

SPELL Development Environment

SPELL version 2.4.4

Distribution list

Full Report:

SES-SSO-SOE

For Information Purposes:

Open Source

Reference:

SES-SSO-SOE-SPELL-2015/02

Date issued:

February 2015

Acronyms


CV	Command Verification
GCS	Ground Control System
GDB	Ground Database
GUI	Graphical User Interface
HMI	Human Machine Interface (equivalent to GUI)
IDE	Integrated Development Environment
MMD	Manoeuvre Message Database
OOL	Out-of-limits
PDF	Portable Document Format
PROC	Automated SPELL procedure
RCP	Rich Client Platform
S/C	Spacecraft
SCDB	Spacecraft Database
SDE	SPELL Development Environment
SEE	SPELL Execution Environment
SES	Société Européenne des Satellites
SPELL	Satellite Procedure Execution Language and Library
TC	Telecommand
TM	Telemetry
URI	Uniform Resource Identifier
USL	Unified Scripting Language
UTC	Coordinated Universal Time


Table of Contents

1	Introduction	6
1.1	Purpose of this document.....	6
1.2	Application overview	6
2	Startup	7
2.1	Launching the application.....	7
2.2	The workbench.....	11
3	Working with SPELL DEV.....	18
3.1	SPELL projects.....	18
3.2	Procedures	21
3.3	Dictionary files	29
3.4	Files in SPELL DEV and the File system.....	30
3.5	Exporting and Importing	33
4	Preferences	36
4.1	Python interpreter preferences page	36
4.2	Code folding preferences page	38
4.3	Code styling preferences page.....	39
4.4	Custom snippets preferences page	40
4.5	Droplets preferences page.....	41
5	Version control systems.....	43
5.1	Introduction to version control systems.....	43
5.2	Actions on resources	43

SPELL version 2.4.4

5.3	Checkout.....	43
5.4	Update to the latest version	43
5.5	Commit changes	43
5.6	Revert changes	44
5.7	Lock files	44
5.8	Add files to version control.....	44
5.9	History of changes.....	44
5.10	Compare with revision.....	44
5.11	CVS inside SPELL DEV	44
5.12	Project checkout	45
5.13	Committing and Locking files.....	47
5.14	Solving conflicts	49
5.15	Comparing with previous revisions	49
5.16	Automatic CVS actions over new procedures.....	50
6	Semantic Checker	52
6.1	SPELL procedure Semantic Check	52
6.2	Types of checks	52
6.3	Semantic checker interface.....	53
6.4	Check results	55

03 February, 2015	SPELL Development Environment Manual	
Page 5 of 57	File: SPELL - Development Environment Manual - 2.4.4.docx	

03 February, 2015	SPELL Development Environment Manual	
Page 6 of 57	File: SPELL - Development Environment Manual - 2.4.4.docx	

1 Introduction

1.1 Purpose of this document

This document is the software user manual for the SPELL development environment, or “SPELL DEV”. It is intended to be used mainly by SPELL procedure developers.

1.2 Application overview

SPELL DEV is an integrated development environment, or IDE. It is a software tool that provides facilities for SPELL procedure development, including an advanced source code editor, custom code snippets or templates, access to a TM/TC spacecraft database, collaborative work support (via Subversion) and so forth.

2 Startup

2.1 Launching the application

In this section the way of launching the application is explained. SPELL DEV is a multi-platform tool: it is available both for GNU/Linux and Windows platforms.

On the following, the installation directory of the application is referred to as the “SPELL DEV home directory” and it is defined by the environment variable `SPELL_DEV_HOME`.

2.1.1 Launcher files

SPELL DEV is started via launcher scripts: `SPELL-DEV` (for Linux platform) and `SPELL-DEV.bat` (for Windows platform). Both scripts can be found in the `bin` directory of the SPELL DEV home.

When the launcher script is executed, the SPELL DEV splash screen appears. After the initialization phase, the workbench appears on the screen. The workbench is the main application window where all the views, editors and controls are.

2.1.2 Workspace

As any other Eclipse IDE-based application, SPELL DEV uses a “workspace” directory to store all user preferences and projects. The launcher scripts provided in the SPELL DEV packages contain a pre-selection of the workspace, setting it to be the directory `$SPELL_DEV_HOME/workspace`. This way the user is never asked to choose a workspace before start working.

This pre-selection can be removed from the launcher scripts if desired; in this case SPELL DEV will behave as any other Eclipse-IDE application: at startup, the user is asked to select the *workspace directory* to work with (unless configured not to do so), before the application window actually shows up.

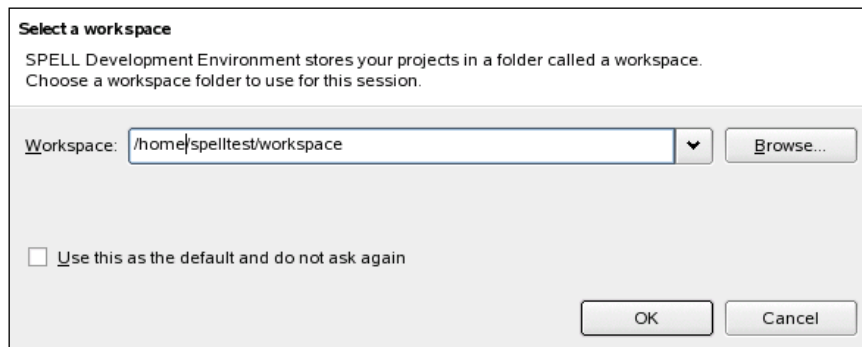


Figure 1. Workspace selection dialog

2.1.3 Selecting the Python interpreter

SPELL DEV requires the presence of a Python interpreter in the operating system, for some features such as the syntax checking mechanism. The first time (and only the first time) the application is started it will also ask the user to select the Python interpreter to use:

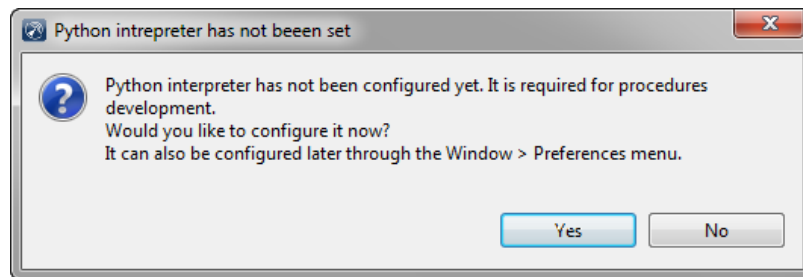


Figure 2. Initial Python interpreter configuration

If the option "No" is chosen, the interpreter can be set later on via the preferences dialog (menu Window/Preferences, section SPELL). Otherwise, the Python interpreter page in the preferences dialog is shown:

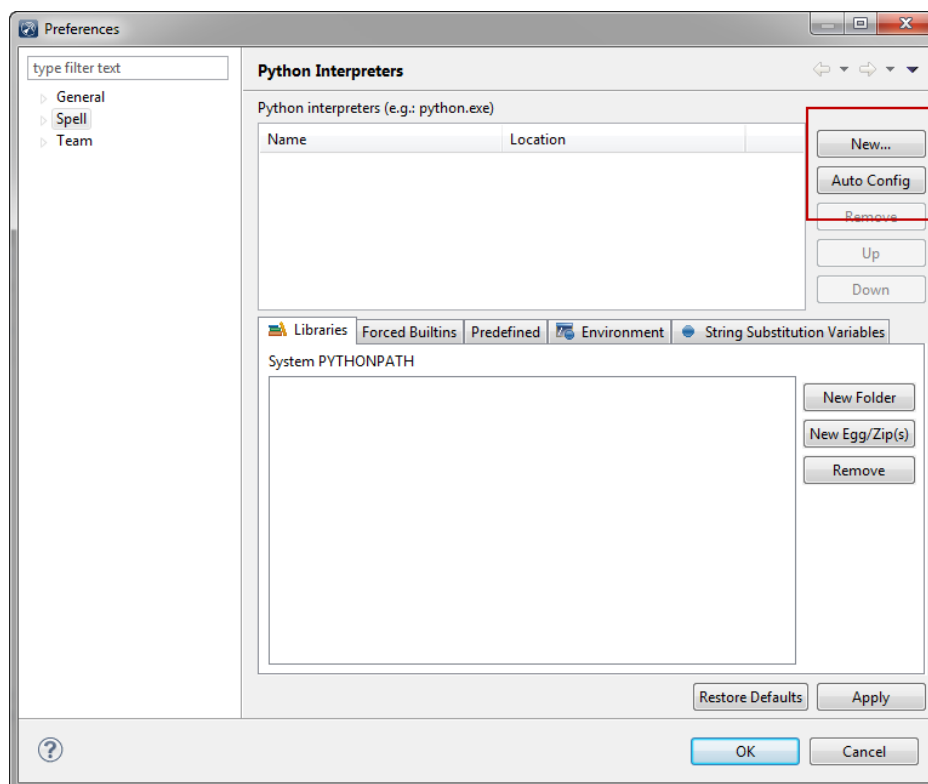
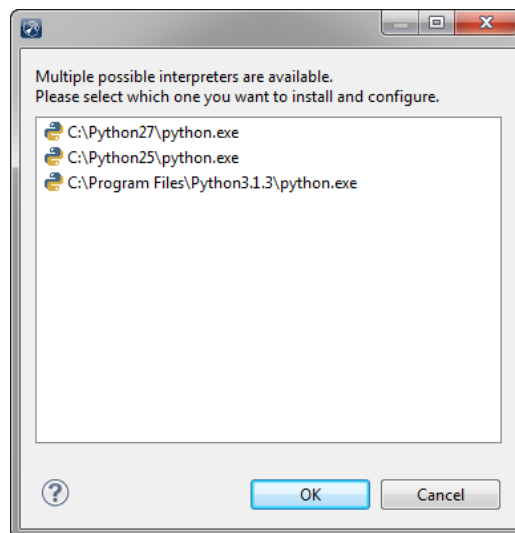


Figure 3. Python interpreter page in preferences dialog

On this preference page, there are two ways of selecting the interpreter to be used. The easiest way to do it is by pressing the "Auto Config" button at the top right side of the page. With this action, SPELL DEV will automatically lookup the default interpreter in the operating system, and will use it as the working interpreter. This feature will only work if the Python interpreter executable can be found in the system path. Otherwise a manual configuration, explained next, is needed.

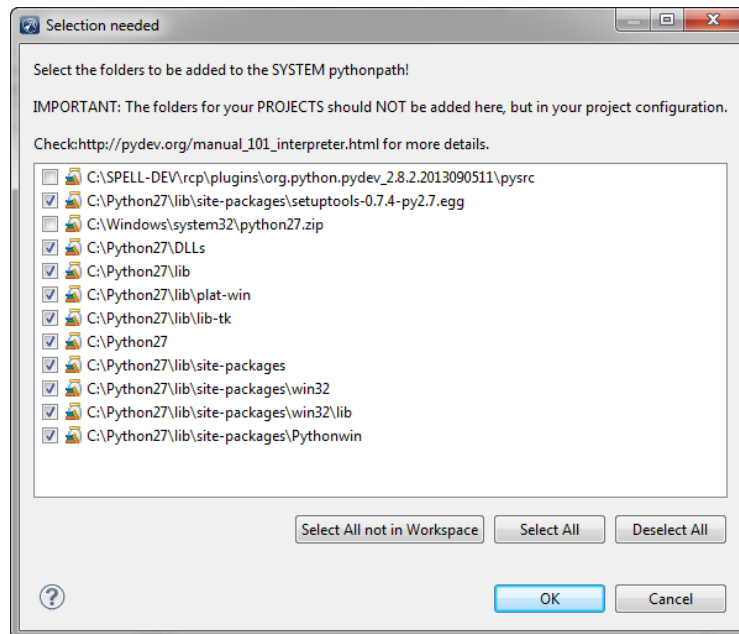
It is possible that several Python interpreters are found. In such a case, the application will ask the user to select one:



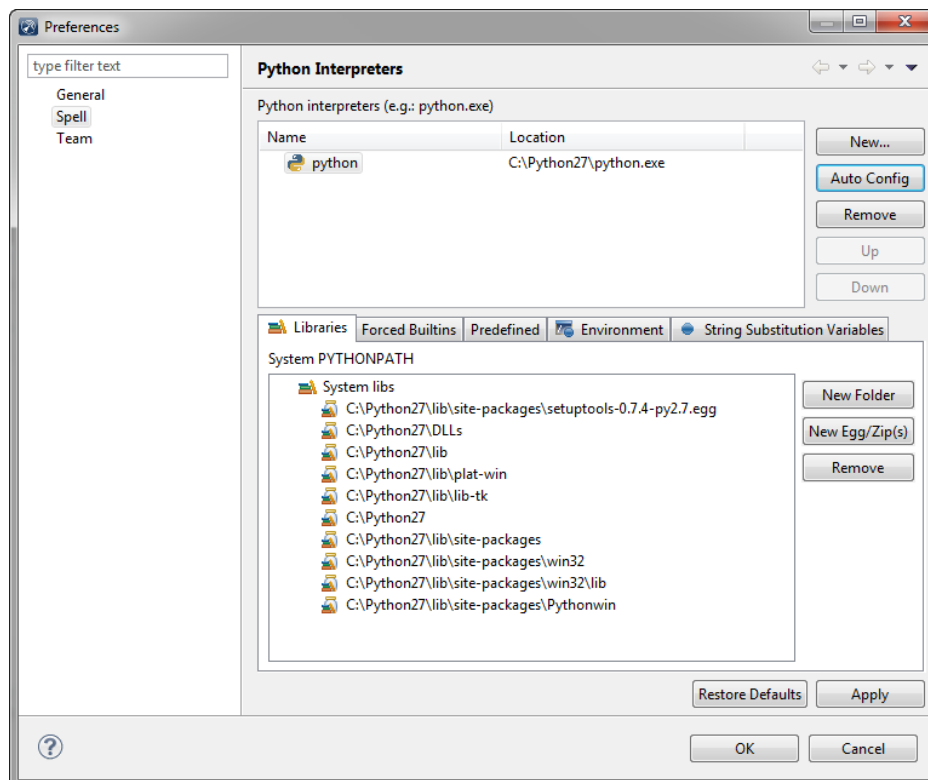
It is always advisable to select a Python version compatible with SPELL. SPELL works fine with Python 2.5, 2.6 and 2.7, but not with 3.0+. These newer versions can be selected in any case for SPELL-DEV work if there is no other choice, but the user must be aware of the differences between the Python frameworks. For example, certain statemens are valid for SPELL and Python 2.5, but they became illegal in Python 3.0 syntax. See [this link](http://docs.python.org) (docs.python.org) for more information.

The second alternative (manual configuration) requires pressing the "New..." button, and to browse on the system directories to look for a Python interpreter. The Python interpreter executable (`Python.exe` or `python`, depending on the platform) shall be selected.

Once an interpreter is selected, the application will present a list of Python libraries to be included in the configuration. Except for very particular scenarios, selecting all presented libraries is a good choice:



Once the process is finished, the interpreter framework should appear in the Python preferences dialog:



The preferences can be closed at this point, it is possible to start working with SPELL procedures now.

As it has been said, SPELL DEV only prompts to set the python interpreter the first time it is launched. Nevertheless, it is always possible to change the selected interpreter using the preferences as described.

