



SPELL Development Environment Manual

Software version 2.0.5

<i>Author</i>	<i>Date</i>	<i>Version</i>	<i>Comment</i>
J. Andres Pizarro Rafael Chinchilla	24 June 2009	1.0	Document created
J. Andres Pizarro Rafael Chinchilla	24 Nov. 2010	1.1	Update to 2.0 release
J. Andres Pizarro	03 Dec. 2010	1.2	Update procedure creation process description.
Rafael Chinchilla	07 March 2011	1.3	Updated with latest features

Prepared By	J. Andrés Pizarro Rafael Chinchilla	Signature: <i>Signature on file</i>
Reviewed By	Thomas Nowak	Signature: <i>Signature on file</i>
Authorized By	Thomas Nowak	Signature: <i>Signature on file</i>

Table of Contents

1	INTRODUCTION	6
1.1	Purpose of this document	6
1.2	SPELL DEV overview	6
2	STARTING WITH SPELL DEV.....	7
2.1	Launching SPELL DEV	7
2.1.1	Launcher files	7
2.1.2	Workspace.....	7
2.1.3	Selecting the Python interpreter	7
2.2	SPELL DEV workbench.....	9
2.2.1	Views	10
2.2.2	Editors	13
3	WORKING WITH SPELL DEV	16
3.1	SPELL projects	16
3.1.1	Creating a SPELL project.....	16
3.1.2	SPELL project elements.....	17
3.1.3	Setting project properties	17
3.2	Procedures	18
3.2.1	Create a new procedure.....	18
3.2.2	Editing a procedure	19
3.2.3	Column mode	20
3.2.4	Automatic code generation.....	21
3.3	Dictionary files	24
3.3.1	Database file types.....	24
3.3.2	Creating a dictionary file.....	24
3.4	Exporting and Importing.....	25
3.4.1	Importing projects from file system.....	25
3.4.2	Importing projects from version control	26
3.4.3	Importing regular files into projects	26
3.4.4	Exporting regular files.....	27
4	CUSTOMIZING SPELL-DEV	28
4.1	Python interpreter preferences page.....	28
4.2	Code folding preferences page.....	30
4.3	Code styling preferences page	30
4.4	Custom snippets preferences page.....	31
4.5	Droplets preferences page	32


5	VERSION CONTROL SYSTEM	34
5.1	Introduction to version control systems.....	34
5.2	Actions on resources	34
5.2.1	Checkout	34
5.2.2	Update	34
5.2.3	Commit	34
5.2.4	Revert	34
5.2.5	Lock	35
5.2.6	Unlock.....	35
5.2.7	Add to control version.....	35
5.2.8	Show history	35
5.2.9	Compare with revision	35
5.3	CVS inside SPELL DEV	35
5.3.1	SPELL project checkout	36
5.3.2	Committing and Locking files	37
5.3.3	Solving conflicts.....	38
5.3.4	Comparing with previous revisions.....	39
5.3.5	Automatic CVS actions over new procedures.....	39
6	SEMANTIC CHECKER.....	41
6.1	SPELL procedure Semantic Check.....	41
6.2	Types of checks.....	41
6.3	Semantic checker interface	42
6.4	Check results	43

Ref. Documents

[1] SPELL Language Manual – UGCS-USL-SPELL-SUM_08_002_2.4

Acronyms

GCS	Ground Control System
GDB	Ground Database
GMV	GMV Aerospace & Defence (www.gmv.com)
IDE	Integrated Development Environment
MMD	Manoeuvre Message Database
SCDB	Spacecraft Database
SES	Société Européenne des Satellites
RCP	Rich Client Platform
TC	Telecommand
TM	Telemetry

3 July 2011	SPELL Development Environment Manual USL/SPELL Software version 2.0.5 UGCS-USL-SPELL-DEV-SUM_10_003_1.3.doc	
Page 6 of 44		

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 SPELL DEV 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 Starting with SPELL DEV

2.1 Launching SPELL DEV

SPELL DEV is a multi-platform tool: it is available both for GNU/Linux and Windows platforms. In this section the way of launching the application is explained. It is assumed that the application has been correctly installed and configured. 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 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

The first time SPELL DEV is launched, the user is asked to select the *workspace* to work with, before the workbench actually shows up. The workspace is a directory where all SPELL projects and files (procedures and data) will be stored.

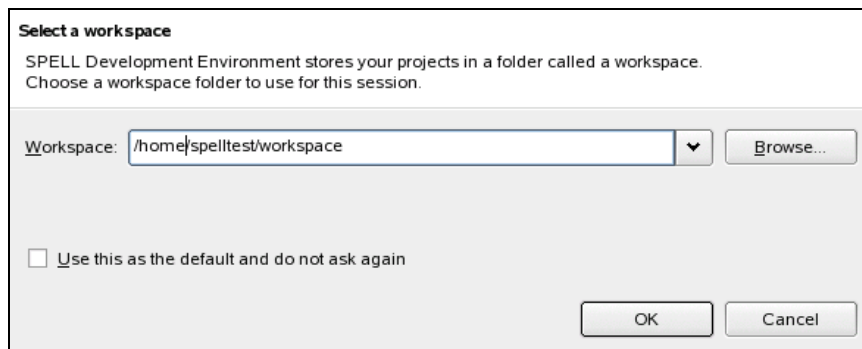


Figure 1. Workspace selection dialog

By default, the application asks the user to select the workspace each time it is started. To avoid this, there is an option for using entered workspace as the default one. Once that option is selected, the user won't be prompted anymore.

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 the application is started it will also ask the user to select the Python interpreter to use.

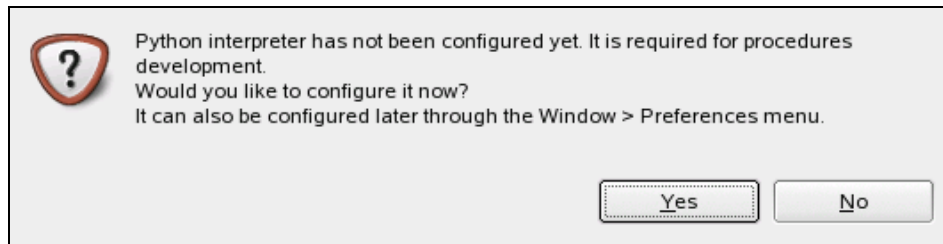


Figure 2. Python interpreter setting question

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:

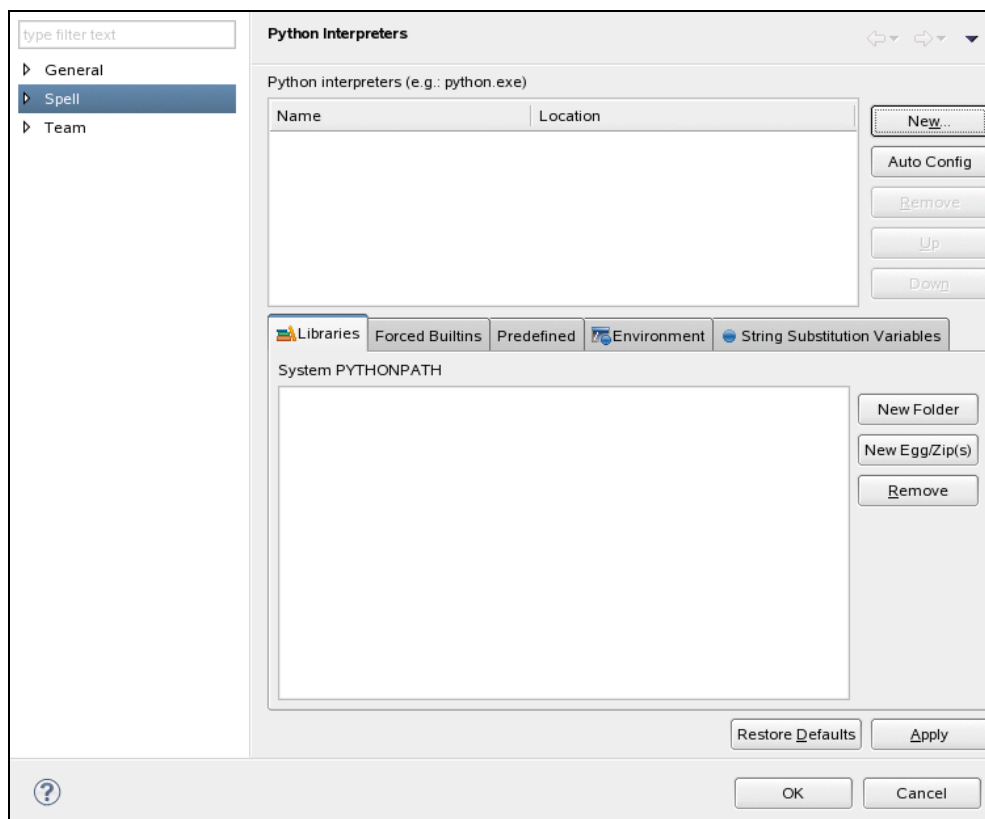


Figure 3. Python interpreter page in preferences dialog

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. The second alternative requires pressing the "New Interpreter" 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.

