CHGDTAARA command).

6.1.3 Can inhouse dictionaries be distributed?

Yes, any dictionary created by your copy of Spell/400 can be distributed in your application. Should the client install Spell/400 they can import the dictionary with command:

```
IMPSPADCT LIB(your-application-library)
```

The imported dictionary will be copied to the SPELL400 library and assumed to be *SUPPLIED, which means the client will not be able to amend or add to it. If you wish this ability then you need to SAVLIB your SPELL400 library and RSTLIB onto their system. This will require a license code on the client system to work.

6.2 Documentation for your client

Your client may need to know something about the SPELLCHECK program. You can include the following.

6.2.1 Security implications

The client should be made aware of the dataarea being public(*use). See 6.1.2 on the preceding page for more information.

6.2.2 Can the window popup be stopped?

Yes, each site can enable/disable the popup. The easy way to do it is:

```
call spellcheck (*DISABLE')
```

And, conversely, to get it working again:

```
call spellcheck (*ENABLE')
```

When *DISABLED, the popup window and actions to perform are ignored - with the exception that a warning is placed in the joblog. If Spell/400 is installed then the spell checking will still work.

6.2.3 Can the window popup be tested?

Yes, each site can show what warning is displayed.

```
call spellcheck (*DSPWRN')
```

The *DSPWRN code honors the ENABLED/DISABLED state - so if SPELLCHECK is *DISABLED then nothing will be shown.

To show the popup window irrespective of whether it would be shown in normal operations, use *POPUP:

```
call spellcheck (*POPUP')
```

6.2.4 Can the window popup be changed by the client?

There is no user interface for this - the idea is that you supply the text for the window in dataarea SPELLCHECK. Of course the user can change this text with an appropriate use of CHGDTAARA command.