2.2 The workbench

The workbench is the complete set of views and tools presented to the user in SPELL DEV. The first time the application is open, it looks like in the image below:

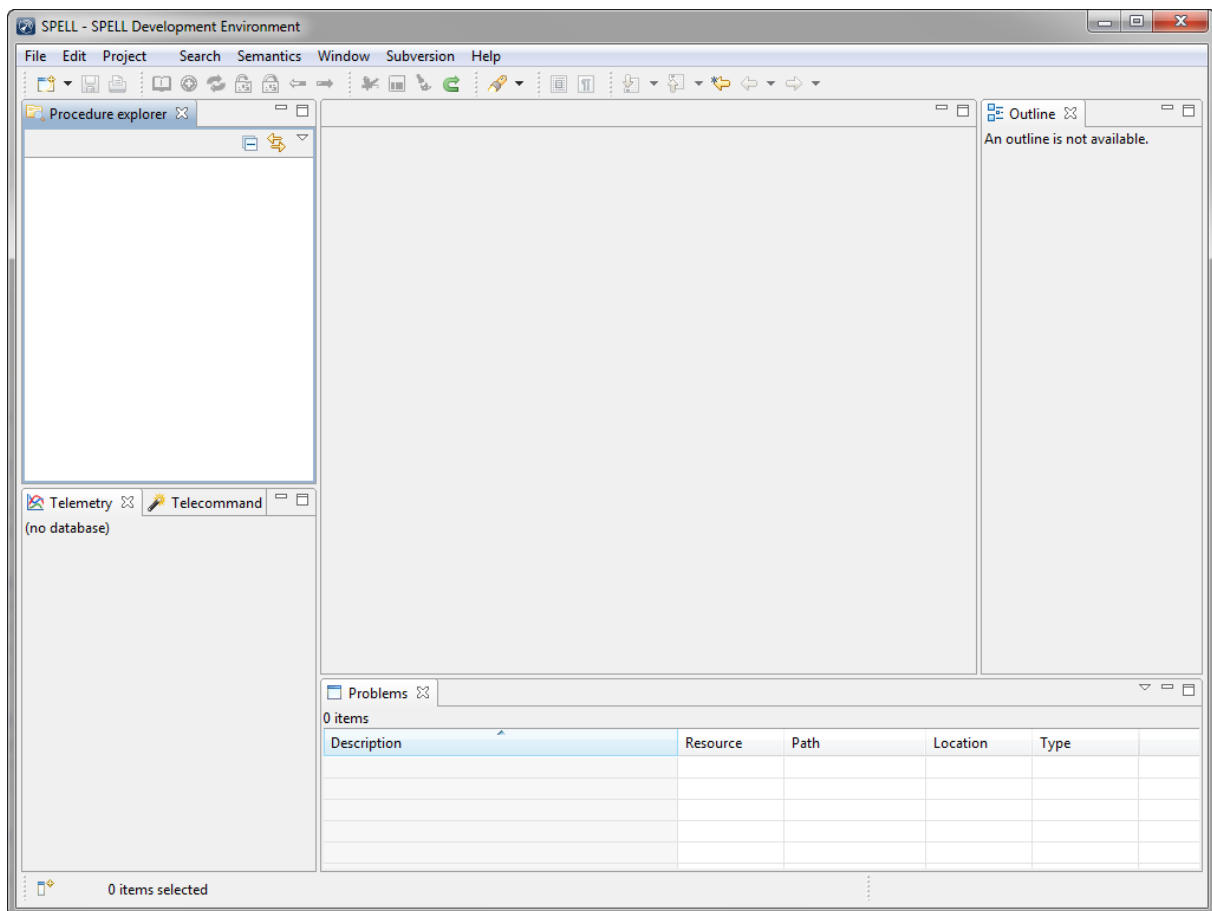


Figure 4. Initial workbench

The most relevant parts are

- **Procedure Explorer**, where all the files and SPELL projects are organized;
- **Outline view**, that provides shortcuts and summaries of the files being edited;
- **Telemetry** and **Telecommand** views, providing information about a TM/TC database of a ground control system.
- **Problems view**. Shows syntax errors, semantic errors and any other issues related to the procedures being edited.
- **Central working area** where all the procedures and other files being edited appear.

2.2.1 Procedure Explorer

This view shows the existing projects in the workspace. It is initially empty. The contents of all projects are shown in a tree structure, where folders and files are represented with different icons depending on their purpose.

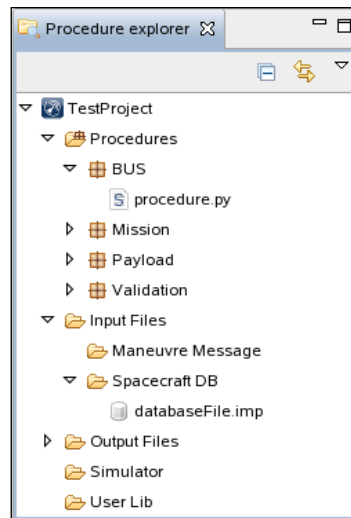


Figure 5. Procedure explorer

From this view it is possible to do file manipulations (copy, cut & paste, rename, delete, move and so forth) and of course open the files to edit them in the central working area (by double-clicking on them).

The contextual (popup) menu in this view shows a number of extra actions:

- New file and Open file actions
- Copy, Cut & Paste
- Delete
- Rename
- Import and Export wizards (used for both files and projects)
- Refresh (see next chapter for file synchronization)
- Team actions (version control for procedure files, e.g. Subversion)
- Comparison tools
- Properties of the elements in the view

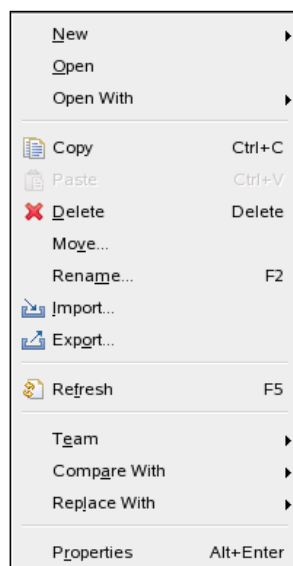


Figure 6. Procedure explorer context menu

2.2.2 Telemetry/Telecommand views

The Telemetry and Telecommand views provide a list of the TM/TC resources (telemetry points and telecommands) which are available in a GCS database. By default, SPELL projects do not have a TM/TC database associated, and therefore these views are initially empty.

When a database is assigned to a project, these views will show available TM parameters and TC elements when developers are working with any of the project files.

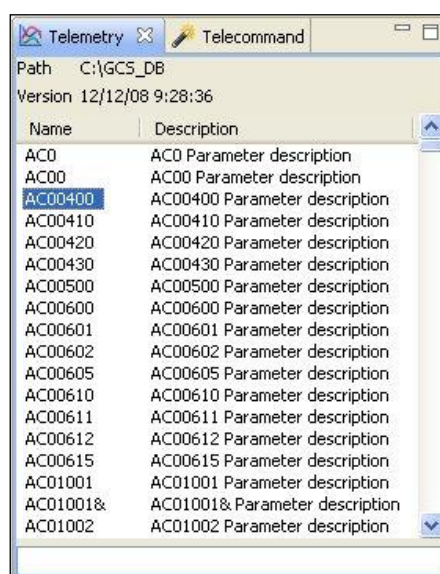


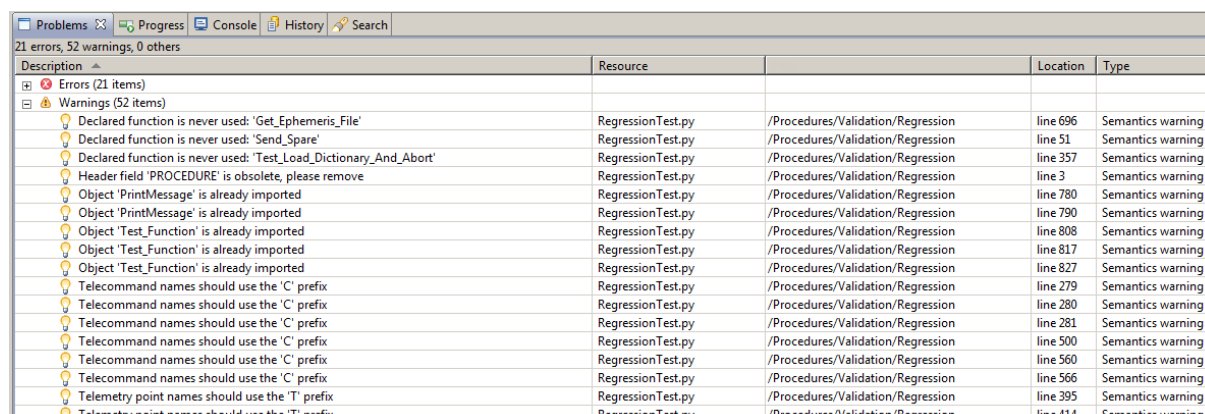
Figure 7. Telecommand and telemetry views

As these views may contain a big amount of TM/TC elements, there is a text widget at the bottom which can be used for filtering elements by their name. By typing some characters, views are refreshed showing the elements whose name contains the given character sequence.

If a procedure or related file is being edited, TM/TC elements can be used for automatic code generation by using the Drag & Drop mechanism described in section [3.2.3](#).

2.2.3 Problems view

When editing a SPELL procedure, the Problems view shows syntax errors found in the procedure, specifying the line number and the possible reason of the language mistakes. It can also show other problems not in relation with syntax, like semantics errors (see Semantic Check in section 6)



Description	Resource	Location	Type
21 errors, 52 warnings, 0 others			
Errors (21 items)			
Declared function is never used: 'Get_Ephemeris_File'	RegressionTest.py	/Procedures/Validation/Regression line 696	Semantics warning
Declared function is never used: 'Send_Spare'	RegressionTest.py	/Procedures/Validation/Regression line 51	Semantics warning
Declared function is never used: 'Test_Load_Dictionary_And_Abort'	RegressionTest.py	/Procedures/Validation/Regression line 357	Semantics warning
Header field 'PROCEDURE' is obsolete, please remove	RegressionTest.py	/Procedures/Validation/Regression line 3	Semantics warning
Object 'PrintMessage' is already imported	RegressionTest.py	/Procedures/Validation/Regression line 780	Semantics warning
Object 'PrintMessage' is already imported	RegressionTest.py	/Procedures/Validation/Regression line 790	Semantics warning
Object 'Test_Function' is already imported	RegressionTest.py	/Procedures/Validation/Regression line 808	Semantics warning
Object 'Test_Function' is already imported	RegressionTest.py	/Procedures/Validation/Regression line 817	Semantics warning
Object 'Test_Function' is already imported	RegressionTest.py	/Procedures/Validation/Regression line 827	Semantics warning
Telecommand names should use the 'C' prefix	RegressionTest.py	/Procedures/Validation/Regression line 279	Semantics warning
Telecommand names should use the 'C' prefix	RegressionTest.py	/Procedures/Validation/Regression line 280	Semantics warning
Telecommand names should use the 'C' prefix	RegressionTest.py	/Procedures/Validation/Regression line 281	Semantics warning
Telecommand names should use the 'C' prefix	RegressionTest.py	/Procedures/Validation/Regression line 500	Semantics warning
Telecommand names should use the 'C' prefix	RegressionTest.py	/Procedures/Validation/Regression line 560	Semantics warning
Telecommand names should use the 'C' prefix	RegressionTest.py	/Procedures/Validation/Regression line 566	Semantics warning
Telemetry point names should use the 'T' prefix	RegressionTest.py	/Procedures/Validation/Regression line 395	Semantics warning
Telemetry point names should use the 'T' prefix	RegressionTest.py	/Procedures/Validation/Regression line 414	Semantics warning

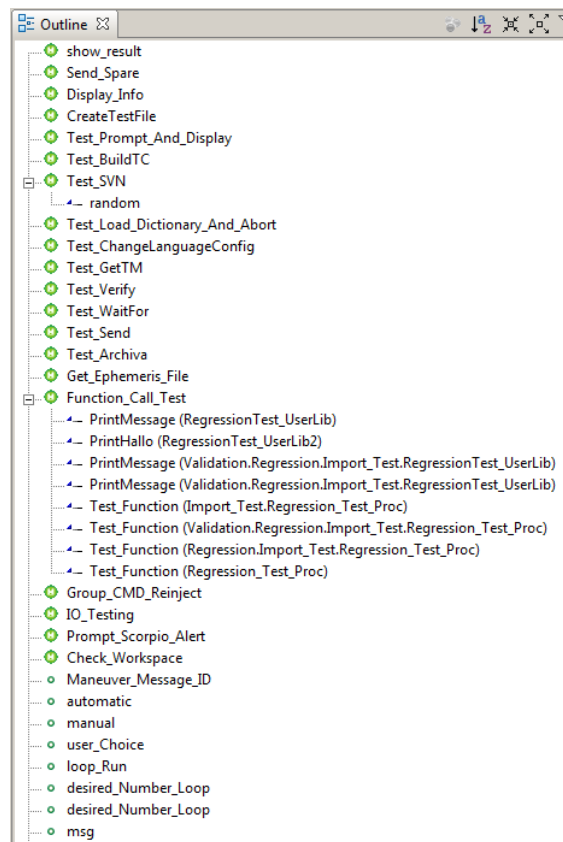
Figure 8. Problems view

In the figure above, one syntax error is shown in the Problems view. First column shows the syntax error, the second shows the procedure file name, the third shows the absolute file path, the fourth shows the line where the error occurred, and the last column shows the error type, which may be a problem or a warning.

Making double click on any error or warning leads to the line where the problem is located.

2.2.4 Outline view

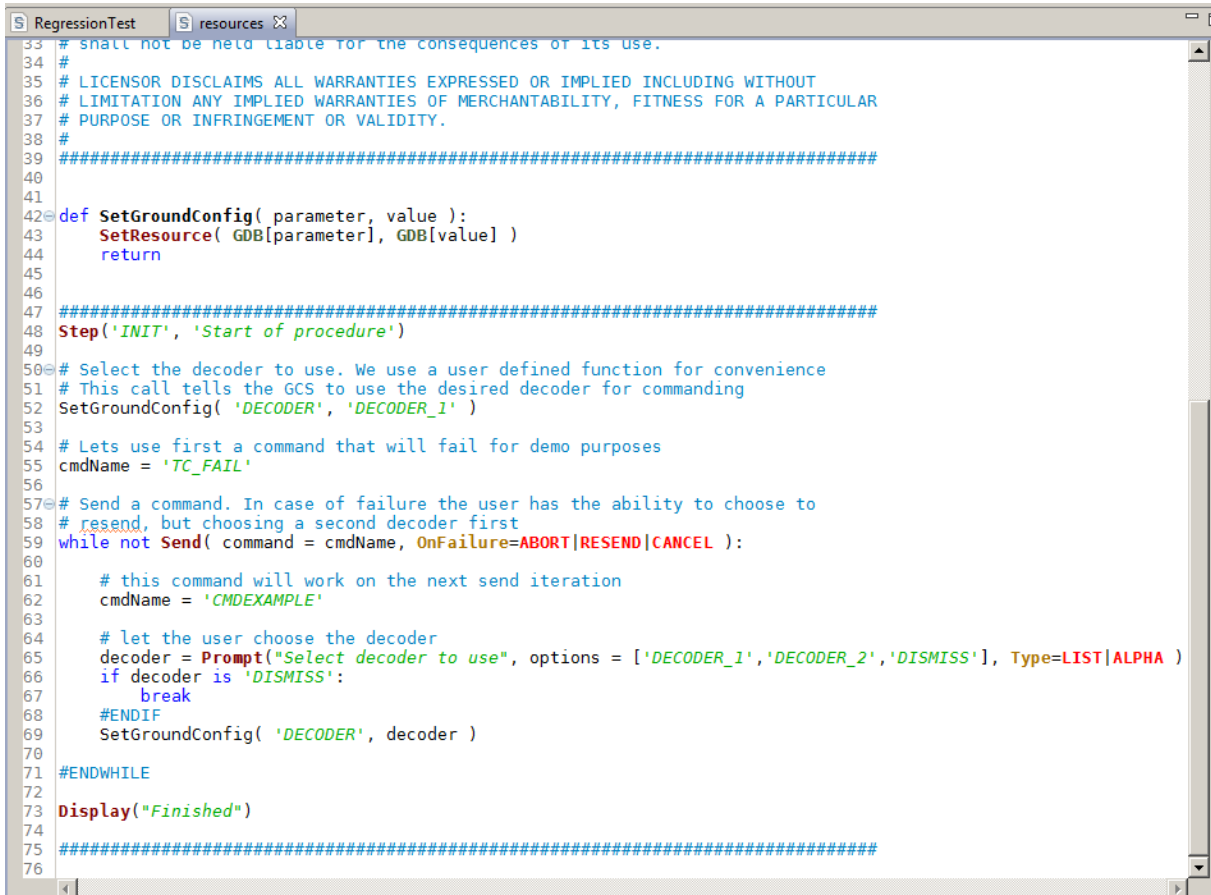
The Outline view is useful when working with large procedures where a lot of variables and functions are defined. This view shows all the programming artefacts that have been defined inside the procedure, such as functions or variables. If one of these elements in the view is clicked, the line where the element is defined becomes visible and highlighted.

**Figure 9. Outline view**

Note that the view allows sorting the elements alphabetically if desired.