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 SPELL DEV workbench

When SPELL DEV application is launched, the workbench is shown.

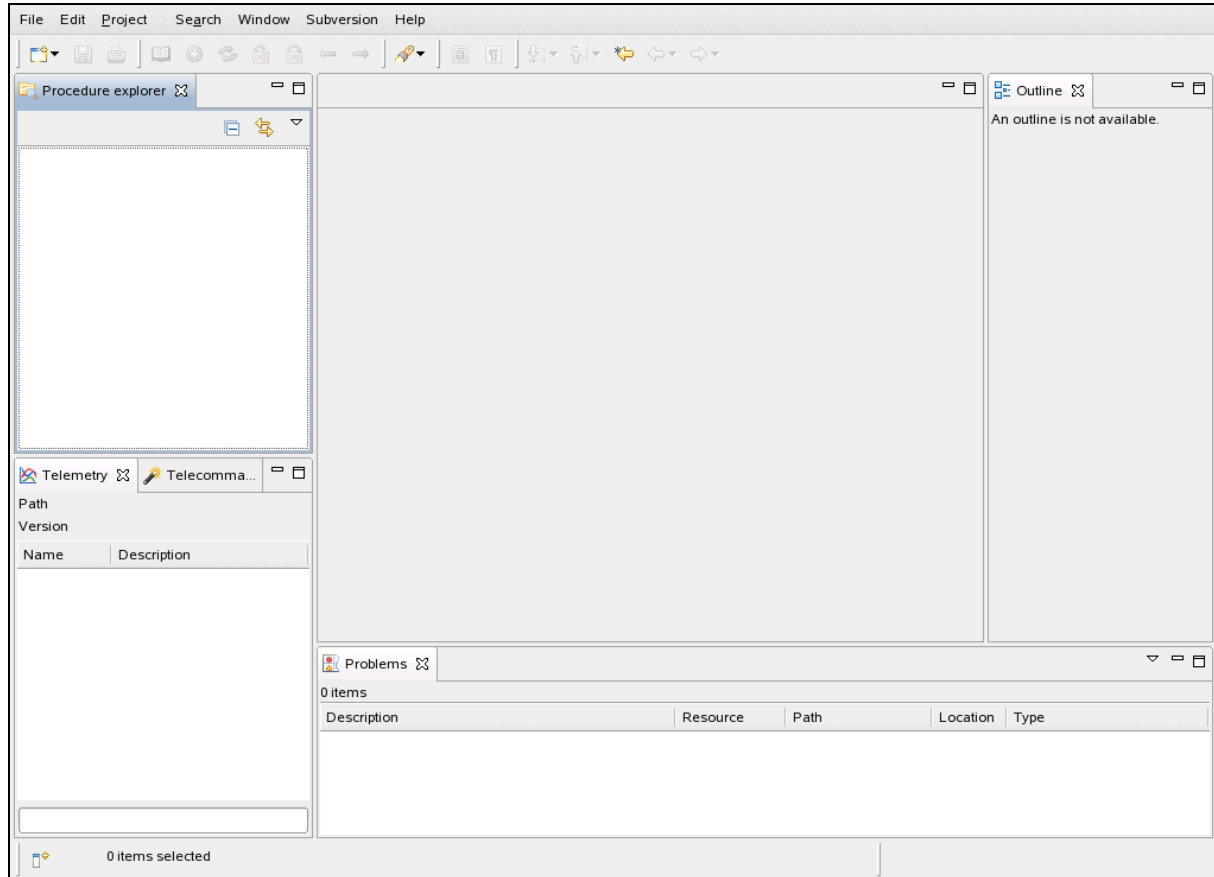


Figure 4. SPELL-Dev default workbench

The application workbench is composed of views, editors and menus/toolbars:

- **Views** windows or sections at the sides of the workbench that provide information and some helpful utilities for the user. In the image above, we can see several views: the Procedure Explorer, the Outline, the Telemetry and Telecommand views, and the Problems view. They will be described later on.
- **Editors** are the areas where developers can work with the SPELL procedures and related files. When the application is open for the first time the editor area, in the centre of the workbench, is empty. Nevertheless, if SPELL-DEV is closed with some procedure editors still open, they will automatically appear again the next time the application is started.
- **Menus & Toolbars** provide access to general actions over the project resources, such as open, save, control version system actions and others.

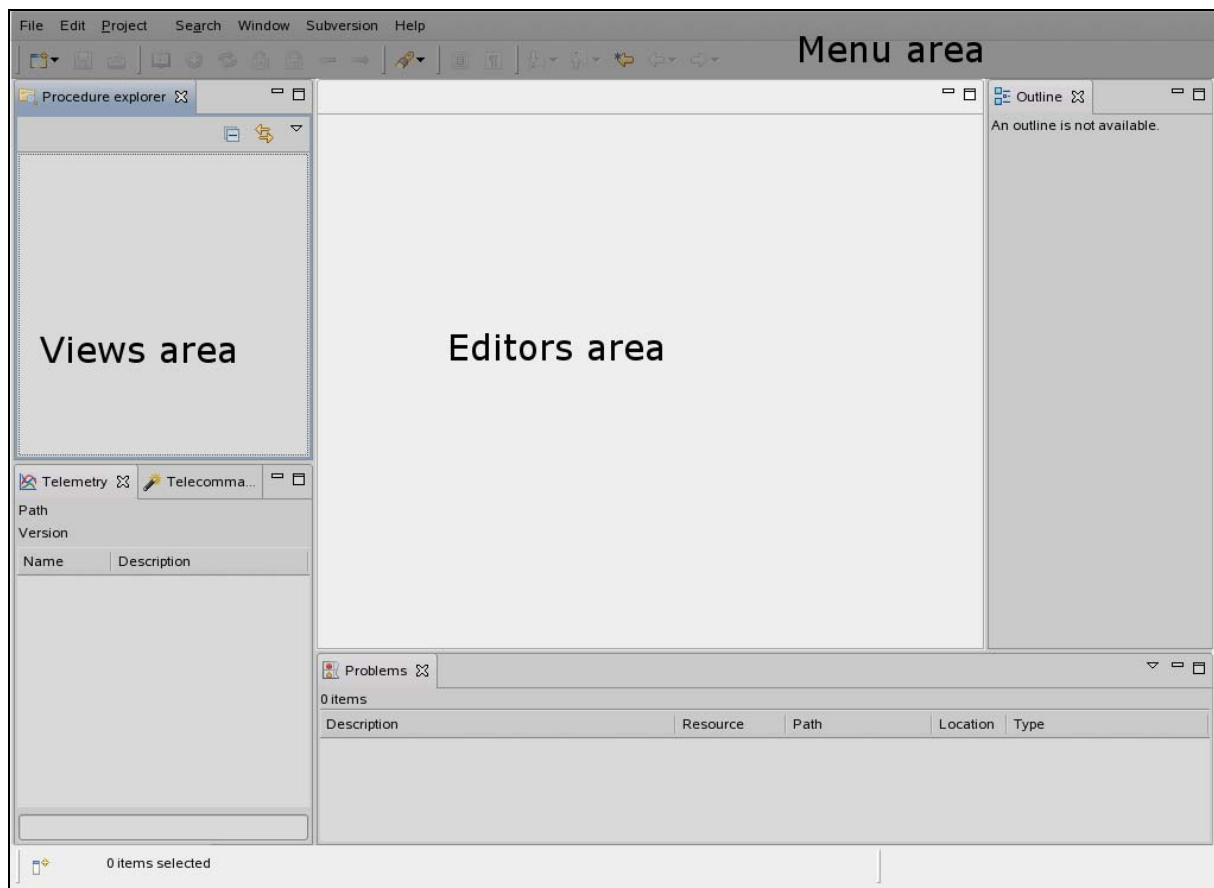


Figure 5. Workbench parts

2.2.1 Views

2.2.1.1 Procedure Explorer

The Procedure Explorer view shows the existing projects in the workspace. The contents of a project are shown in a tree structure, where folders and files are represented with different icons depending on their purpose.

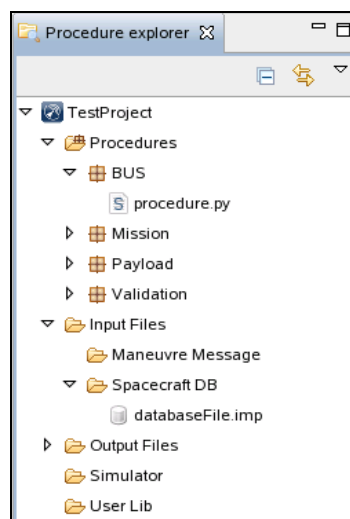


Figure 6. Procedure explorer

Apart from showing the workspace contents, this view allows to perform basic actions such as copy, cut, paste or move of workspace resources. These actions can be performed by using the context menu this view provides or the drag & drop mechanism.