2.2.5 Procedure editor

The Procedure Editor allows editing SPELL procedures using the SPELL programming language. It is a flat text editor, including some SPELL language specific features, such as syntax highlighting and automatic edition mechanisms which ease procedure development. This editor is automatically opened when a procedure is double clicked on the Procedure explorer view.



```

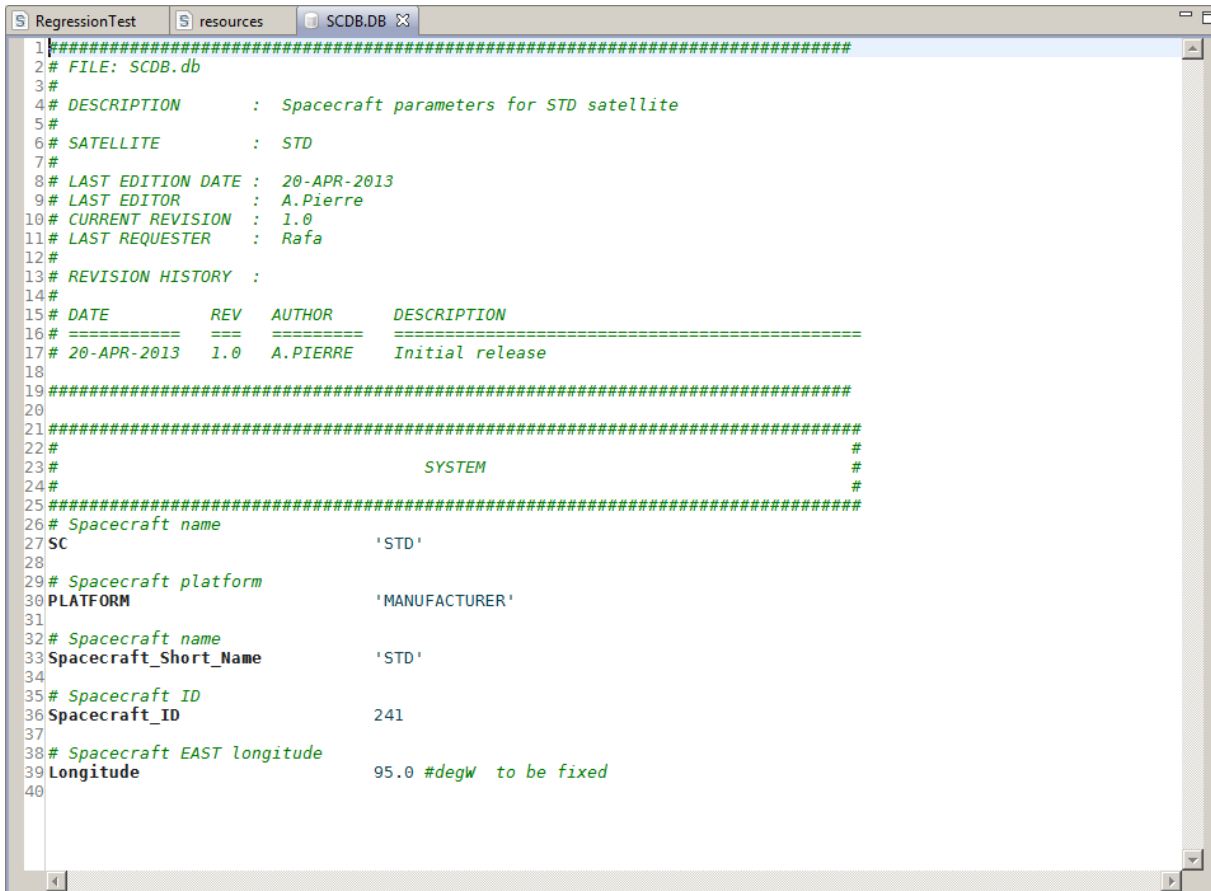
33 # shall not be held liable for the consequences of its use.
34 #
35 # LICENSOR DISCLAIMS ALL WARRANTIES EXPRESSED OR IMPLIED INCLUDING WITHOUT
36 # LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
37 # PURPOSE OR INFRINGEMENT OR VALIDITY.
38 #
39 #####
40
41
42 def SetGroundConfig( parameter, value ):
43     SetResource( GDB[parameter], GDB[value] )
44     return
45
46
47 #####
48 Step('INIT', 'Start of procedure')
49
50 # Select the decoder to use. We use a user defined function for convenience
51 # This call tells the GCS to use the desired decoder for commanding
52 SetGroundConfig( 'DECODER', 'DECODER_1' )
53
54 # Lets use first a command that will fail for demo purposes
55 cmdName = 'TC_FAIL'
56
57 # Send a command. In case of failure the user has the ability to choose to
58 # resend, but choosing a second decoder first
59 while not Send( command = cmdName, OnFailure=ABORT|RESEND|CANCEL ):
60
61     # this command will work on the next send iteration
62     cmdName = 'CMDEXAMPLE'
63
64     # let the user choose the decoder
65     decoder = Prompt("Select decoder to use", options = ['DECODER_1','DECODER_2','DISMISS'], Type=LIST|ALPHA )
66     if decoder is 'DISMISS':
67         break
68     #ENDIF
69     SetGroundConfig( 'DECODER', decoder )
70
71 #ENDWHILE
72
73 Display("Finished")
74
75 #####
76

```

Figure 10. Procedure editor

2.2.6 Database files editor

The database file editor helps developers to edit SPELL database files. Like the procedure editor, it has syntax highlighting for special elements. The SPELL database files consist of lines with key-value pairs, so this editor provides an auto-indent feature to improve database files readability. Multi-line values can be used in the SPELL database, by using the backslash character at the end of each line.



```

1#####
2# FILE: SCDB.db
3#
4# DESCRIPTION      : Spacecraft parameters for STD satellite
5#
6# SATELLITE        : STD
7#
8# LAST EDITION DATE : 20-APR-2013
9# LAST EDITOR       : A.Pierre
10# CURRENT REVISION  : 1.0
11# LAST REQUESTER    : Rafa
12#
13# REVISION HISTORY :
14#
15# DATE            REV   AUTHOR   DESCRIPTION
16# =====
17# 20-APR-2013     1.0   A.PIERRE   Initial release
18#
19#####
20#
21#####
22#
23#                               SYSTEM
24#
25#####
26# Spacecraft name
27SC                               'STD'
28#
29# Spacecraft platform
30PLATFORM                               'MANUFACTURER'
31#
32# Spacecraft name
33Spacecraft_Short_Name                'STD'
34#
35# Spacecraft ID
36Spacecraft_ID                        241
37#
38# Spacecraft EAST longitude
39Longitude                            95.0 #degW to be fixed
40#

```

Figure 11. Database editor

3 Working with SPELL DEV

3.1 SPELL projects

Projects are the main structural element inside SPELL DEV. They group a set of procedures with the dictionaries and some other input files they require to be executed. Typically, a SPELL Project will be associated to a certain spacecraft and will contain all SPELL procedures related to it.

3.1.1 Creating a SPELL project

To create a new SPELL project, follow these steps:

1. In the menu area, select File > New > SPELL project.

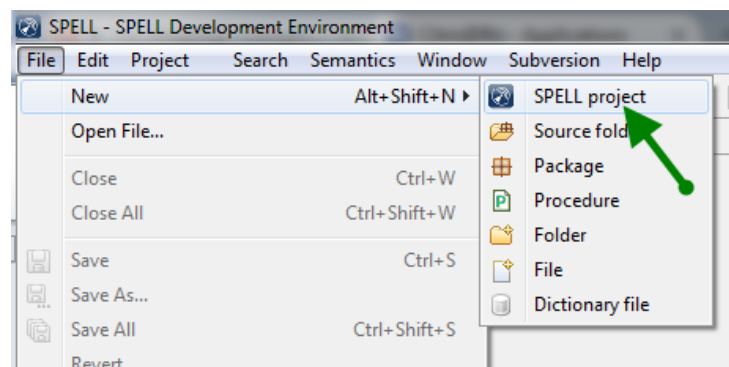


Figure 12. New element wizards

2. When the project creation dialog appears, fill it with the project name, the folder where it will be stored, and the SPELL language interpreter to be used. If selected, also a default folder structure for the new project will be created. Otherwise, no folder will be created inside the project directory. This default folder structure can be modified in the application preferences page.

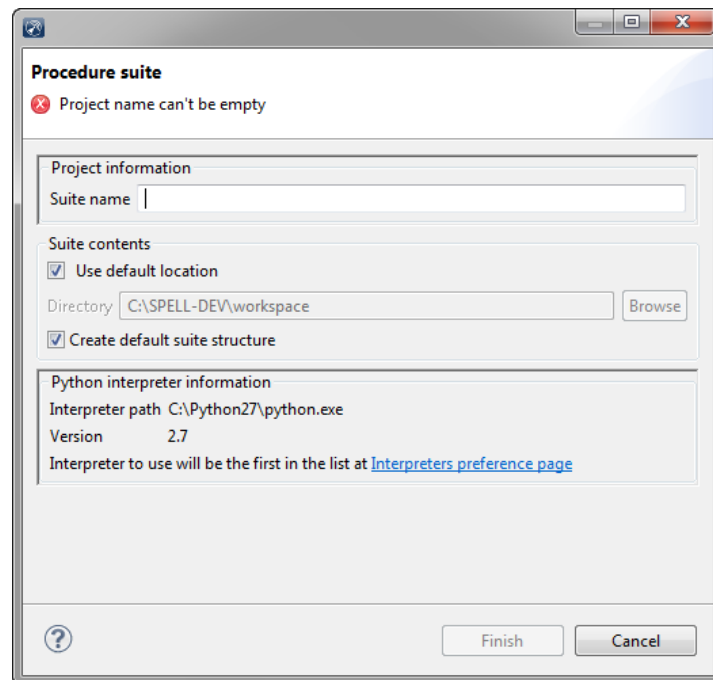
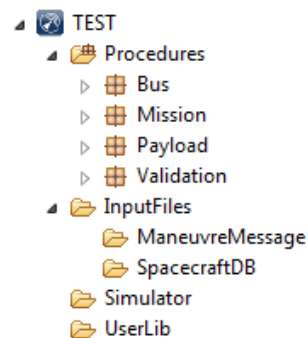


Figure 13. New SPELL project wizard

- When pressing the Finish button, the project will be created with a default folder structure:



This initial structure is configurable from the configuration XML file for SPELL DEV.

3.1.2 Project elements

The Procedure explorer allows to browse through a project's contained elements. These elements are presented to the user with a label and a specific icon depending on their target. The following table presents the different elements which can exist in the Procedure Explorer.

Name	Description
Project	SPELL project
Source folder	Special folder which may contain procedures and packages
Package	Folder which defines how contained procedures can be imported or exported
Folder	System folder
Procedure	SPELL procedure
Dictionary	SPELL dictionary

The default project structure is defined in the configuration file the SPELL DEV instance has loaded.

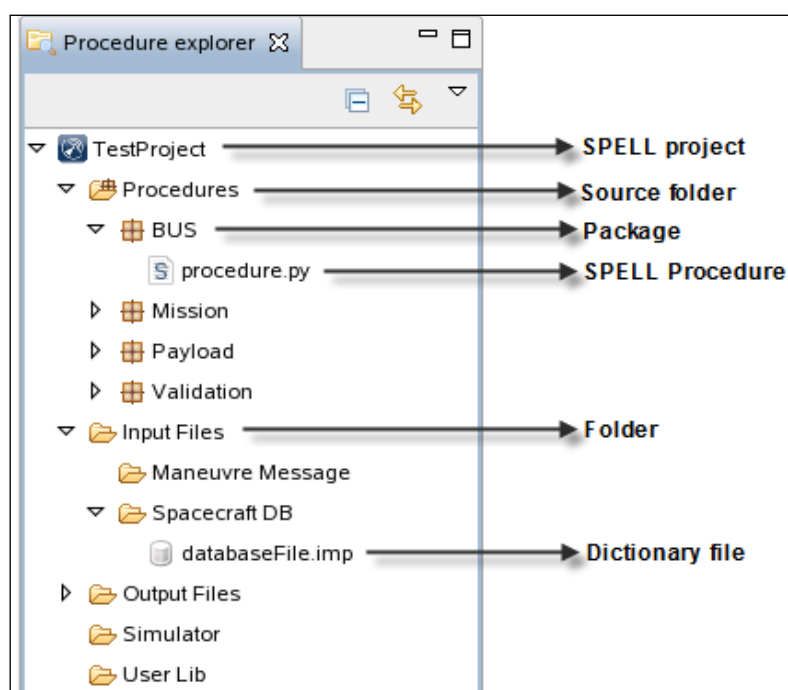


Figure 14. SPELL project elements

Every SPELL project may contain only one single Source folder, and SPELL procedures must be created inside it or its inner packages or folders.

3.1.3 Setting project properties

3.1.3.1 Database properties

As it has been explained, the telemetry and telecommand views show information contained in the GCS database associated to a SPELL project. There are two ways of changing the database for a project:

1. If there is any procedure or database file open, in the menu area there is a Project menu, which shows a Change database properties action. Click on it and the properties dialog will appear focusing on the database properties page.
2. Make right click on the suite folder in the Procedure explorer view, and press on Properties in the context menu. Properties dialog will appear, and database properties page can be selected at the right side of the dialog.

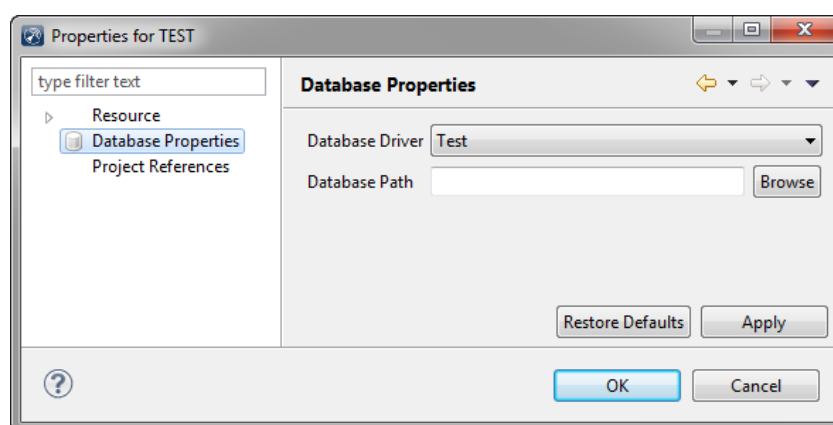


Figure 15. Database properties page

3.2 Procedures

3.2.1 Create a new procedure

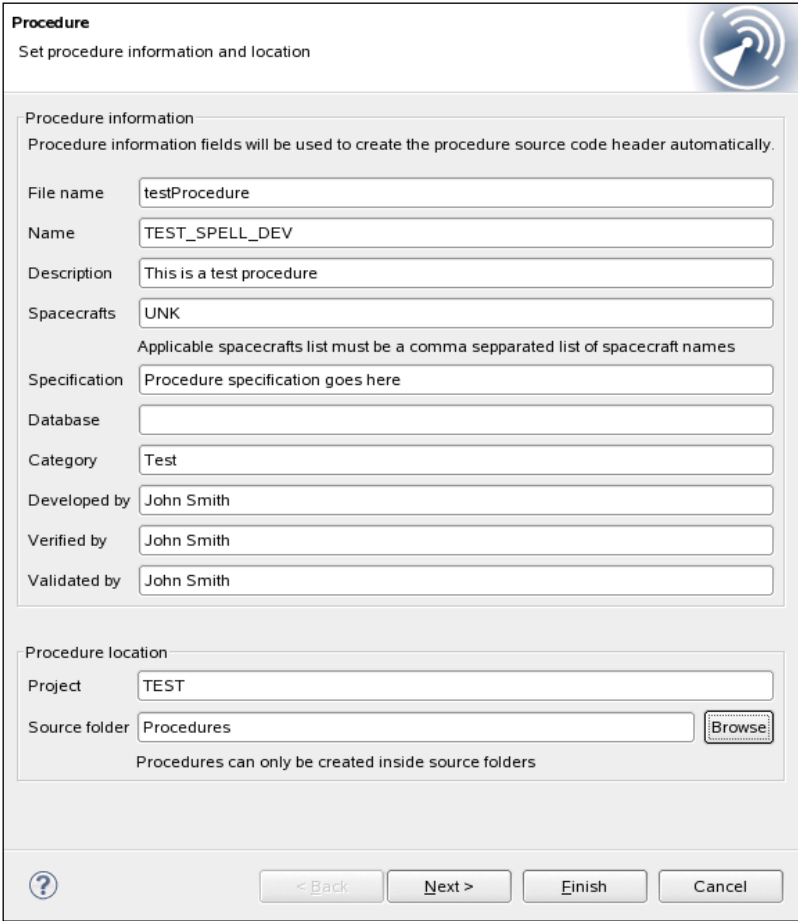
To create a new procedure file, follow these steps:

1. Select the folder of the project, where the procedure is to be created.
2. In the menu area, select File, then New, and finally Procedure. A dialog will appear.



Figure 16. New procedure wizard access

- In the procedure creation dialog, the user shall enter the procedure information, as well as its location, such as the project where the procedure will be contained, and the path where the procedure will be located. Path can be easily entered by clicking on the Browse buttons and selecting a target folder or package in the shown dialog. Procedure information is used to automatically insert the code header in the new procedure.



Procedure
Set procedure information and location

Procedure information
Procedure information fields will be used to create the procedure source code header automatically.

File name: testProcedure

Name: TEST_SPELL_DEV

Description: This is a test procedure

Spacecrafts: UNK
Applicable spacecrafts list must be a comma separated list of spacecraft names

Specification: Procedure specification goes here

Database:

Category: Test

Developed by: John Smith

Verified by: John Smith

Validated by: John Smith

Procedure location

Project: TEST

Source folder: Procedures

Procedures can only be created inside source folders

Figure 17. New procedure wizard

- Press the Finish button and the procedure will be created and opened for its edition. When it is created, the procedure contains the source code header with the information provided in the dialog's page.

Notice that a procedure shall always be contained inside a package, and a package shall always be contained inside a source folder. Both source folders and packages can be created through the File menu in the menu area.

Another way of creating procedures is by using the context menu in the navigator view. If right click is done when the source folder, or any of their inner elements is selected, a New > Procedure option will appear in the context menu. After selecting it the dialog shown in the figure above will be shown.

The dialog contains an additional page to automatically put the new procedure under control version. This dialog page is presented and explained in section [5.3.5](#).

3.2.2 Editing procedures

To edit a procedure, just make double click on it in the Procedure explorer view, and a text editor will be opened in the editor area to edit the procedure. In case it is already opened, the procedure editor will be brought to top in the editor area.

3.2.2.1 Column mode

The column mode is a powerful feature which allows developers to manipulate rectangular regions of code and perform operations like copying, cutting, pasting and modifying the indentation. Also in column mode, typing characters affect the whole set of selected lines at the same time.

To enable or disable column mode, just toggle the button located in the menu area.



Figure 18. Column mode icon

To select a rectangular block of source code, hold the left mouse button and draw a rectangle with the desired size. When releasing the mouse button, a code region will be selected and ready to manipulate.

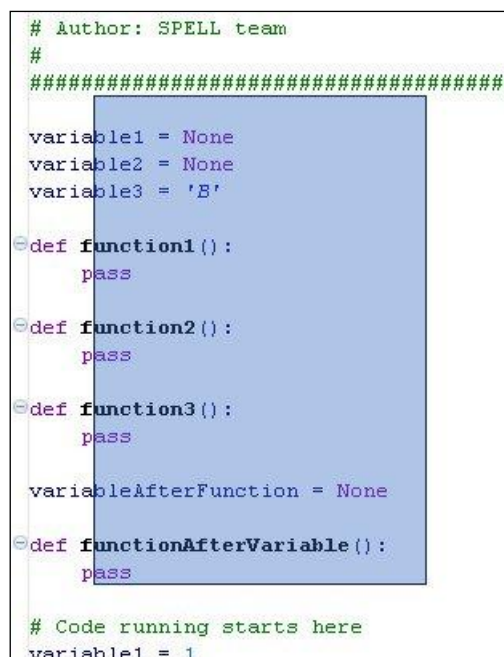
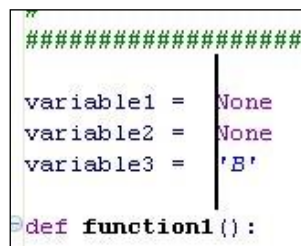


Figure 19. Rectangular selection

If a region of code needs to be aligned, hold the left mouse button and draw a vertical line in the source code editor. When releasing the mouse, character at the right might be moved by pressing space key. It is not recommended to use tab keys for indentation using column mode.



```
#####  
  
variable1 = None  
variable2 = None  
variable3 = 'B'  
  
def function1():
```

Figure 20. Block indentation

If a different key is pressed, the pressed character will be inserted in each line.

3.2.2.2 Automatic code generation

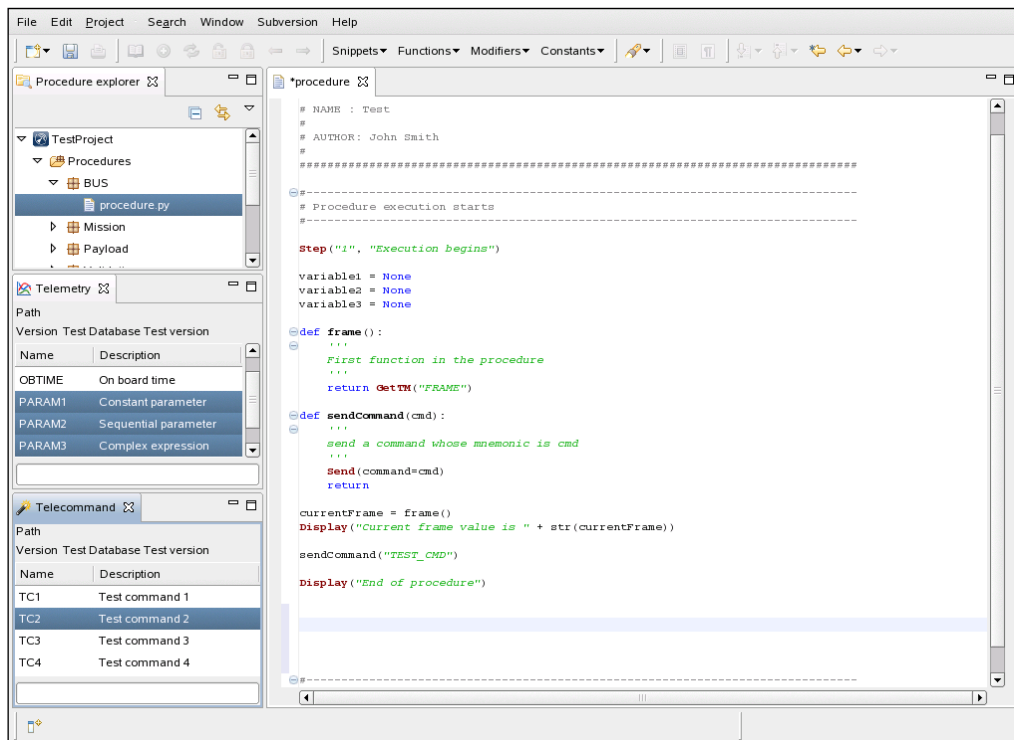
SPELL DEV provides coding facilities to ease users to coding procedures faster. These facilities consist of a database views drag and drop mechanism and the snippets toolbar.

Database Drag & Drop

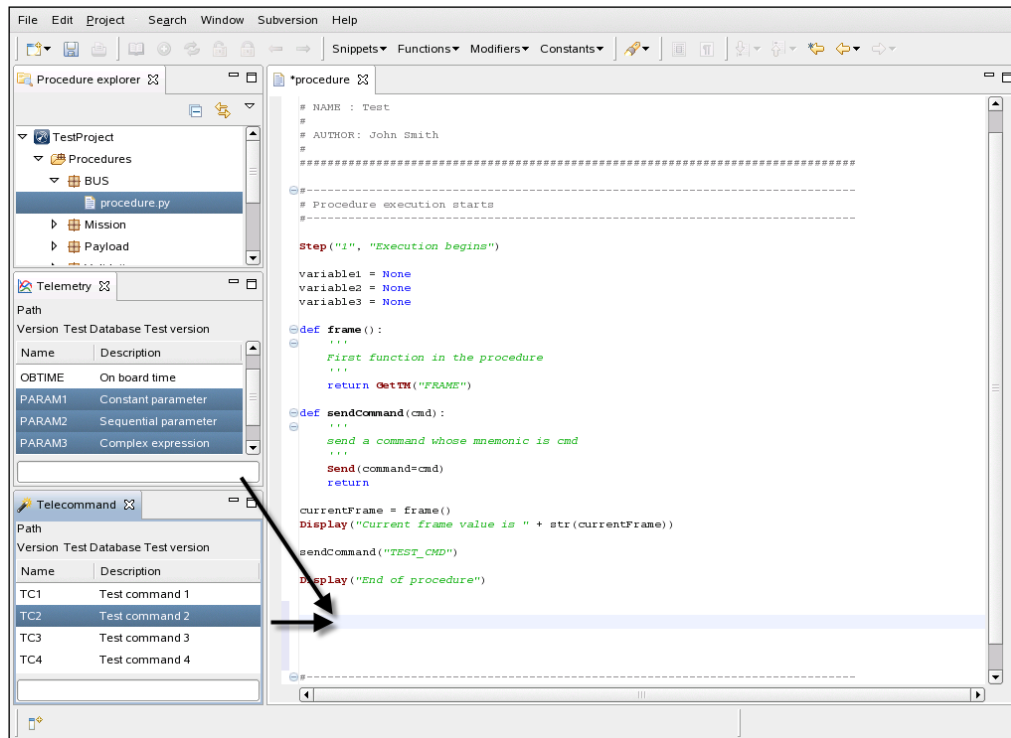
Telecommand and telemetry views provides a drag and drop mechanism which generates Spell source code automatically. Generated code consists of basic structure for each of the SPELL language primitives, so users are supposed to tune the generated code by adding modifiers to it.

To code a procedure using this feature, follow these steps:

1. User shall select the elements from the telecommand and telemetry views he wants to work with.

**Figure 21. Select TM and TC elements from the views**

2. Drag selected elements by pressing the left mouse button and moving the mouse pointer over the SPELL procedure.

**Figure 22. Drag selected elements**

- Drop the dragged elements by releasing the left mouse button over the procedure editor. A popup will appear with the SPELL code that can be generated with the dropped elements.

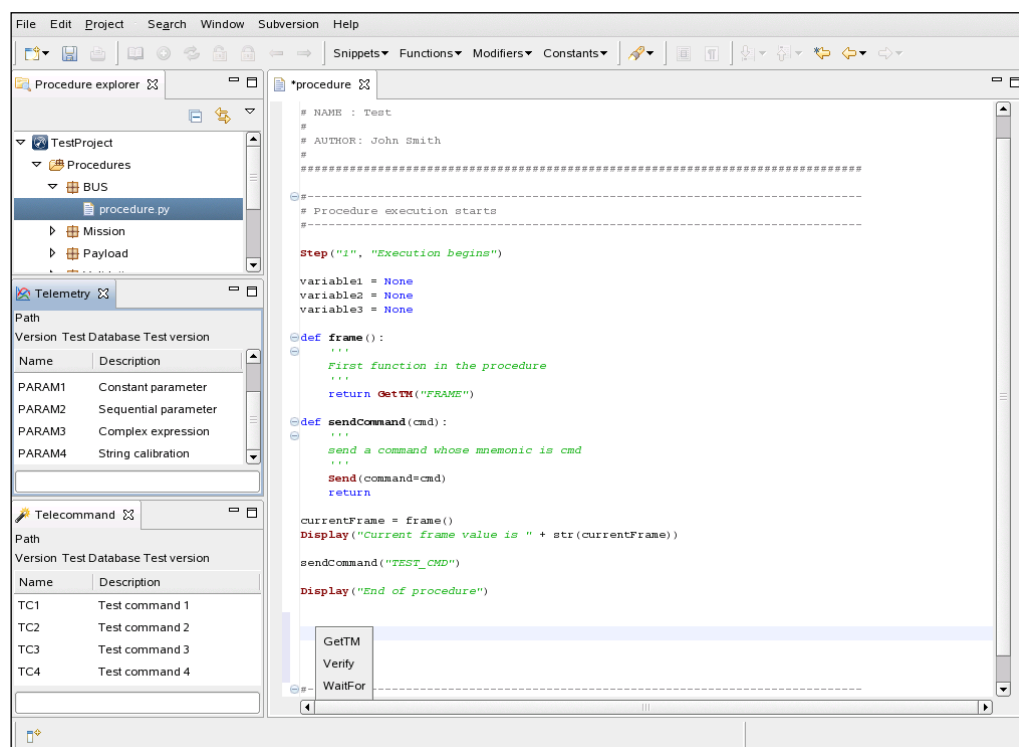


Figure 23. After dropping, code generation options are shown

- Once the code option has selected, code will be automatically inserted in the place where the drop was made.

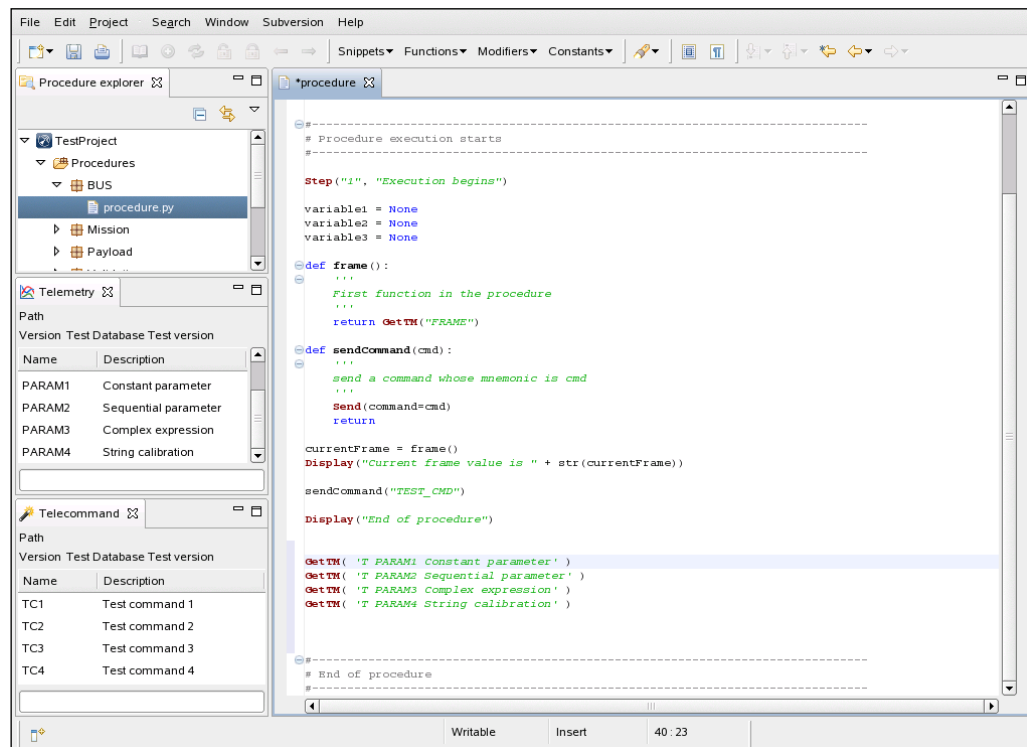


Figure 24. Code has been inserted

Droplets code can be customized through the preferences dialog. Droplets customization preference page is presented in section [4.5](#).

Snippets toolbar

Another facility for editing SPELL procedures is to insert code snippets available in the toolbar area. Once a SPELL procedure is opened, a toolbar with 4 buttons appear. This toolbar allows developers to insert code pieces related to functions, modifiers and constants. SPELL DEV also allows the user to define his own code snippets for inserting in the procedure.

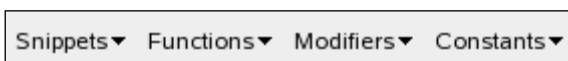


Figure 25. Snippets toolbar

To insert a snippet in the procedure currently being edited, just press any of the button in the toolbar, and select the snippet that wants to be inserted.

Users can define their own snippets, as well as to edit or remove previously defined ones. The mechanism for creating custom snippets is described further in section [4.4](#).

3.3 Dictionary files

3.3.1 Database file types

There are two different kinds of dictionary files:

- DB files, used for the spacecraft and ground databases
- IMP files, used as manoeuvre messages files

3.3.2 Creating a dictionary file

To create a database file, follow these steps:

1. In the menu area, select File, then New, and finally Dictionary file. A wizard will be shown.

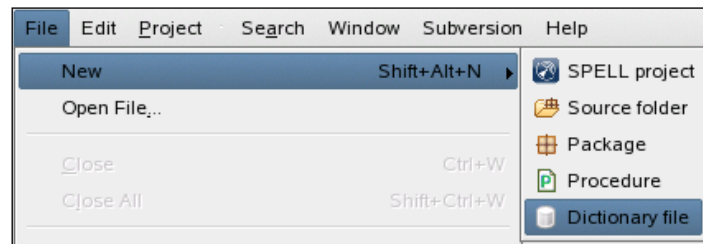


Figure 26. New dictionary wizard access

2. In the dictionary file creation dialog, user must enter the path where the file will be stored, and file name, including the file extension, and the database file type. The path can be easily entered by clicking on the Browse button.