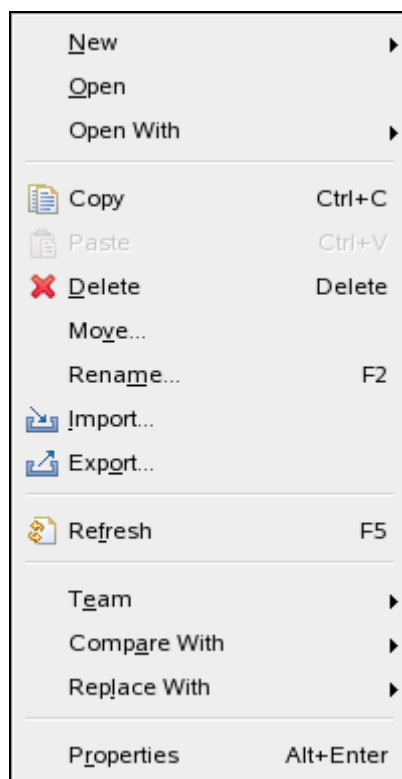


Figure 7. Procedure explorer context menu

IMPORTANT: care must be taken whenever files are moved or copied and SPELL DEV is using version-controlled resources with Subversion. Version-controlled procedures cannot be simply moved from one directory to another, but the working rules and procedures of Subversion for copying and moving files must be observed.

2.2.1.2 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.

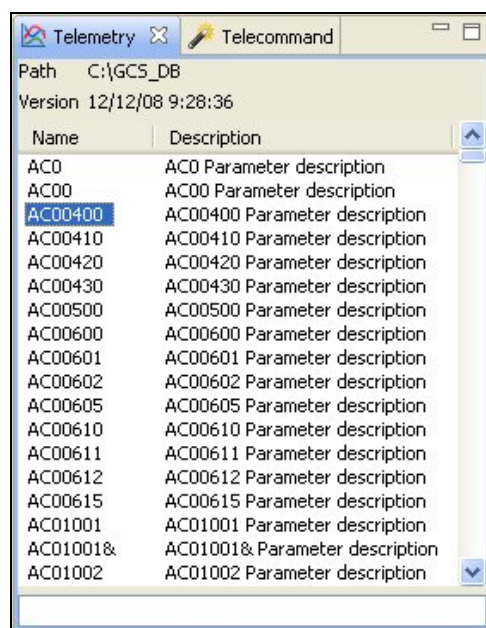


Figure 8. Telecommand and telemetry views

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](#).

2.2.1.3 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)

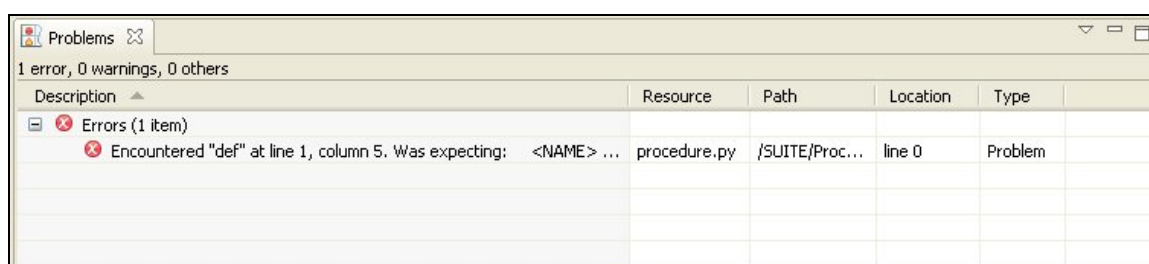


Figure 9. Problems view

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 shown 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.

2.2.1.4 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.

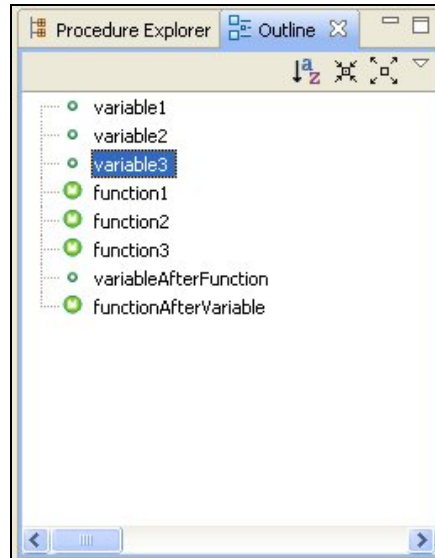
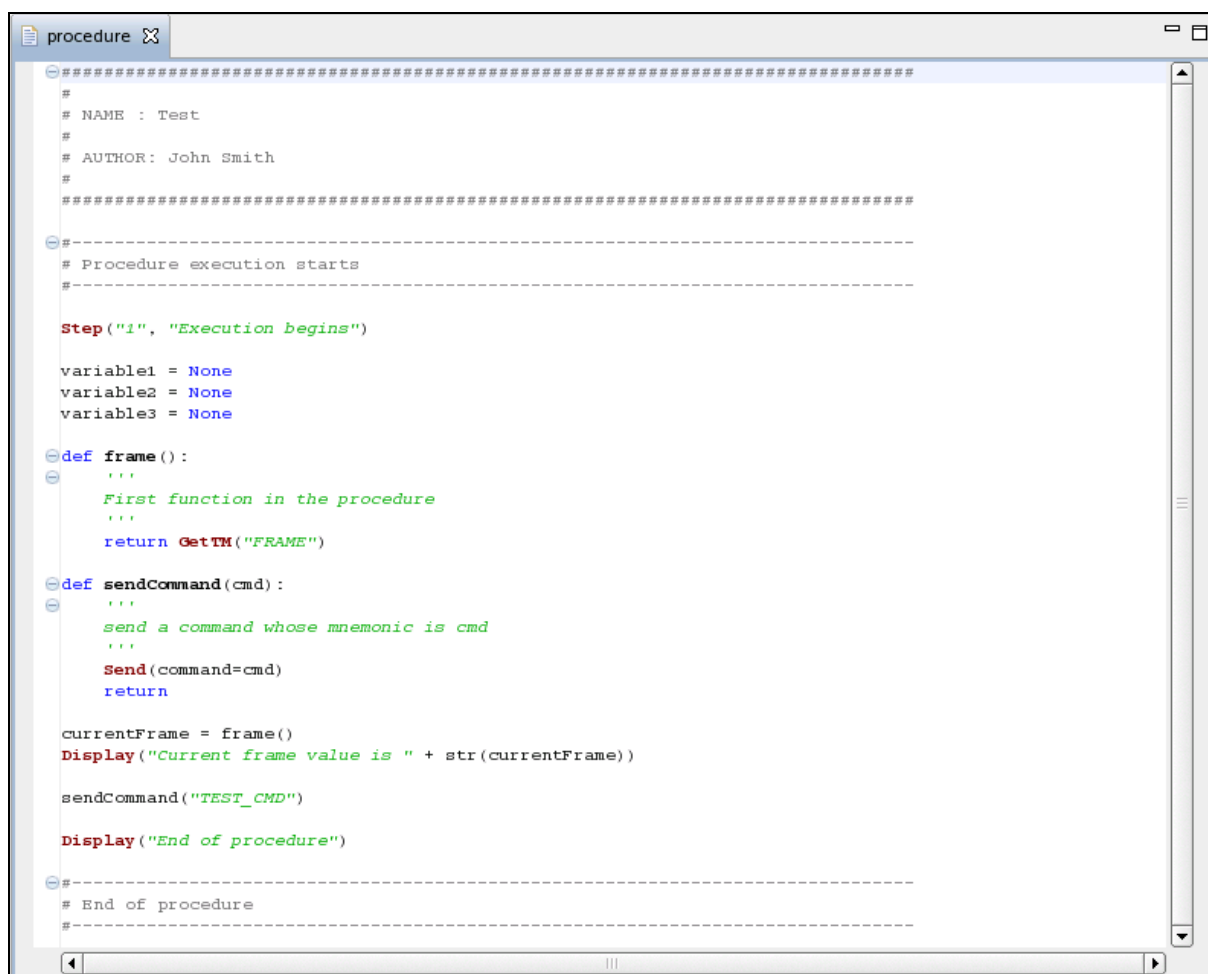


Figure 10. Outline view

2.2.2 Editors

2.2.2.1 SPELL 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.



```

=====
#
# NAME : Test
#
# AUTHOR: John Smith
#
=====
# -----
# Procedure execution starts
# -----
Step("1", "Execution begins")

variable1 = None
variable2 = None
variable3 = None

def frame():
    """
    First function in the procedure
    """
    return GetTM("FRAME")

def sendCommand(cmd):
    """
    send a command whose mnemonic is cmd
    """
    Send(command=cmd)
    return

currentFrame = frame()
Display("Current frame value is " + str(currentFrame))

sendCommand("TEST_CMD")

Display("End of procedure")

# -----
# End of procedure
# -----
  
```

Figure 11. SPELL procedure editor

2.2.2.2 SPELL database files editor

SPELL database file editor helps developers to edit SPELL database files. Like the procedure editor, it has syntax highlighting for special elements.