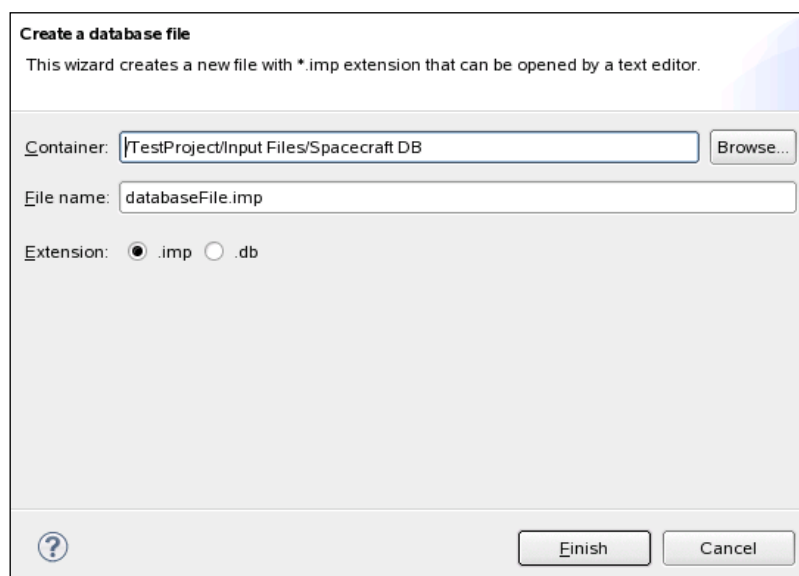


Figure 27. New dictionary file wizard

3. Press the Finish button and the database file will be created and opened for its edition

To edit a dictionary, make double click on the dictionary file in the Procedure explorer view, and a text editor will be opened to edit the file. If it is already opened, the file editor will be brought to the top in the editors' area.

3.4 Files in SPELL DEV and the File system

All Eclipse IDE based applications including SPELL DEV keep an internal model of all the files. The application performs many operations by using this model, and it is paramount to keep this model synchronized with the "real files" in the O.S. file system. SPELL DEV normally takes care of that synchronization transparently to the user. It is in any case a good idea to keep in mind this concept when working with SPELL DEV.

3.4.1 File manipulation from outside SPELL DEV

Because of the internal file models, the user should refrain from manipulating the SPELL DEV project files manually through a file explorer (e.g. Windows Explorer). When doing that, the internal SPELL DEV model will get out of sync. SPELL DEV is prepared to react to these situations and warn the user appropriately, but this "out of synchronization" state is typically confusing for users not accustomed to Eclipse IDE-based applications.

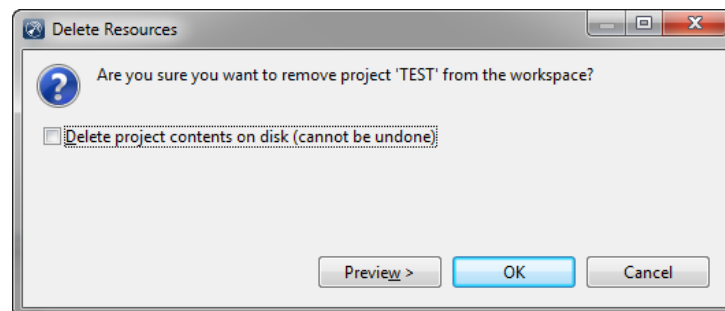
Any files relevant to SPELL DEV, that is, any file or folder existing inside a SPELL DEV Project, must be manipulated (renamed, deleted, moved, copied,...) from within the Procedure Explorer view.

If there is no other choice but to manually manipulate files from outside SPELL DEV for any reason, there are some actions that can be carried out to re-synchronize SPELL DEV and the file system:

- Use the **Refresh** action (F5) on any of the items, files, folders or projects, in the Procedure Explorer.
- Use the **Refresh** action (F5) on a currently open editor which is out of sync with the real file in the file system (normally the user should be advised to do so by the application automatically)
- If refresh does not help, it can be also be a good idea to close and open the project containing the resources (files and folders) out of synchronization.
- Deleting the project from the workspace (not from the file system!) and re-importing it into SPELL DEV can also help in some situations (see next section)

3.4.2 Deleting projects in SPELL DEV

Because of this internal file model of the application, the user must choose what to do exactly when **deleting** a project from the workspace:



If the "Delete project contents on disk" checkbox is NOT marked, the project will be removed from the SPELL DEV workspace, but not from the file system. That is, the project will not be visible in the application anymore, but the actual files will remain on the workspace directory.

In order to, not only remove the project from SPELL DEV, but also delete the files from the workspace directory, the checkbox needs to be marked. Note that this action cannot be undone.

3.4.3 File naming and operating systems

Users should be aware of the particularities of the operating system under which SPELL DEV is executed, regarding file names. For example:

- Windows platform is not CASE sensitive ("test.txt" is the same as "TEST.TXT")
- Linux platforms IS case sensitive ("test.txt" is a completely different file than "TEST.TXT")

This very simple idea is sometimes forgotten and this may cause troubles to users, in working environments that involve both Windows and Linux systems and version control systems like Subversion.

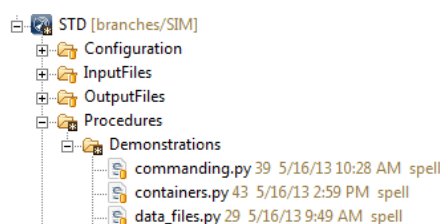
3.4.4 File manipulation and version control

When using version control systems in SPELL DEV like CVS, Git, Mercurial or Subversion, care must be taken whenever files or folders are moved, renamed or deleted in the Procedures Explorer. Version-controlled files cannot be simply moved from one directory to another, but the some rules of the version control systems are to be observed.

In the following some typical operations are explained when Subversion is being used as version control system.

3.4.4.1 Deleting files

When a file is deleted, it disappears from the Procedure Explorer, but still a *commit operation* is needed to complete the deletion of the file in the Subversion repository. This is shown in the view by marking the folder containing the deleted file as "modified" (typically with a black asterisk). In the image below, a file within the "Demonstrations" folder was deleted. In order to completely delete the file, the project, a higher-level folder or at least the "Demonstrations" folder shall be committed in Subversion.



The same applies to folder deletion operations.

3.4.4.2 Moving files

Moving a file can be seen as a two step operation:

- Copy the file to the new location (this is a new file)
- Delete the old copy of the file

By looking at the operation in this way, it is easy to see that BOTH the folder that originally contained the file, and the folder that contain the new file, need to be committed in Subversion.

3.4.4.3 General advice

Some general ideas to ease the work with Subversion are given below:

- Always update your SPELL projects before start working on them, every day. Somebody may have changed the files since the last time you updated them!
- Try to commit small changes, for example, if a file is deleted or moved, commit the changes for that single operation and then continue. If many operations are combined in a row it may be more difficult to recover from any situation.

3.5 Exporting and Importing

3.5.1 Importing projects from the file system

It is possible to import already existing SPELL projects into the current SPELL DEV workspace. This may be required when changing the workspace location. If the developer wants to keep using a project which was in an old location, it is possible to bring it to the new workspace.

To do so, select the menu item “File/Import...”. The import wizard dialog appears:

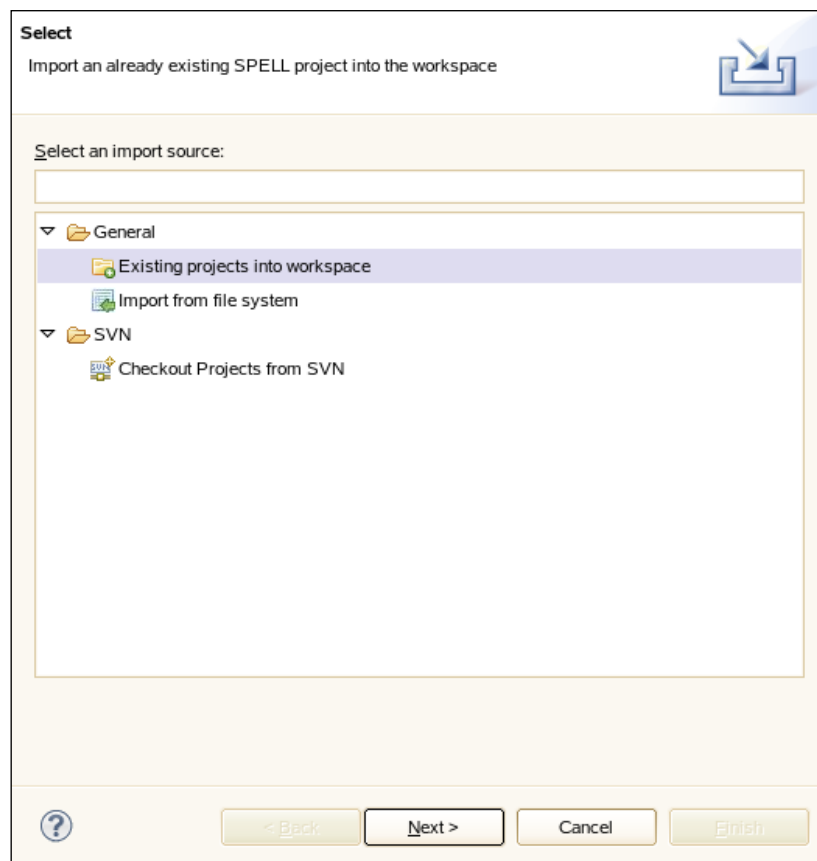


Figure 28: import dialog

Then choose the option “General/Existing projects into workspace”. The next page in the wizard allows the user to browse the file system and to select the project folder to import (see Figure Figure 29).

Once a correct location is chosen, all SPELL projects found within it would appear on the “Projects” list. Then, the user can select the projects to import. By clicking on the “Finish” button the selected projects are imported into the current workspace.

Imported projects appear then in the Procedure Explorer view.

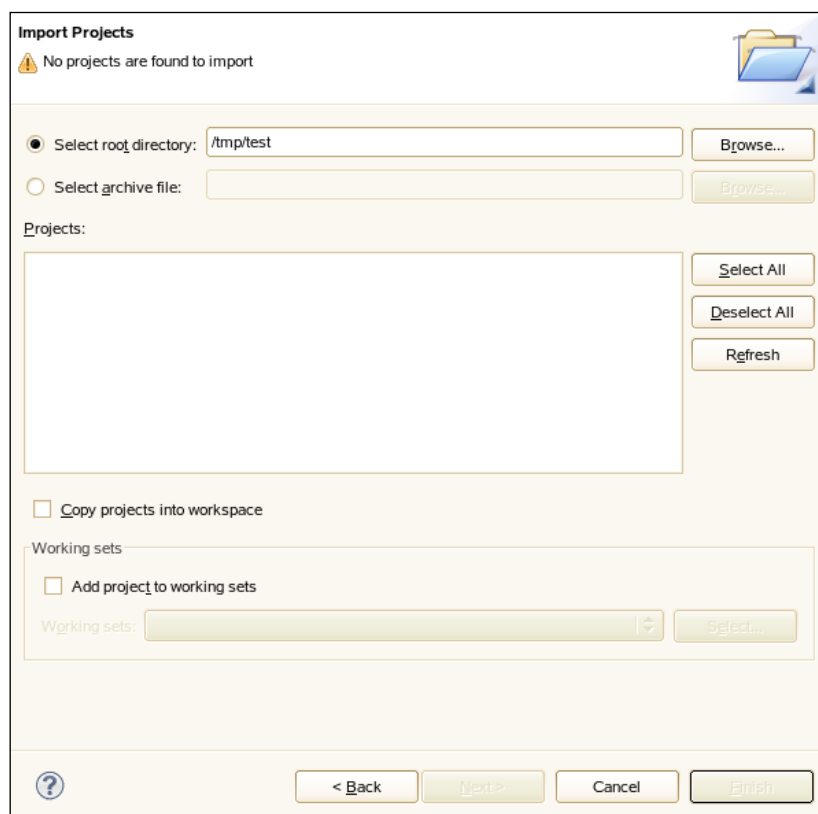


Figure 29: project selection list

3.5.2 Importing projects from version control

Please refer to section 5.12.

3.5.3 Importing regular files into projects

It is also possible to import any kind of files existing in the local file system by selecting one folder in a SPELL project, and then choosing the option "Import from file system" in the import wizard (see Figure 28).

The next page of the wizard allows the user to navigate through the file system, and select an origin folder for importing files. Once the folder is specified, the wizard provides controls to select which files shall be copied into the target folder in the workspace.

See Figure 30 for a snapshot of the wizard page.

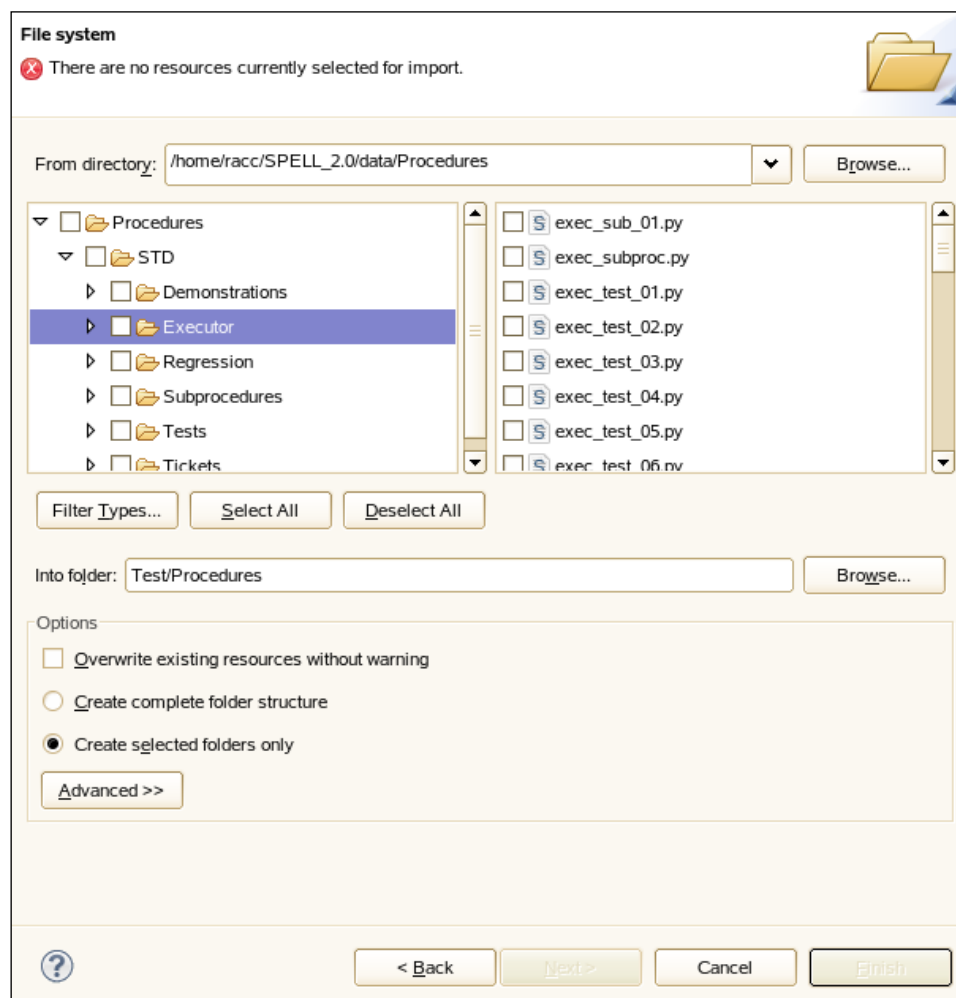


Figure 30: importing files

3.5.4 Exporting files

It is also possible to export files to the local file system, by choosing the menu item “File/Export...” and then choosing the option “Export resources to file system”.

The appearing wizard page is similar: it allows the user to select which resources of the selected SPELL project or folder will be exported, and the location of the destination directory.

4 Preferences

SPELL DEV allows customizing the way procedure source code is presented to the user, as well as it allows to extend automatic code generation mechanism. This can be done through the Preferences dialog. To open it, select the Preferences option including in the Window menu at the top.