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.

```

*databaseFile.imp
#####
#
# Name: test database for SPELL-Dev user manual
#
#####
#
# Author: SPELL team
#
#####
#Number
KEY_NUMBER          123.32

#Character sequence
KEY_NAME            'SPELL'

#List
KEY_LIST            [1,2,3,4,5,6,7]

#Dictionary
KEY_DICTIONARY      {'a':123, 'b':'spell'}

```

Figure 12. Database editor

3 Working with SPELL DEV

3.1 SPELL projects

Projects are the main structural element inside SPELL DEV workspace. 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.

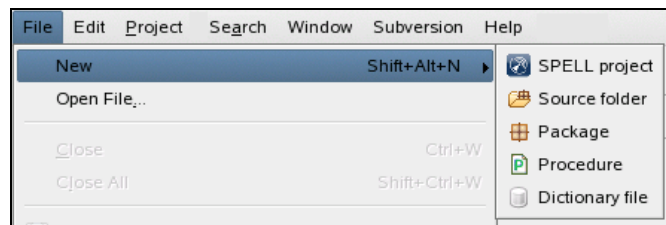


Figure 13. New element wizards

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.

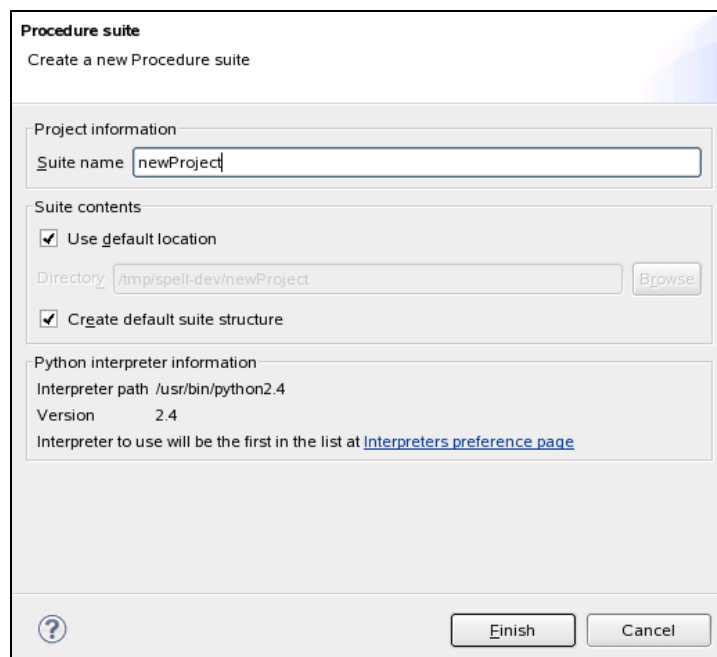


Figure 14. New SPELL project wizard

- When pressing the Finish button, the project will be created, and it will be shown in the [Procedure Explorer](#)

3.1.2 SPELL project elements

As explained in section [2.2.1.1](#), 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 inside a SPELL project:

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.

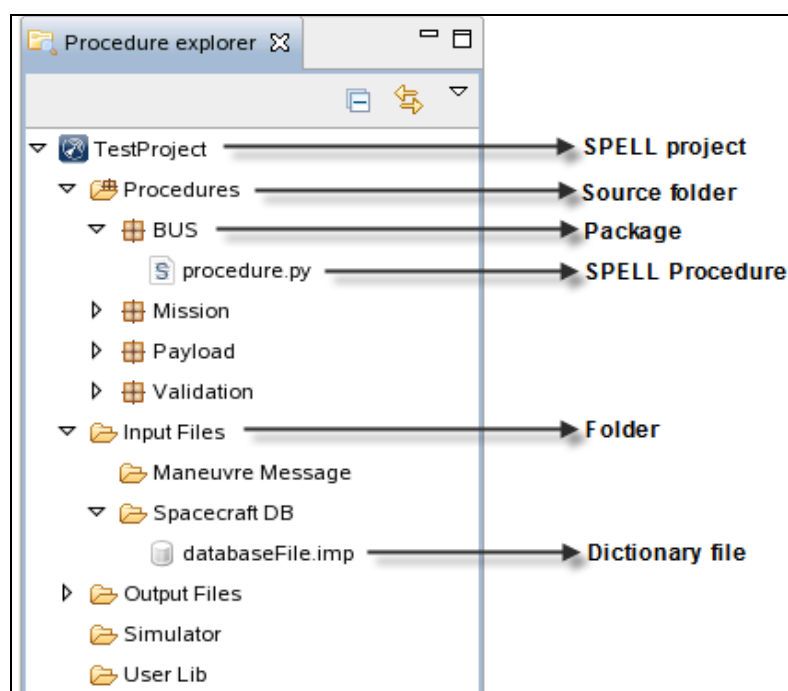


Figure 15. SPELL project elements

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.

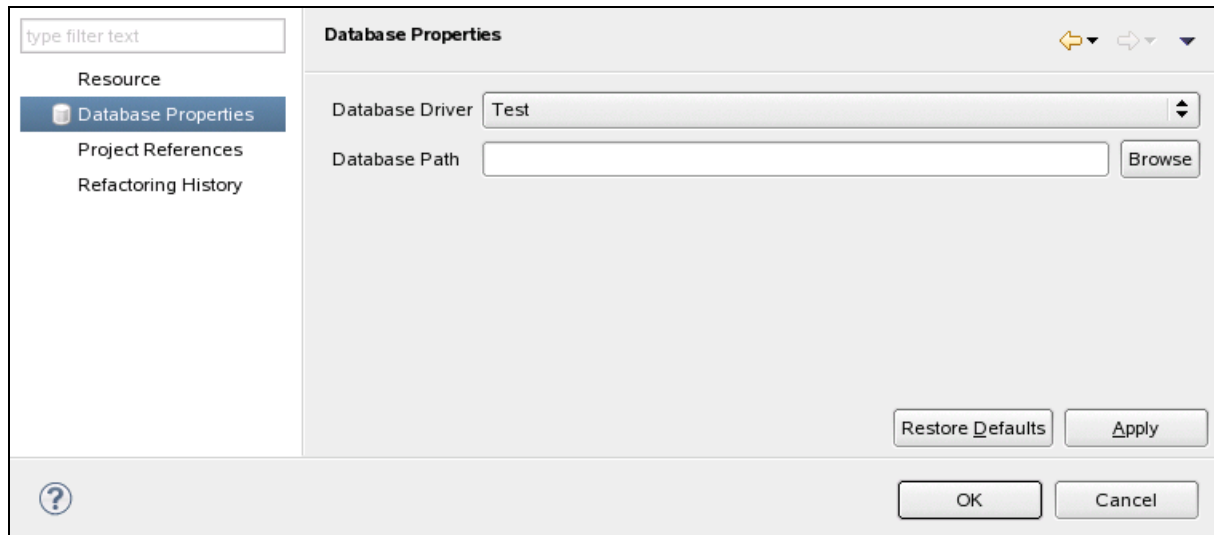


Figure 16. Database properties page

3.2 Procedures

3.2.1 Create a new procedure

To create a new procedure file, follow these steps:

1. In the menu area, select File, then New, and finally Procedure. A dialog will appear.

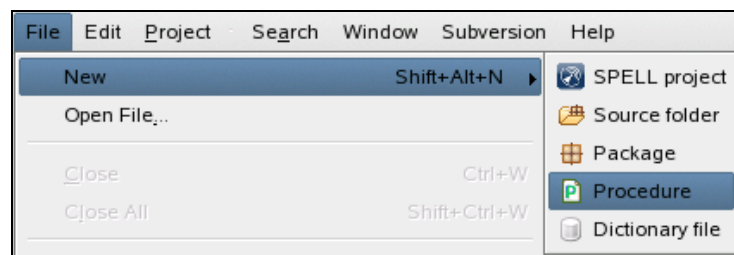
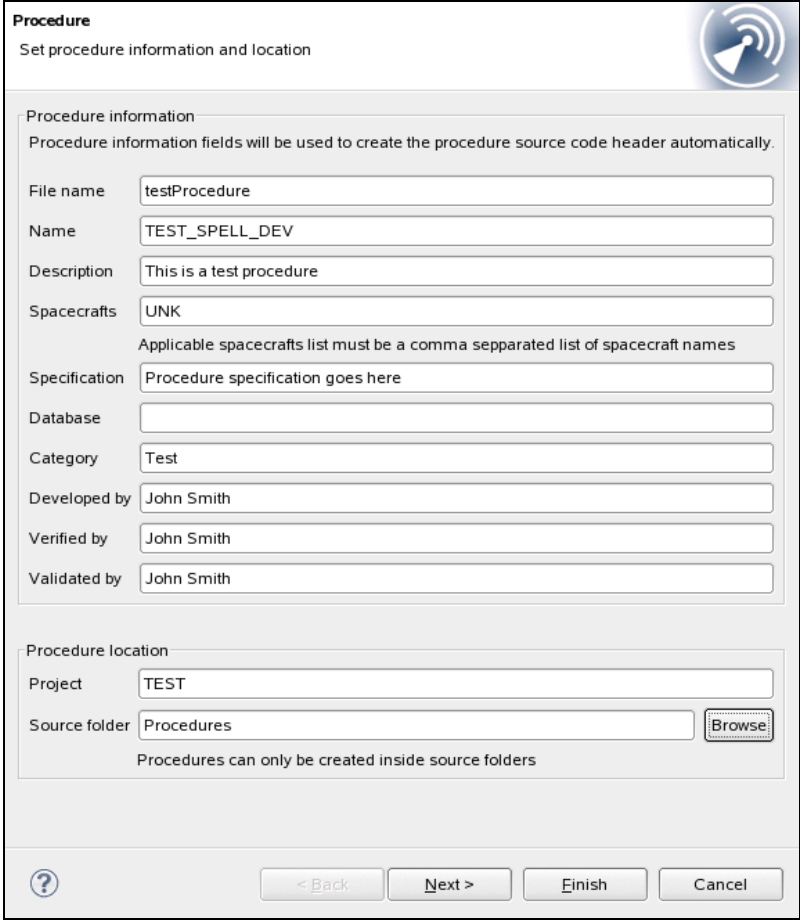


Figure 17. New procedure wizard access

2. 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 18. 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 a procedure

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.

3.2.3 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.

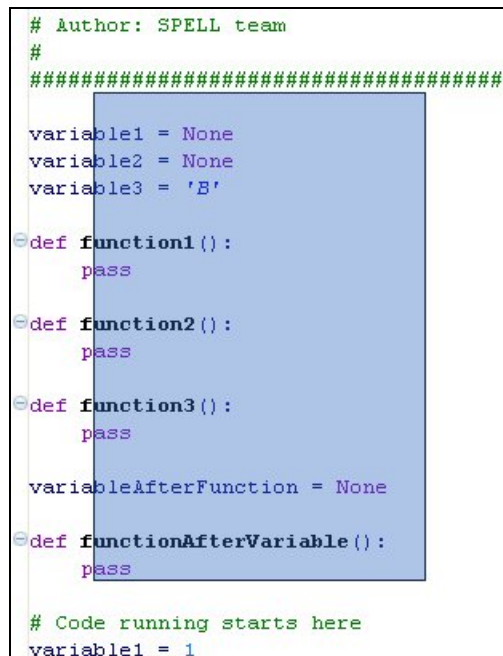
To enable or disable column mode, just toggle the button located in the menu area.



Figure 19. Column mode icon

3.2.3.1 Selecting a rectangular region

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.



```

# Author: SPELL team
#
#####

variable1 = None
variable2 = None
variable3 = 'B'

def function1():
    pass

def function2():
    pass

def function3():
    pass

variableAfterFunction = None

def functionAfterVariable():
    pass

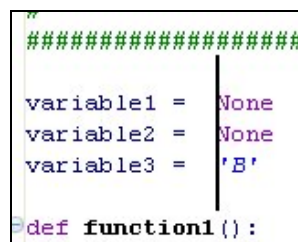
# Code running starts here
variable1 = 1

```

Figure 20. Rectangular selection

3.2.3.2 Indenting a region

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 21. Block indentation

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

3.2.4 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.

3.2.4.1 Database drag and 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.

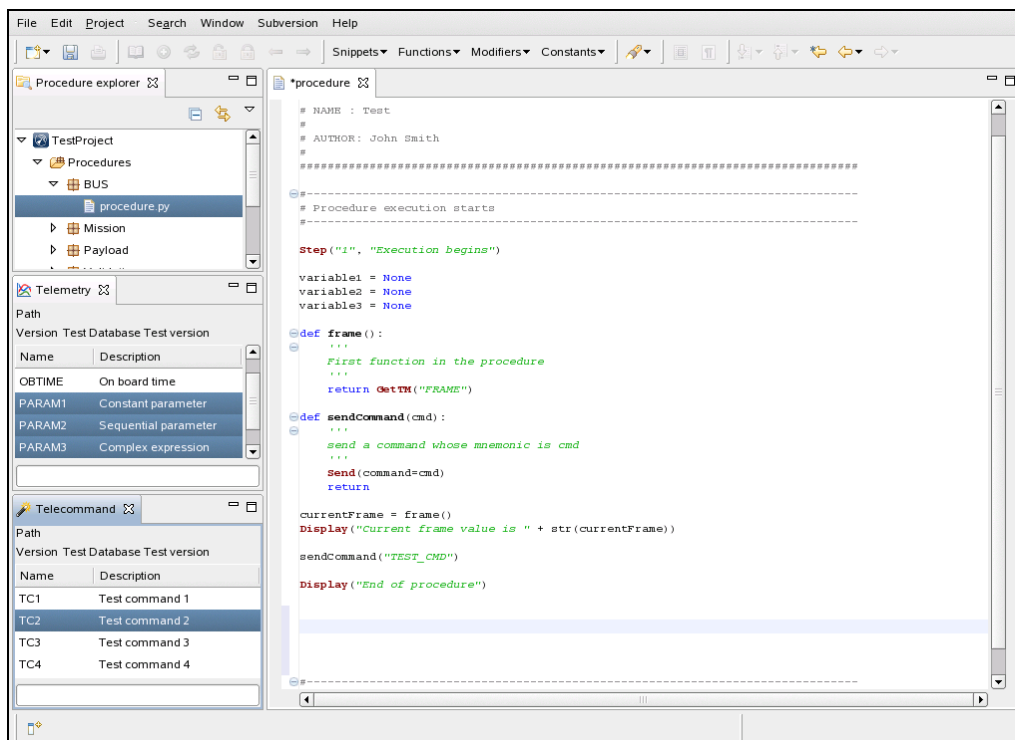


Figure 22. Select TM and TC elements from the views

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

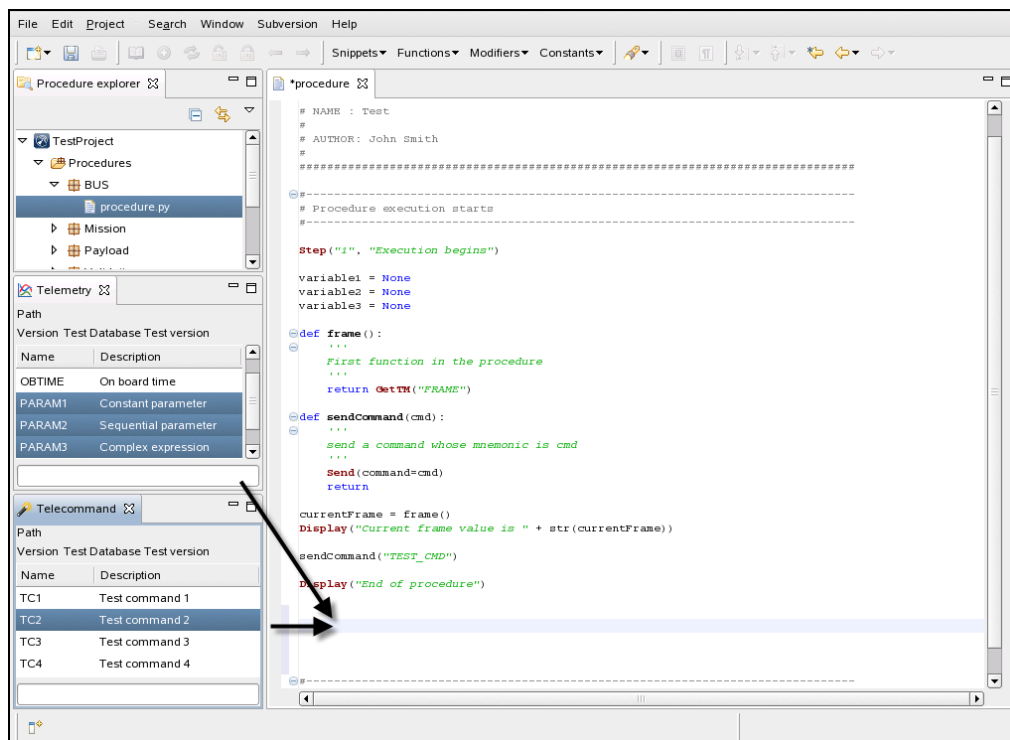


Figure 23. Drag selected elements

3. 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.

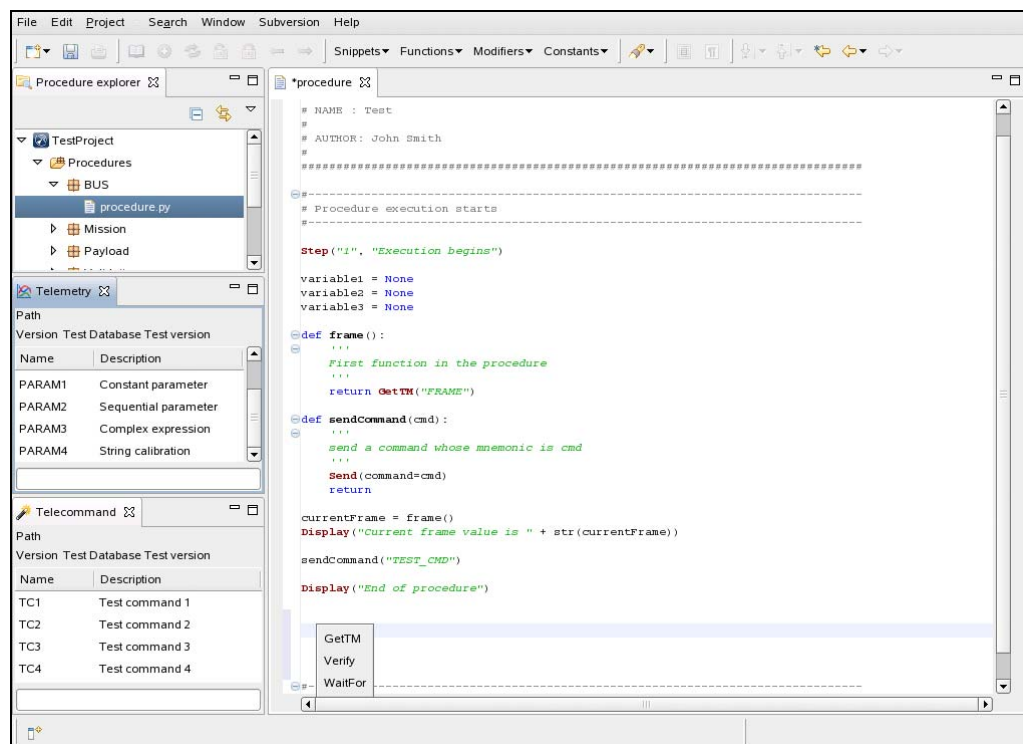


Figure 24. After dropping, code generation options are shown

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

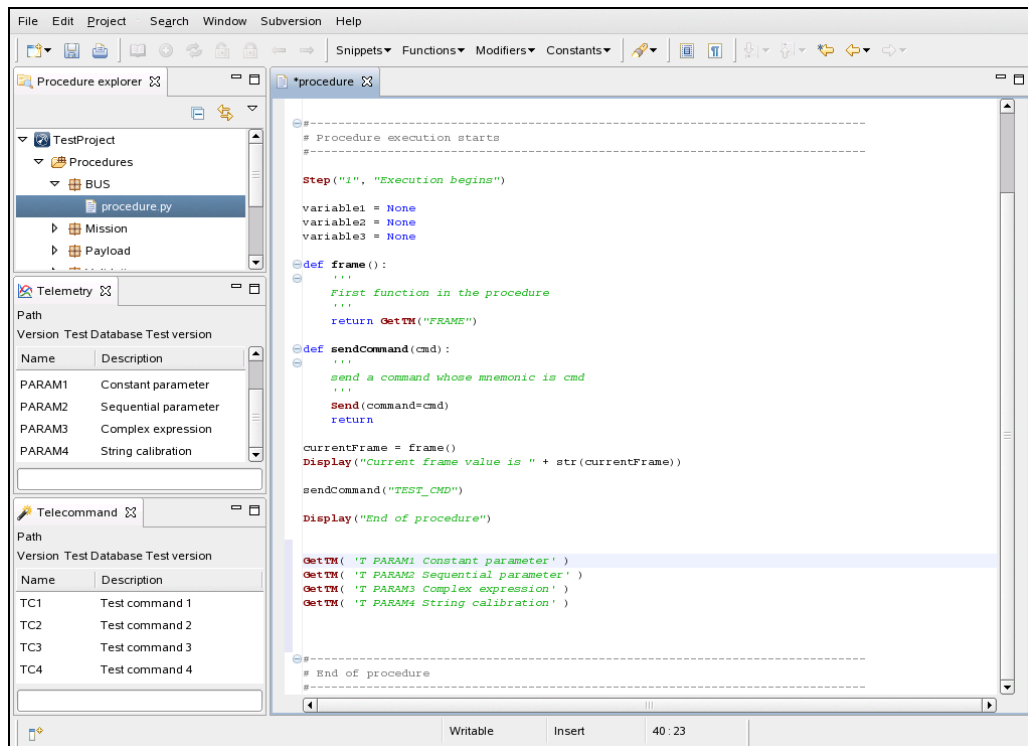


Figure 25. Code has been inserted

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

3.2.4.2 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 26. 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.

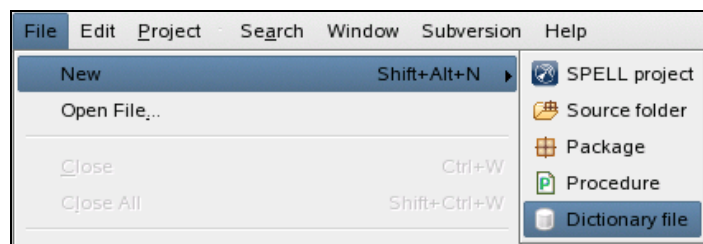


Figure 27. New dictionary wizard access

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.

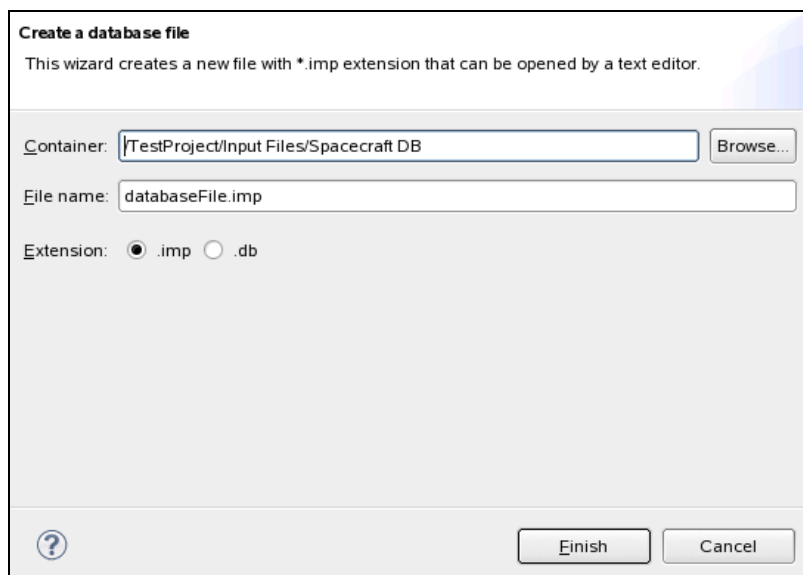


Figure 28. New dictionary file wizard

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 Exporting and Importing

3.4.1 Importing projects from 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:

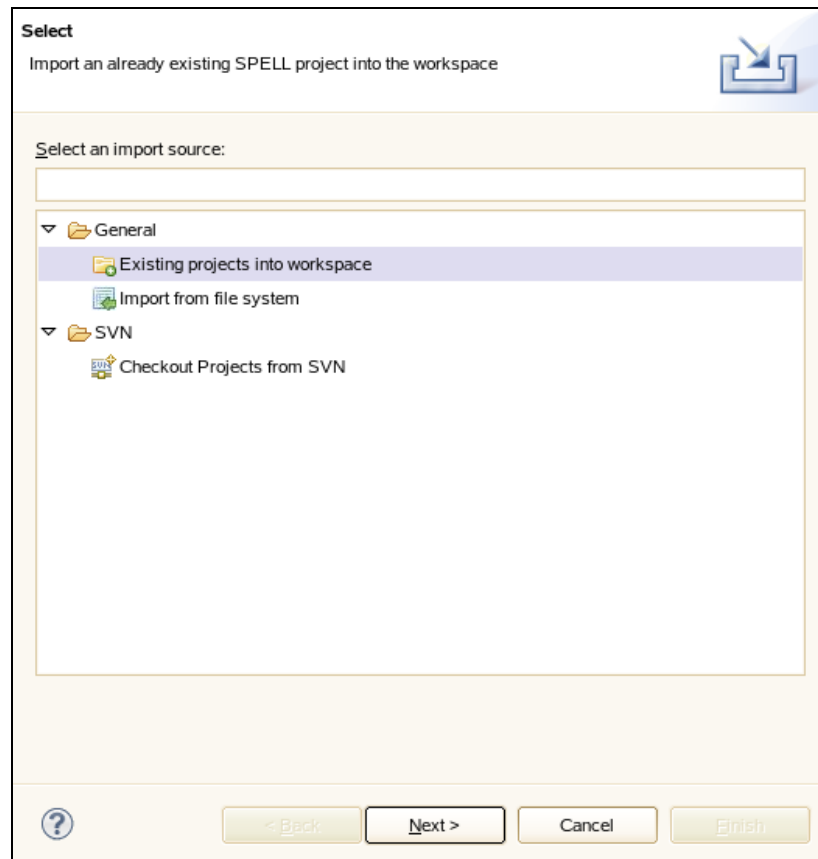


Figure 29: import dialog

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 30).