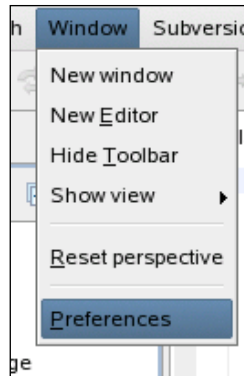


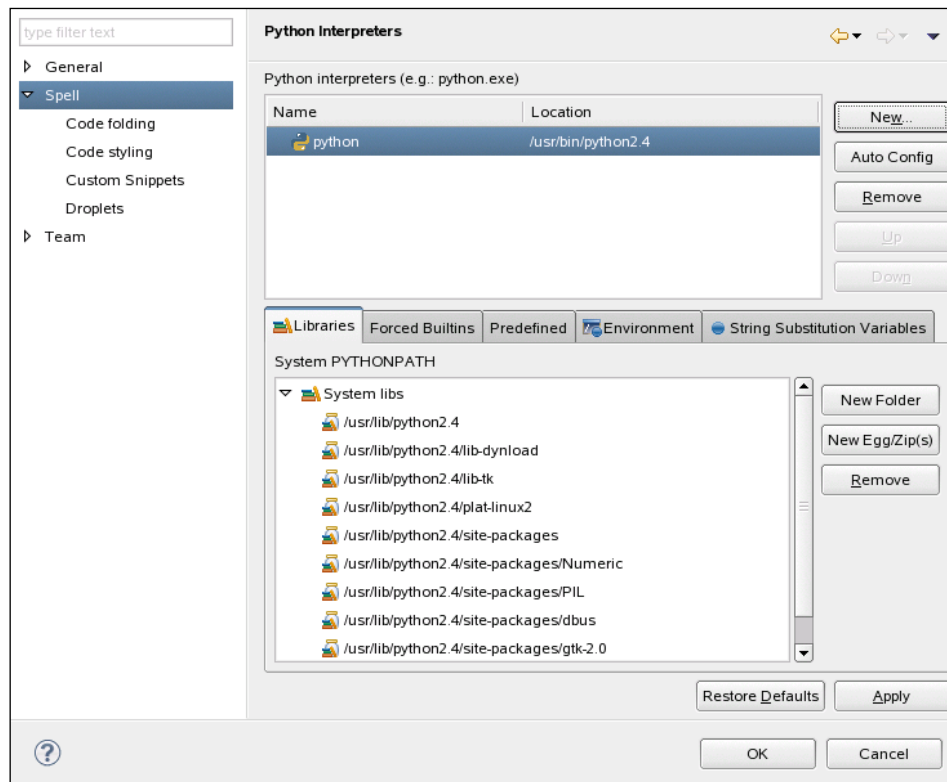
Figure 31. Open preferences dialog

In the preferences dialog, there is a section called Spell where the users can tune up the way they want to use SPELL. That section contains 4 preference pages, plus the main one which is show when the user clicks on the Spell word. Preference pages are presented below, and described in depth in the following sections:

- **Spell:** Allows setting the Python interpreter to use.
- **Code folding:** Allows setting which code blocks can be collapsed and expanded in the SPELL editor.
- **Code styling:** Allows changing the look and feel of the SPELL language tokens inside the SPELL editor.
- **Custom snippets:** Creates custom pieces of code that users insert frequently in SPELL procedures.
- **Droplets:** Modifies the generating code when drag and drop is performed over the SPELL editor.

4.1 Python interpreter preferences page

As explained before SPELL DEV needs a python interpreter in order to provide features such as the syntax checking or filling the outline view with contents. This page helps the user setting the python interpreter to use for SPELL procedure edition.

**Figure 32. Python interpreter preference page**

On this preference page there are two ways of selecting the interpreter to be used. The easiest way to do it is by pressing the "Auto Config" button at the top right side of the page. With this action, SPELL DEV will automatically lookup the default interpreter in the operating system, and will use it as the working interpreter. The more complex way is done by pressing the "New Interpreter" button, and browse on the system directories to look for a Python interpreter.

Once the python interpreter has been found, a new dialog is shown, where the user can specify the additional folder will be added to the system PYTHONPATH while working with the selected interpreter. By adding a folder to the python path, any contained python package or module can be imported from any on-edition procedure. After pressing the "OK" button, the preference page shows the added interpreter.

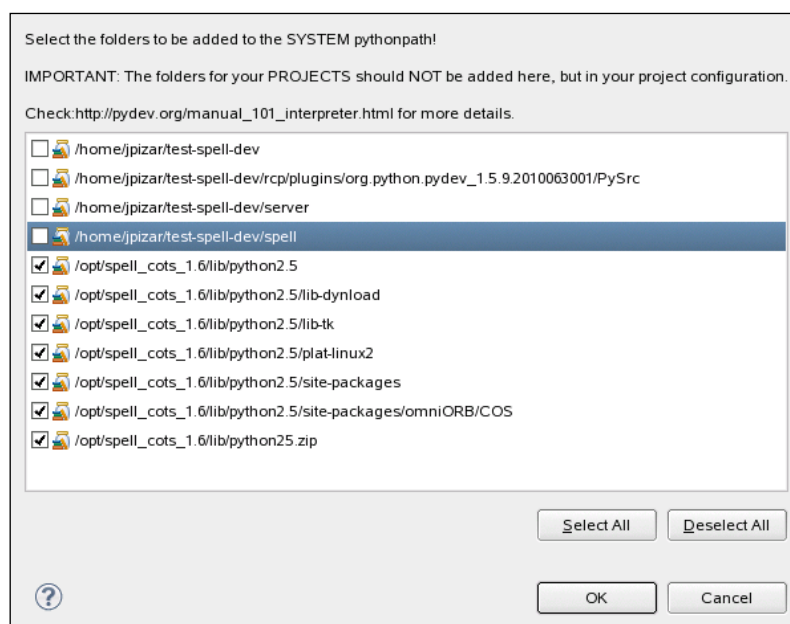


Figure 33. Python path interpreter settings

From this page new interpreters can be added, so users can select the interpreter to use depending on the project they are working on. Also existing interpreters can be removed, made the default interpreter by putting them the first in the list, or modify their Python path settings.

4.2 Code folding preferences page

When a SPELL procedure is being edited, some code regions can be collapsed in order to make them easier to read and find specific sections inside the code. The regions that can be collapsed can be configured from the Code folding preferences page.

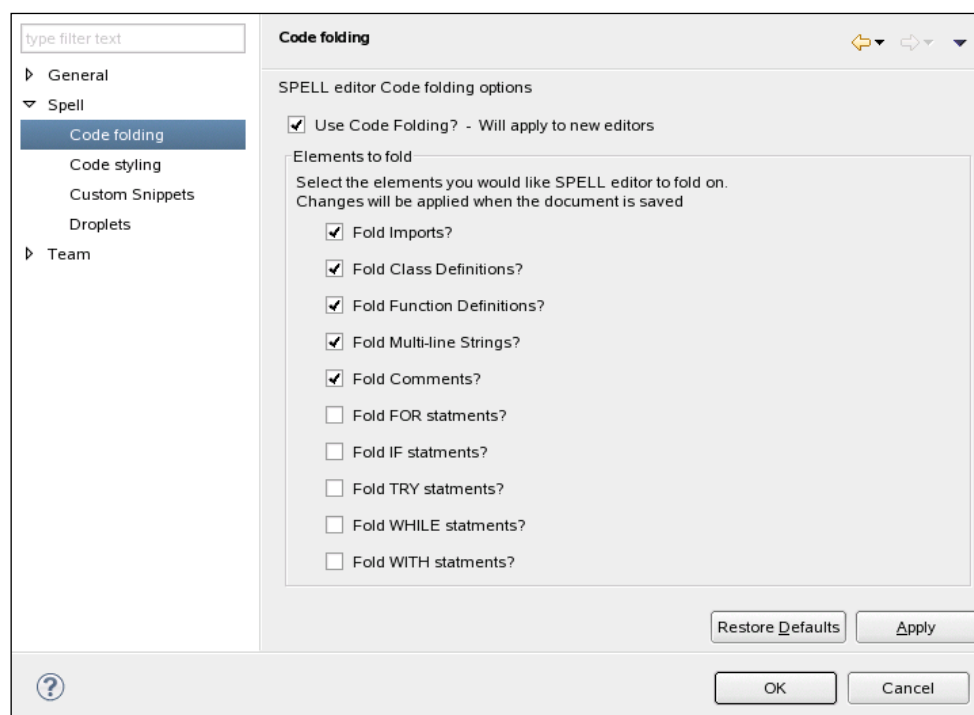


Figure 34. Code folding preferences page

As shown in the image above, code blocks for conditional statements, loops, or multi-line strings can be selected. By pressing the “Apply” or the “OK” buttons, the selected options will be stored, so when a new SPELL editor is opened, the selected elements can be collapsed and expanded by selecting the tiny “+” icon shown at the left of each region.

4.3 Code styling preferences page

SPELL code tokens and reserved words are shown in a default style, which can be changed from the Code styling preferences page. In this page, a sample piece of code is shown, and code tokens can be to change their presentation.

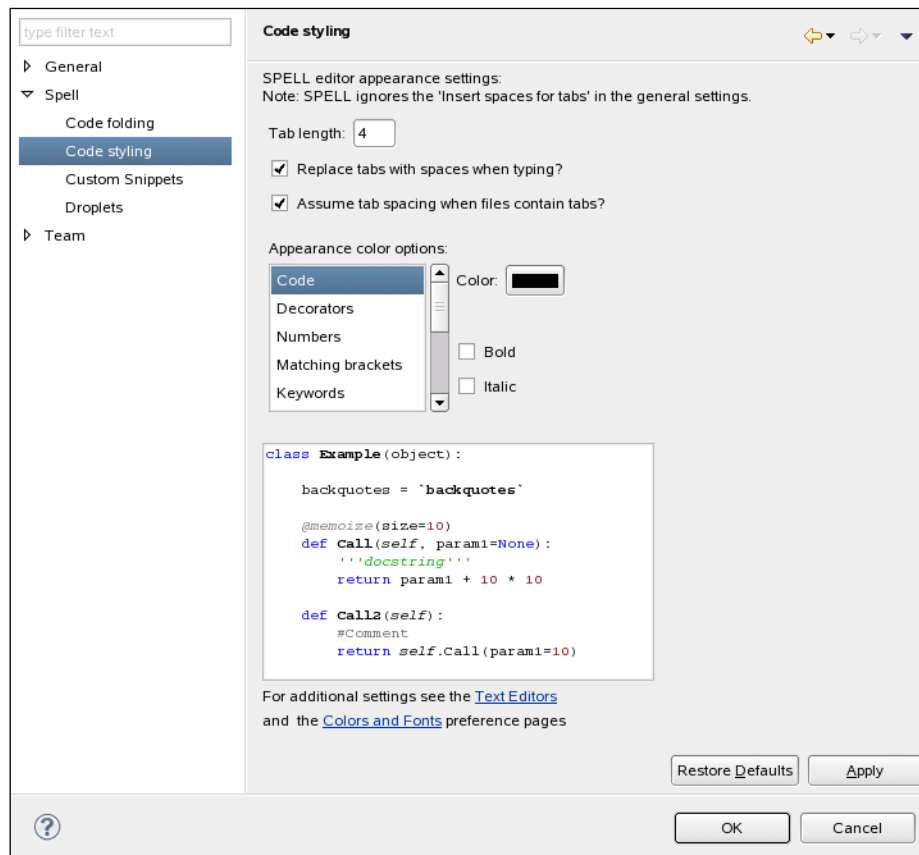


Figure 35. Code styling preferences page

For every code element, its foreground colour, and font weight (bold, italic, or both) can be selected. Also options for conversion between Tab keys and spaces are shown in this page. When the user clicks on the “Apply” or the “OK” button, preferences are stored, and they will be applied next time a SPELL procedure editor is opened.

4.4 Custom snippets preferences page

When procedure developers need to insert a piece of code several times along the different procedures, SPELL DEV provides a mechanism for easing the way this code blocks can be inserted. Custom pieces of code can be defined inside the Custom snippets preferences page.

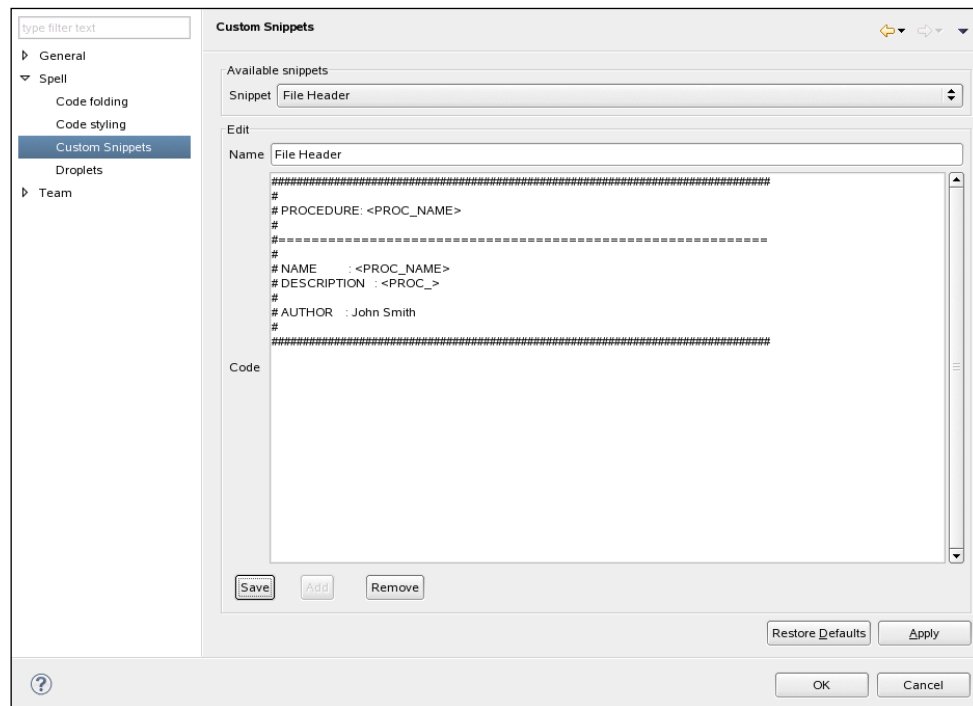


Figure 36. Custom snippets preferences page

In this page any piece of code can be written and saved with a unique name. The same way existing snippets can be modified or deleted. When the users save this page and exit from the preferences dialog, Custom snippets can be found in the snippets toolbar, and their code will be inserted by selecting the option.

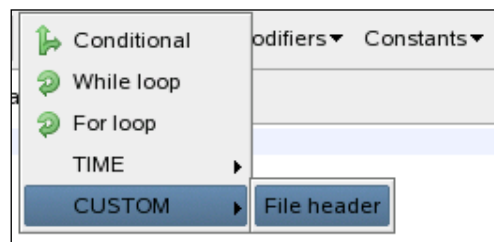


Figure 37. Custom snippets option

4.5 Droplets preferences page

The procedure editor can generate code automatically by dropping elements from the Telemetry and Telecommand views over the editor area. The code to generate can be customized through the Droplets preference page.

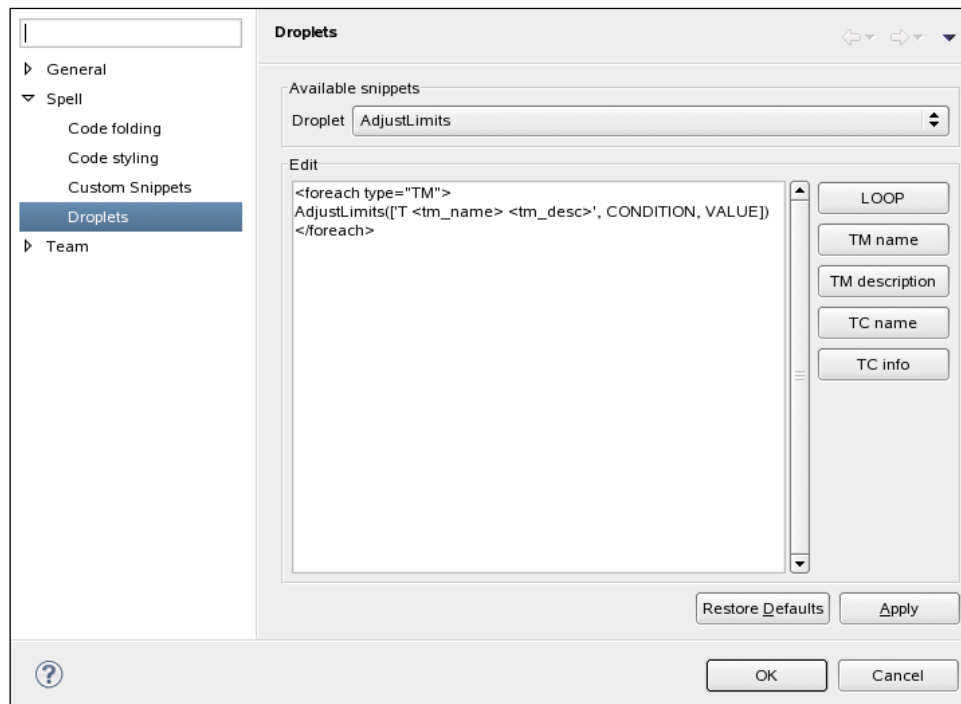



Figure 38. Droplets preferences page

At this page, existing Droplets can be selected, and their generated code is shown. As the image above shows, code is presented in an XML-like syntax, where each element can be substituted by one

- **<foreach>**: Defines a loop which will iterate over the elements. This tag has a mandatory attribute named "type" which defines the set of elements over the loop will iterate. Its value can be TM for telemetry parameters or TC for telecommand elements. It also can have another attribute whose name is separator. Its value can be any string which will be used for separating code blocks generated for each iterated element.
- **<tm_name>**: Inserts the telemetry parameter name
- **<tm_desc>**: Inserts the telemetry parameter description
- **<tc_name>**: Inserts the telecommand name
- **<tc_info>**: Inserts the telecommand description and arguments, if the command has.