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.

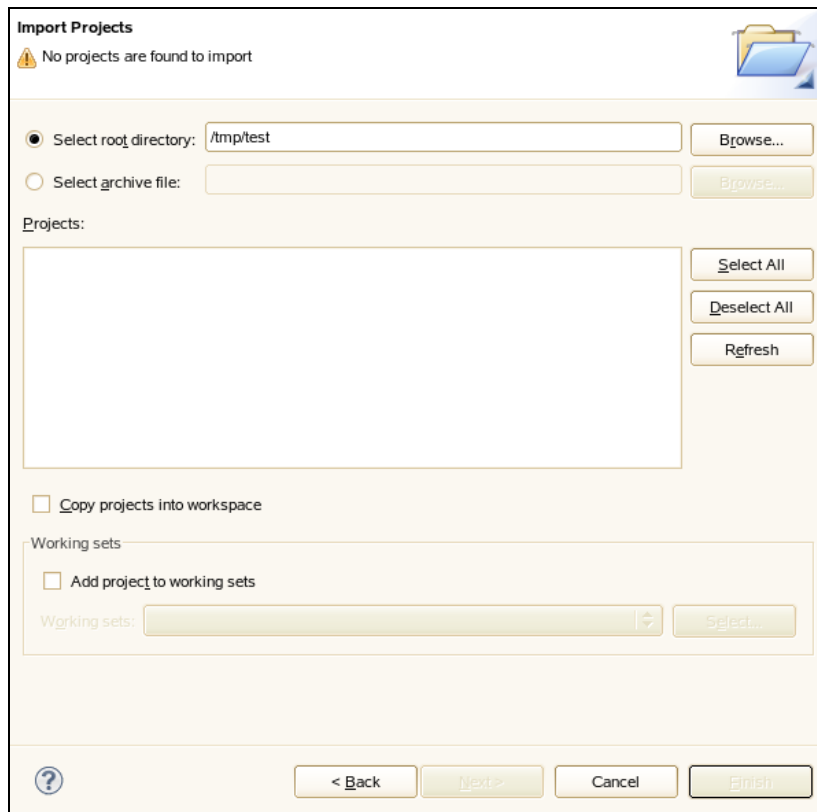


Figure 30: project selection list

3.4.2 Importing projects from version control

Please refer to section 5.3.1.

3.4.3 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 29).

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 31 for a snapshot of the wizard page.

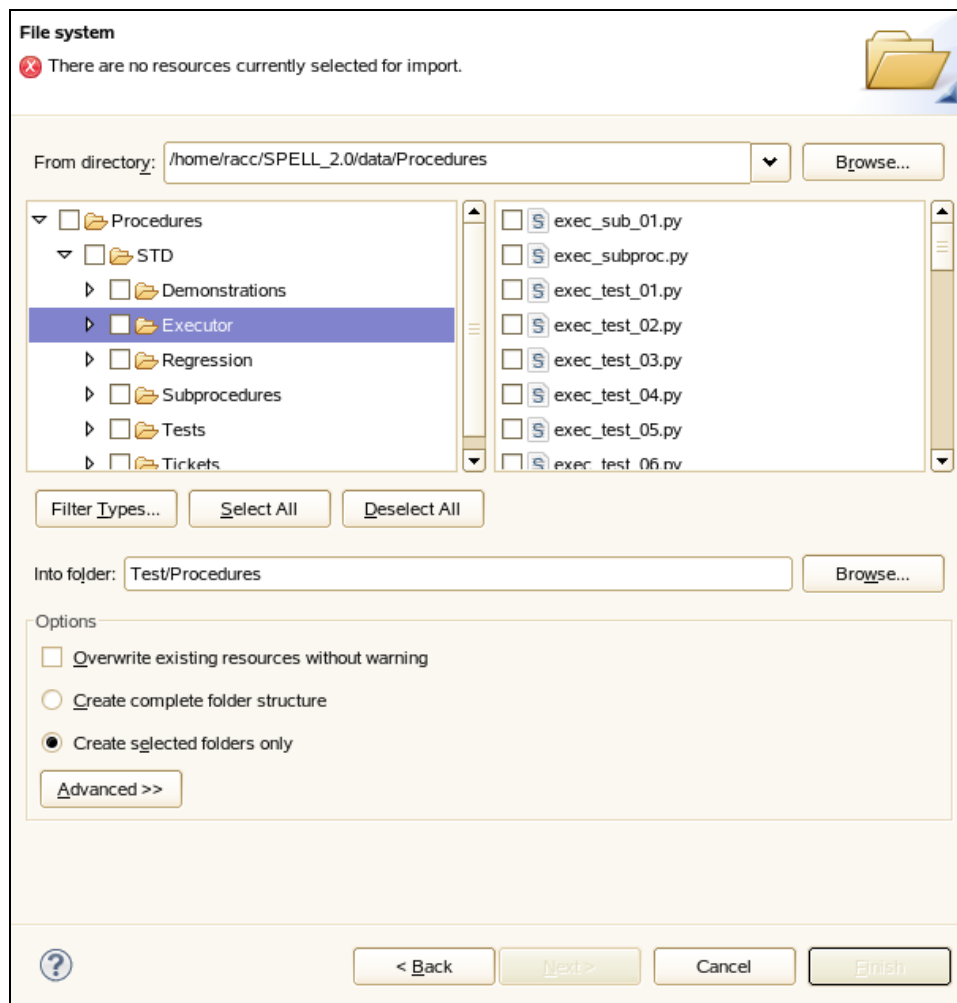


Figure 31: importing files

3.4.4 Exporting regular 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 Customizing SPELL-Dev

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.

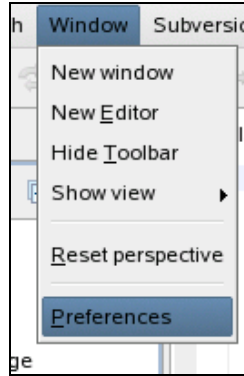


Figure 32. Open preferences dialog

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 in section [2.1.3](#), 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.

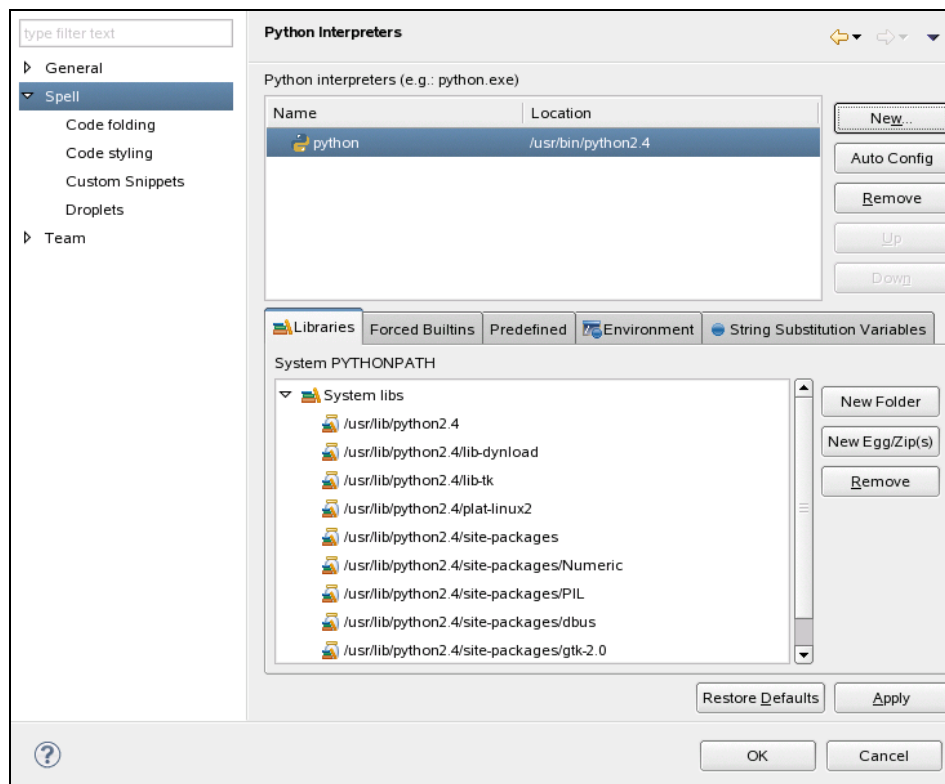


Figure 33. Python interpreter preference page

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.