03 February, 2015	SPELL Development Environment Manual	
Page 43 of 57	File: SPELL - Development Environment Manual - 2.4.4.docx	

5 Version control systems

5.1 Introduction to version control systems

Version control systems are tools which allow users to track all the workflow and changes over the files included in a project. This allows users, who can be separated in space or time, to collaborate by working on the same project.

Developers can work concurrently on the same project within their own working copy, and send their modifications over the resources to the system server, which maintains a copy up to date of every file in the project, as well as it tracks changes made to them along the time.

Control version systems also allow developers to compare resources between different revisions or dates, or even get a 'photo' of the whole project at one revision.

5.2 Actions on resources

5.3 Checkout

To create a working copy of the project the developers is working on, a checkout has to be made. This action is made once per working copy.

When making a checkout, an up to date working copy of the project is downloaded from the server to the development computer. Once the checkout is finished, developers are ready to work on the project.


5.4 Update to the latest version

When updating a working copy, all the files included in the project are set to the latest version. It is convenient to make an update of the working copy each time the developer is starting to work on the project.

It may happen that changes included in a newer revision don't merge with the changes made in the working copy. Then a conflict is produced. To solve this, we must open the file, solve the conflict manually, and the mark it as resolved for being able to commit our changes.

5.5 Commit changes

When changes are made to files, and those changes are ready to be published to other developers, they have to be committed. Changes committed will be sent to the server along with some notes about changes made. This note will be used for logging purposes.

03 February, 2015	SPELL Development Environment Manual	
Page 44 of 57	File: SPELL - Development Environment Manual - 2.4.4.docx	

Once the commit has been done, other developers can update their working copy to get the latest version of the changed files.

5.6 Revert changes

When a file in the working copy is reverted, all its changes are removed and its contents are restored to the working copy's current revision.

5.7 Lock files

When a developer locks a file claims to have exclusive access to a file, avoiding that some other developers make changes over that resource. It is useful for avoiding concurrent work over the same file, which may lead into changes that may not be merged.

A developer can only lock a file if it has not been previously locked by another one. Locked files by one user cannot be committed by others.

Once the developer who locked a file has finished working on it, he must unlock it. This allows other users to work on it, either by acquiring the lock or even without making a lock over the file.

5.8 Add files to version control

If a new file needs to be added to the project, before committing it there is need to schedule it to be added in the repository next time a commit is done.

5.9 History of changes

As the server tracks the changes made to every file, developers are allowed to check the changes made to them since they were added to the repository, specifying the developers who made every change and the notes added by them explaining every change.

5.10 Compare with revision

Version control systems allow developers to compare changes between different revisions. This is useful for comparing the version inside the working copy against an older one, or even to old revision to check changes made.

5.11 CVS inside SPELL DEV

SPELL DEV includes already a version control system (Subversion), allowing procedure developers to work concurrently over the same project. Actions explained before can be performed by using the toolbar or the Subversion menu at the menu area. Shortcuts for these actions can also be reached by making right click over the files in the procedure explorer, along with some extra features that may be helpful for the developers. They can be found in the Team, Compare with and Replace with menus.



Figure 39. SPELL DEV CVS toolbar

Figure 13 shows the CVS toolbar in the menu area, which provides shortcuts to basic CVS actions: show history, add, update, unlock, lock, revert and commit.

5.12 Project checkout

To create a working copy of the project located in the server, follow these steps:

1. In the Procedure explorer view, make right click and select the Import... option
2. In the dialog, expand section SVN, and select "Checkout projects from SVN" option. Press Next button. A checkout wizard will appear.

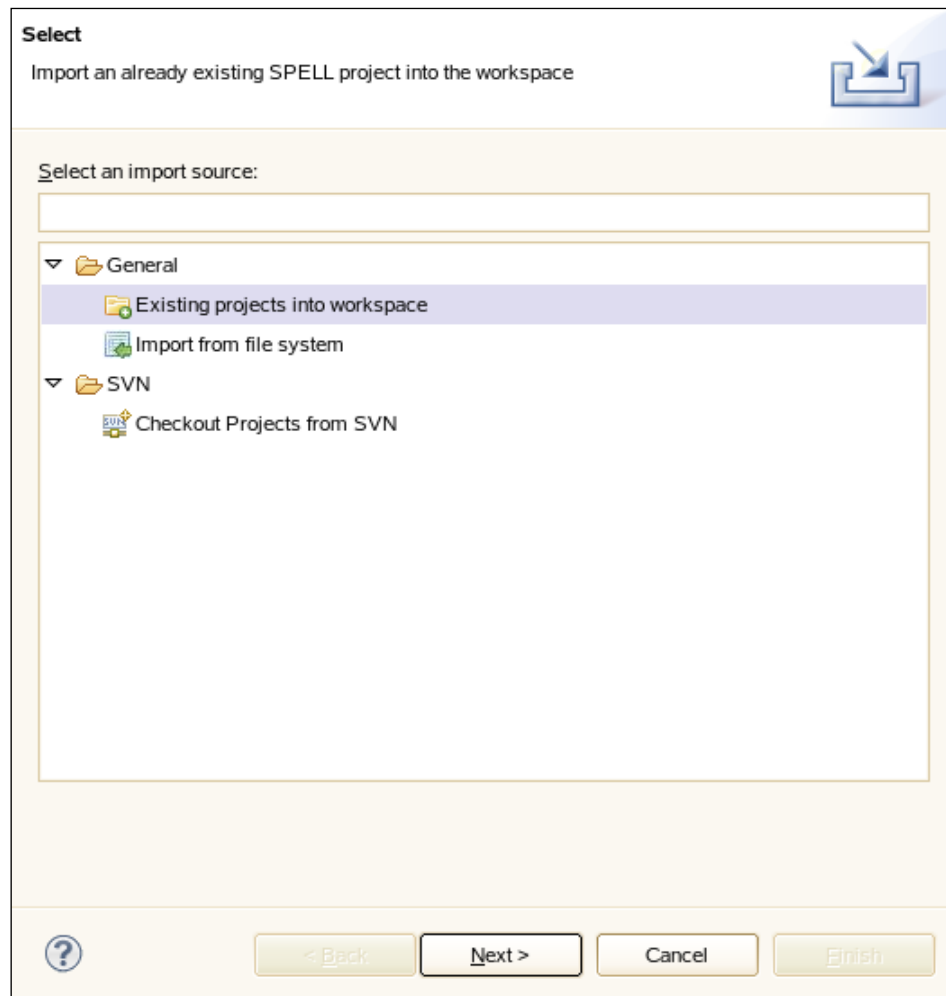


Figure 40. Checkout project option

3. The first page of the wizard will ask for a valid URL where the project is located. If another checkout was previously made, the URL will be shown to select it, unless the checkout wants to be made from another repository.

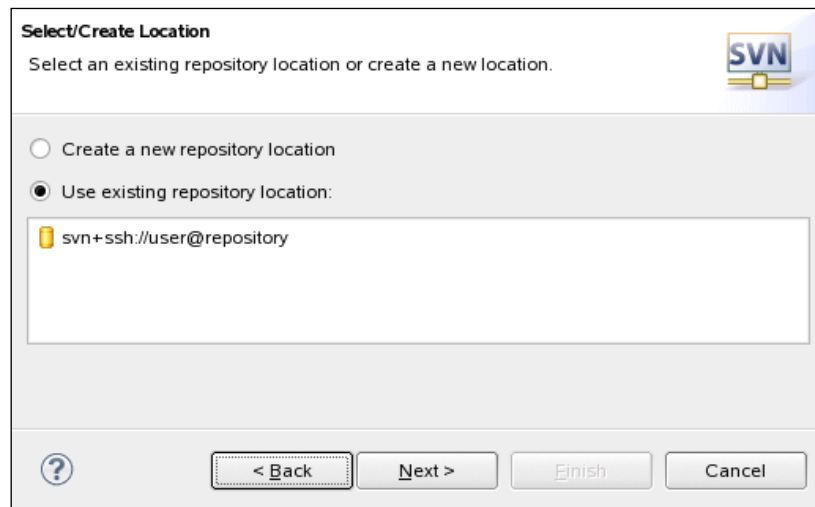


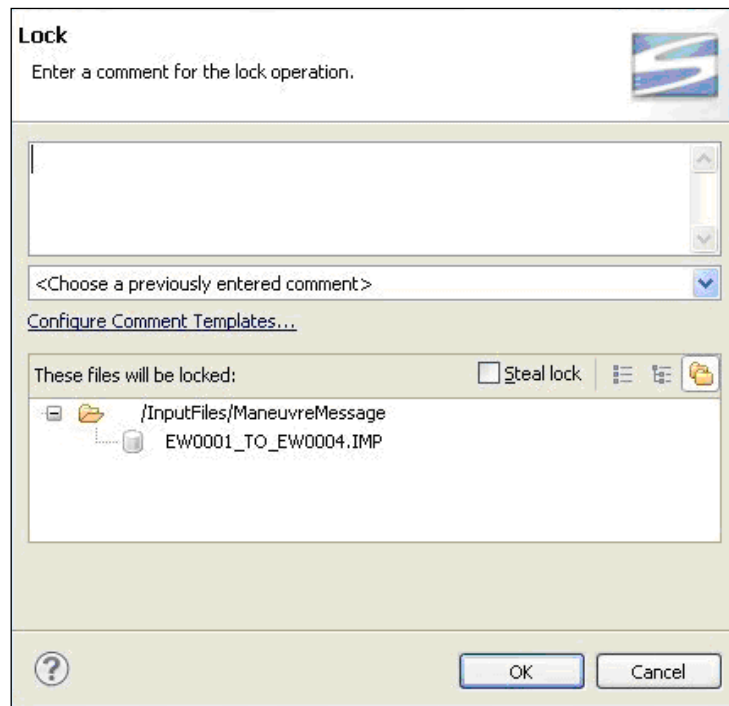
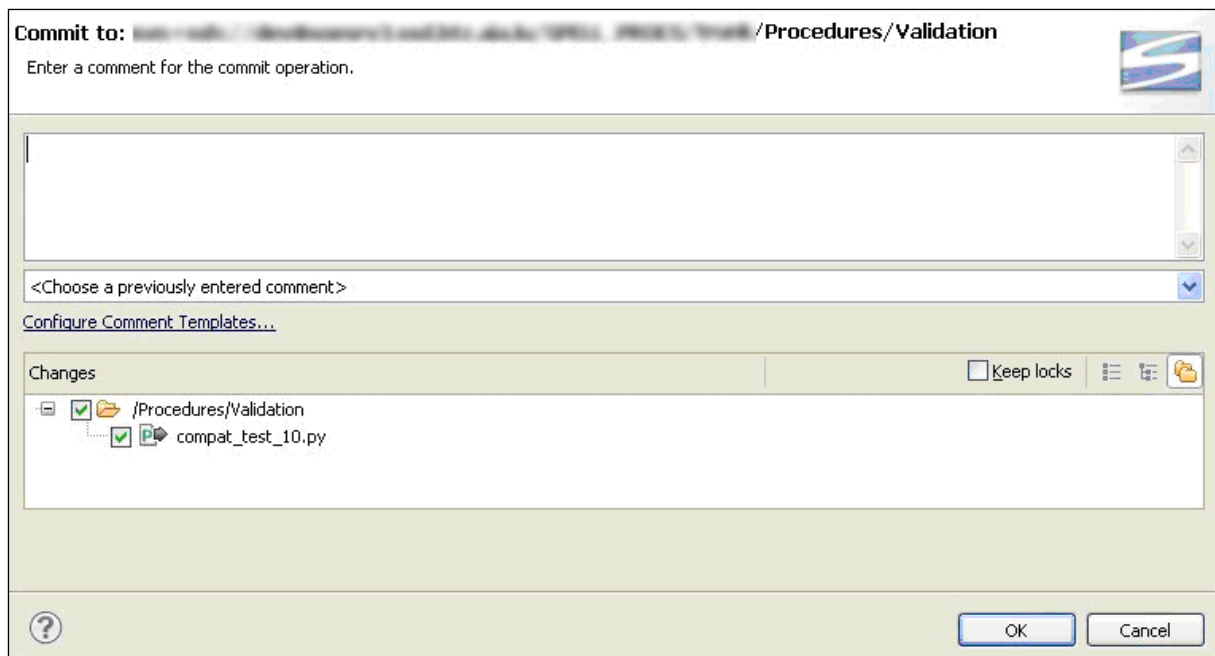
Figure 41. Checkout project from existing repository

4. Next, the wizard shows a tree with the folder contained in the given URL. Select the one which contains the target project.
5. Pressing the Finish button will download a copy of the project at the latest revision.

5.13 Committing and Locking files

When trying to commit some files, or even locking them, a dialog will be shown to the user. This dialog shows a little text editor, as well as the files which the action will apply over.

In the text area, developers must enter some notes about the reason of the commit or lock action, while in the files selection area, we can deselect some files if finally there is no need to apply the action over them.

**Figure 42. SPELL DEV lock dialog****Figure 43. Commit dialog**

5.14 Solving conflicts

After updating the working copy conflicts may appear, and the developer is asked to solve them manually. Conflicts can be detected because in the procedure explorer views new files with extensions `.mine` and `.rXXXX`, where XXXX is the revision number which conflicts with our working copy, appear. For example, if `file.extension` causes conflict, there should be some new files in the same folder with these names:

- `file.extension.r1234`
- `file.extension.r1235`
- `file.extension.mine`

This means that our local file conflicts with the changes made at revisions 1234 and 1235.

To solve the conflict, open the file which caused the conflict. The conflictive region is delimited by special tags indicating which code part is from our working copy, and which other are from other revisions. For example:

```

<<<<<<< .mine

Changes made in my working copy

=====

Revision 1234 changed something that can't be merged

>>>>>>> .r1234

```

This piece of source code shows the region where our working copy file conflicts with revision 1234. Local and revised regions are separated by `=====`

To make conflict resolved, just modify the source code in a way that makes sense to your procedure or database file. Then save the file, and mark conflicting file as resolved. To make this, make right click on the file in the Procedure explorer, and in the context menu put the mouse over Team group and then select the Mark Resolved option. When a conflict is marked as resolved, `.mine` and `.rXXXX` files are removed automatically.

5.15 Comparing with previous revisions

It is possible to compare local copies of the files inside a project with changes made to it along the different revisions. To compare between different revisions, follow these steps:

1. In the Procedure explorer view, make right click over the file you want to compare.
2. When the context menu appears, put the mouse over the Compare with option, and then select the Revision... option. A special editor showing all the revisions who affected the selected file will appear.
3. Select the revision you want to use for comparing by making double click on it.
4. At the bottom of the editor the local copy at the left and the revision copy at the right will be shown.

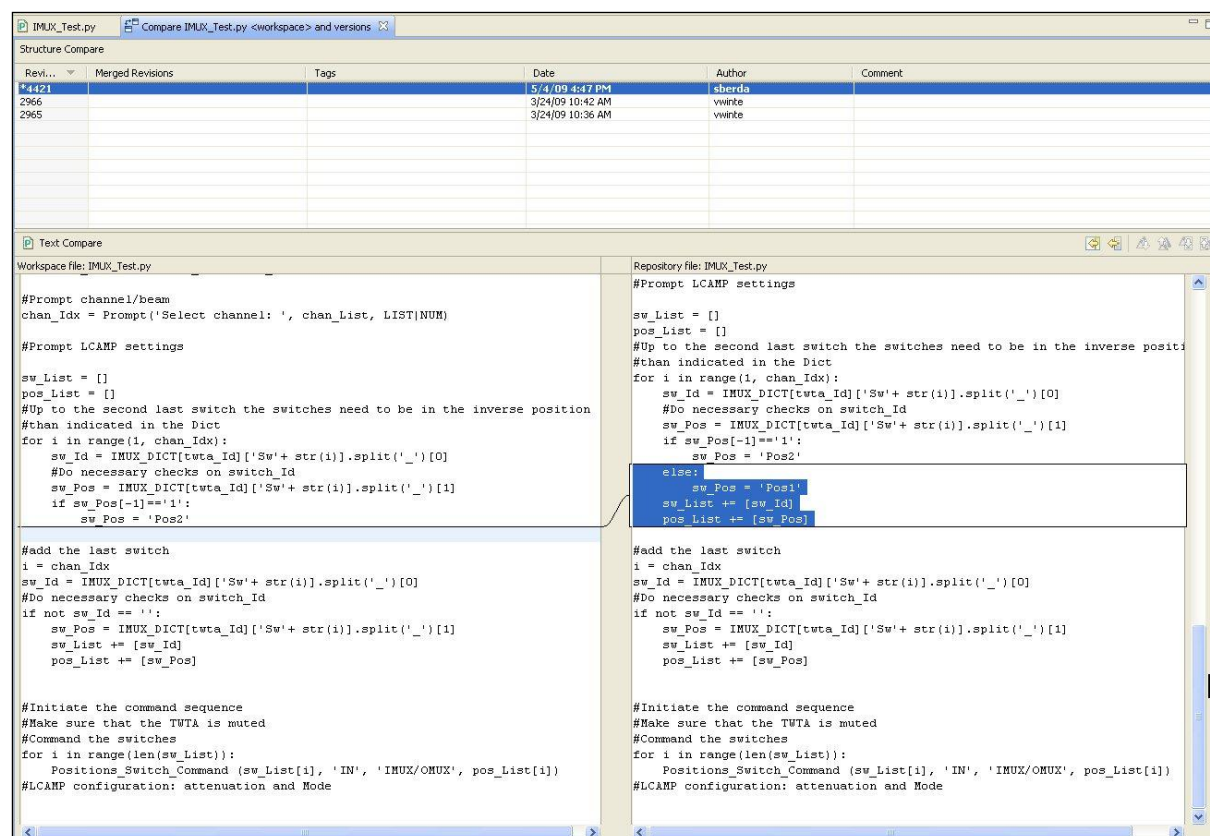


Figure 44. Compare with editor

Figure 24 shows a compare with editor with a sample procedure. In local copy some source code lines have been removed from the latest revision. If we select to compare with that revision, which is the one at the top, two editors appear as explained in step 4, showing differences between both versions.

5.16 Automatic CVS actions over new procedures

When a new procedure is being created, the procedure creation wizard has a page where the user can set if the new file must be control versioned, as well as some additional options.

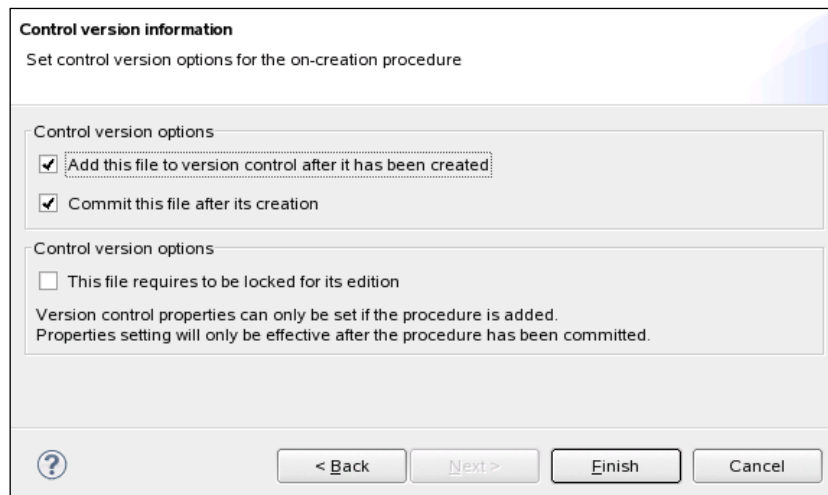


Figure 45. Control version page in the New Procedure wizard

Once the user enters the procedure information and location, if the Next button is pressed the control version page is shown. On this page, three options are available to the users:

- **Add this file to version control:** File will be version controlled when this file is created.
- **Commit this file:** File will be committed to the repository. This option is available is the option above is selected, as a file can't be committed to the repository unless it has been previously added.
- **File requires lock for its edition:** If this option is selected, the new file will have a subversion property which makes it read-only by default. If any user wants to edit it, there will be need of locking the file for its edition.

This page values will only be applicable on procedure whose containing project is already under version control. Otherwise entered values are ignored.

6 Semantic Checker

6.1 SPELL procedure Semantic Check

In addition to the syntax check, SPELL DEV provides an additional set of checks to ensure SPELL procedure code correctness. Since Python (and by extension, SPELL) language is dynamic and very flexible, it becomes a need to use additional checks in order to ensure the correct usage of SPELL functions and the consistency of the procedure code.

These additional checks compose the “semantic check”. Rather than analyzing the syntactic correctness of the code, they analyze the way the SPELL services are used and the way the procedures are coded.

The semantic checker is an offline tool, meaning that it is executed in the SPELL DEV application without actually executing the procedures. It is a parser-based mechanism, not interpreter-based. That means that the check has some limitations: some of the rules cannot be analyzed when the developer uses variables to, for example, provide function arguments. Since the content of the variables is defined at runtime, the system cannot determine the contents of the variable during the parsing. Nevertheless, running the semantic checker offline has other advantages: there is no need to setup a testing execution environment, and all the setup required for it. As a result, the semantic check can be done simply and quickly for the whole set of procedures without requiring extra effort (no S/C simulator required, no GCS setup required, etc.)

6.2 Types of checks

The semantic check package comprises a variety of rules, including:

- **Goto consistency:** all `Step` and `Goto` calls in a procedure shall be consistent. For example, there shall not be a go-to instruction jumping to a `Step` which is not defined in the procedure.
- **Function call consistency:** there should not be calls to functions which have not been defined in the procedure. And the other way around, there should not be functions defined in the procedure which are never used.
- **Function argument consistency:** the amount of arguments used on function calls shall be consistent with the function definitions. Value types cannot be checked due to the dynamic nature of the SPELL language, though.
- **Header consistency:** the correctness of the procedure header fields is checked.
- **SPELL modifiers:** for every SPELL function call, there is a set of recognized or allowed modifiers that can be used. The semantic check ensures that all modifiers given to a function are meaningful and consistent. When possible, the modifier values are checked.
- **SPELL function arguments:** when possible, the correctness of the arguments passed to SPELL functions is checked. Of course, this cannot be done if variables are used, since their actual content is defined at runtime.
- **TM/TC database consistency:** when the containing SPELL project has a TM/TC database associated (see section 3.1.3.1) database-related checks can be performed as well. These checks ensure that the TM points and telecommands used in the procedure do exist in the database, and that the TM point values, telecommand arguments and telecommand argument values are consistent with the definitions.

6.3 Semantic checker interface

The semantic check is not performed automatically by SPELL DEV, but on demand. The user shall select one or more procedures from the workbench, and trigger then the check manually.

The semantic check is controlled via the menu “Semantics” or via the semantics toolbar.

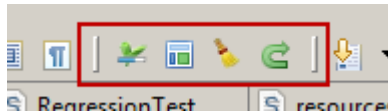


Figure 46: semantics check toolbar

The toolbar (see image above) contains four buttons:

- Open the semantic check start dialog
- Generate semantic check report files (for after a semantic check has been performed)
- Clean semantic check results (for after a semantic check has been performed)
- Re-parse user library

Same options appear in the "Semantics" menu:

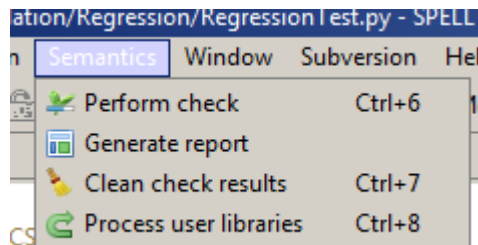
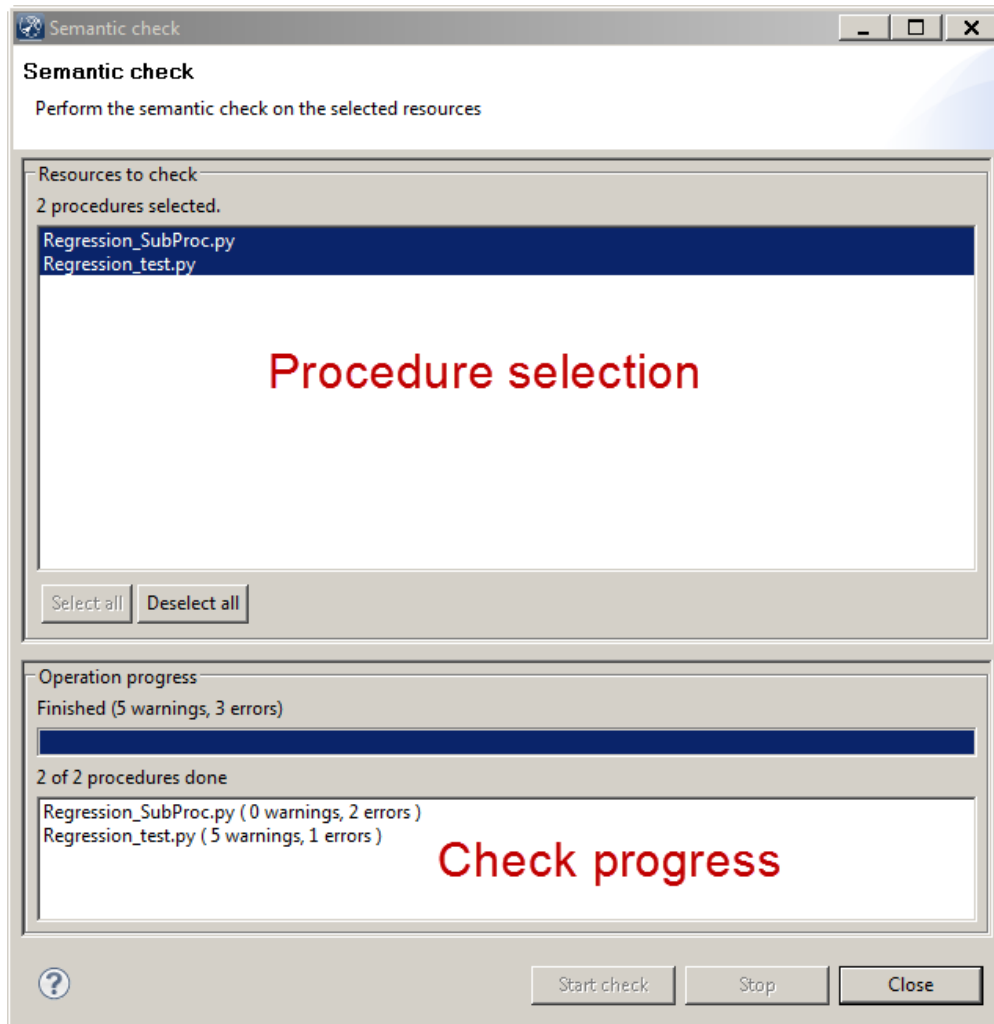


Figure 47: semantics check menu

Both “Perform check” and “Clean check results” actions require selecting at least one resource (project, folder inside project, or procedures) in the Procedure Explorer view to work. The first one starts the check on all selected resources, and the second one cleans up the check results.

6.3.1 Perform a semantic check

The "Perform Check" action will bring up the Semantic Check configuration dialog.



In this dialog, and before the actual check process is started, the procedures to be taken into account can be selected. When a project is selected before "Perform check" action is used, all procedures within the project are gathered. When a folder is selected in turn, all the procedures within that folder are taken.

Initially all the procedures contained in the selected resource will be included in the Semantic Check dialog list.

By pressing the "Start check" button the process will be started, and it can be interrupted by using the "Stop" button.

The second half of the Semantic Check dialog shows progress of the check process. The list at the bottom shows the procedure files together with a summary of found errors and warnings. The files appear progressively, as soon as their check is finished. By double-clicking on any procedure name on the progress list of the Semantic Check Dialog, the corresponding procedure editor will be open.

Note that this dialog provides information about the check process but it does not provide full insight on the semantic errors and warnings: for this, the **Problems View** is used (see "Check results" section)

6.3.2 Generate a check report

The “Generate report” action creates sets of text files containing all the detected issues organized per procedure. The files are created in a folder chosen by the user.

The “Process user libraries” action is provided in order to trigger the model refresh of the project user library files. For the sake of performance, the user libraries are NOT processed every time a procedure is analyzed. They are parsed only once, at the moment of the first check since the SPELL DEV application was started.

Therefore, if the user libraries are changed, it is necessary to (a) restart the SPELL DEV application, which is not really convenient normally, or (b) trigger the explicit processing of the libraries using the described action.

When working on a specific SPELL procedure editor, the semantic check is automatically triggered when saving the procedure modifications, *provided that the check has been performed already once* on that procedure. That is, the semantic check is repeated on the saved procedure only if it has been used before on it.

6.4 Check results

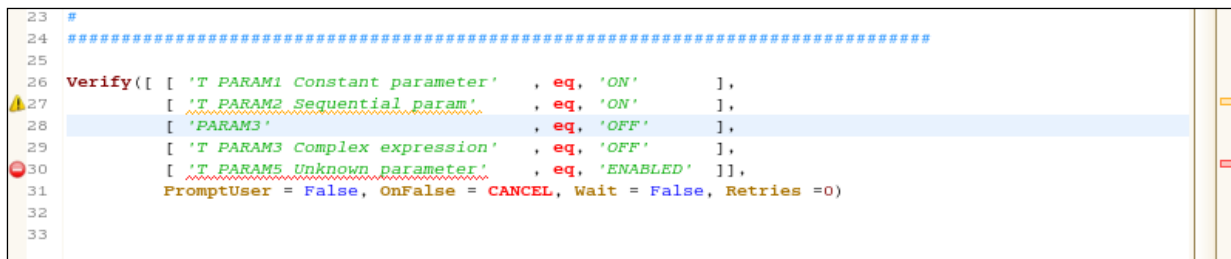
The result of the semantic check consists on a set of *problem markers*, entities that are shown in different places of the workbench:

- In the Problems View, as a list of errors and warnings associated to procedure files (resources)
- In the editor window, in the side bars on each side. They appear as small colorized markers that point out the error locations, and help the user on navigating the procedure code searching for errors.
- In the editor window, as colorized underlines on the source code.

The results are associated to procedures, and they persist as long as the user does not explicitly clean (remove) them from the workbench, or there is another semantic check that makes the problems to disappear (once they have been corrected by the developer). They also disappear if the SPELL DEV application is closed or restarted.

Problem markers can be warnings (yellow color) or errors (red colors).

The following snapshot shows an example of a procedure editor with some semantic markers on it:



```

23 #
24 #####
25
26 Verify([ [ 'T PARAM1 Constant parameter' , eq, 'ON' ],
27 [ 'T PARAM2 Sequential param' , eq, 'ON' ],
28 [ 'PARAM3' , eq, 'OFF' ],
29 [ 'T PARAM3 Complex expression' , eq, 'OFF' ],
30 [ 'T PARAM5 Unknown parameter' , eq, 'ENABLED' ] ],
31 PromptUser = False, OnFalse = CANCEL, Wait = False, Retries = 0)
32
33

```

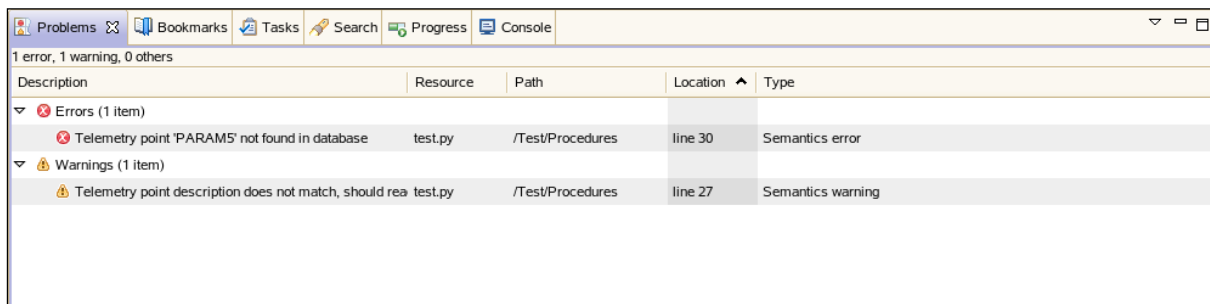
Figure 48: editor with semantics markers

In the example, one error and one warning marker can be seen. Notice that the left hand bar is used to highlight problem positions in the currently displayed source code, whereas the right hand bar is used to show ALL problems in the current procedure.

Both markers (left and right side) show problem information when moving the mouse over them, in the form of a tooltip. The markers on the right hand bar can be also used to navigate to the problem locations, by clicking on them.

Notice also the underlines in the source code, indicating the precise element that is causing the problem.

The following figure shows the Problems View, showing the same problem markers:



Description	Resource	Path	Location	Type
1 error, 1 warning, 0 others				
Errors (1 item)				
Telemetry point 'PARAM5' not found in database	test.py	/Test/Procedures	line 30	Semantics error
Warnings (1 item)				
Telemetry point description does not match, should rea	test.py	/Test/Procedures	line 27	Semantics warning

Figure 49: Problems View showing the semantics problems

It is also possible to navigate to the problem locations by double-clicking on an item of the problem view. It is possible as well, to select items in the Problems View, copy them (via CTRL+C) and then paste the corresponding information on a text file.

The Problems View is native to Eclipse RCP applications, and provides the usual functionalities. Items can be sorted by type, severity, resource or location.