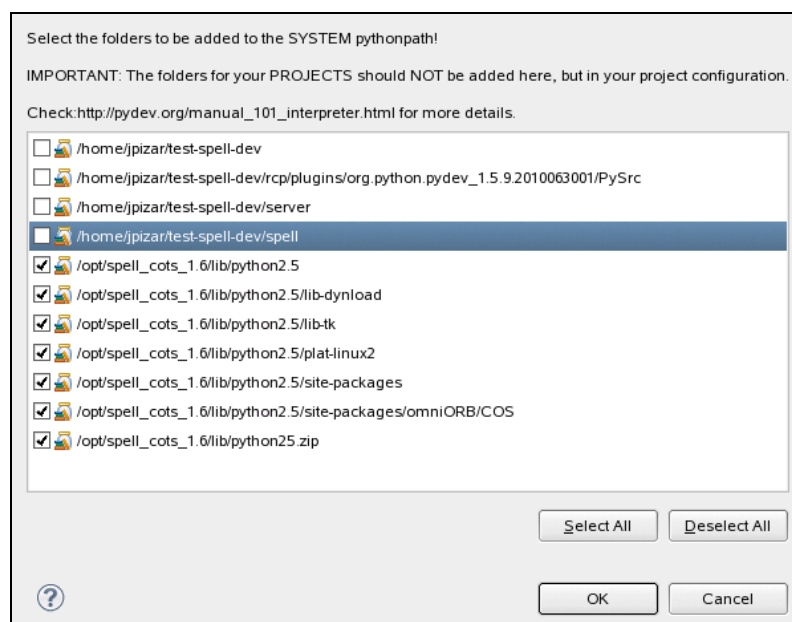


Figure 34. Python path interpreter settings

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.

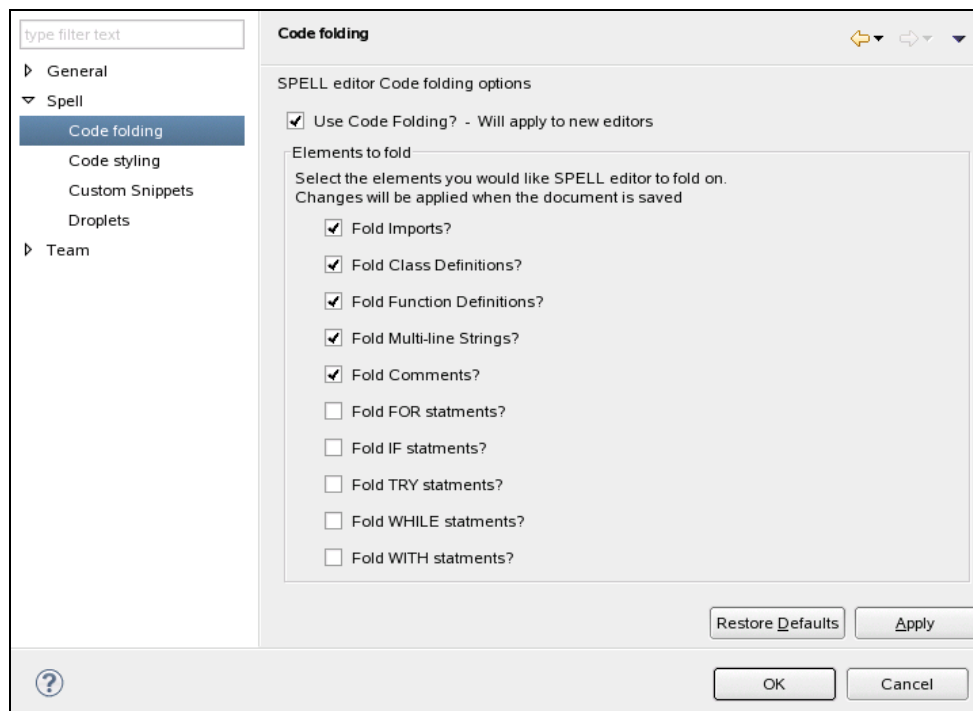


Figure 35. Code folding preferences page

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.

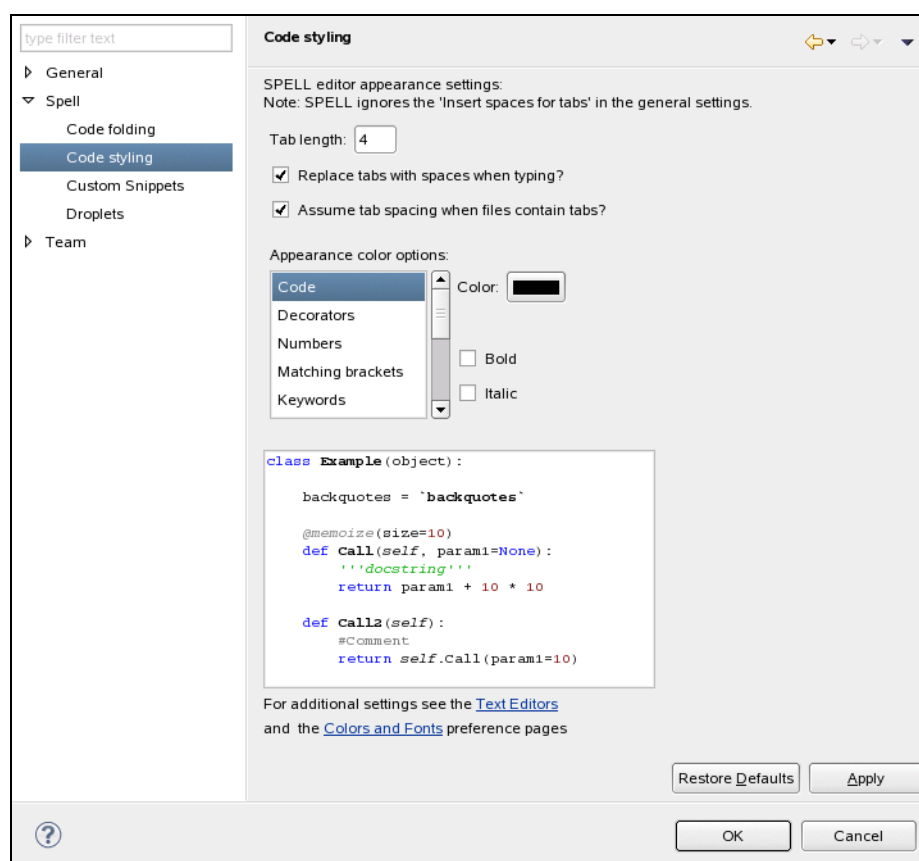


Figure 36. Code styling preferences page

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.

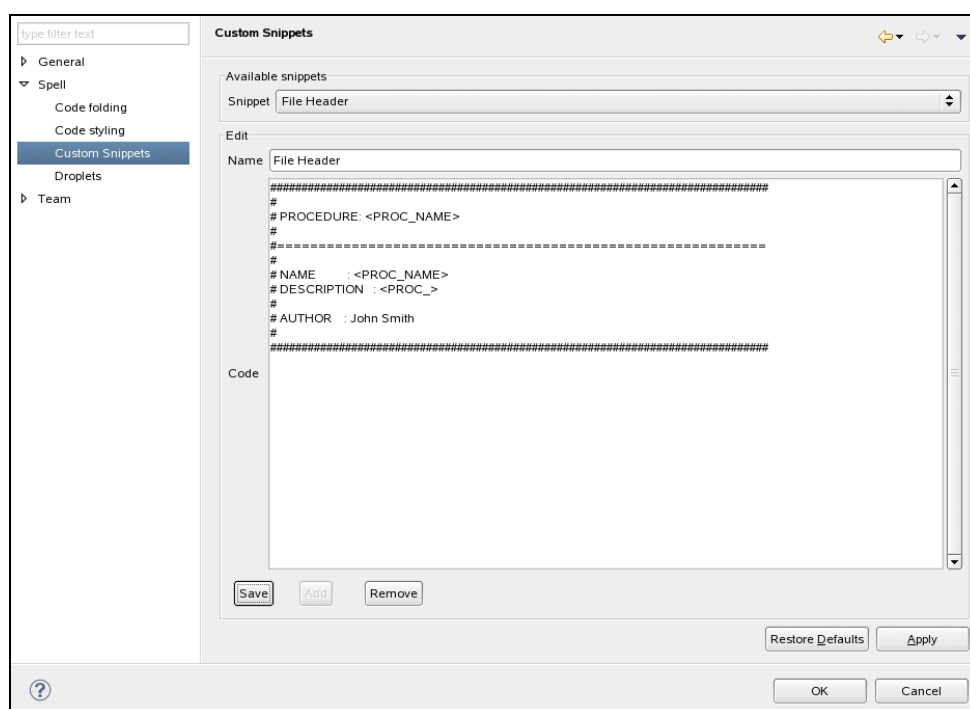


Figure 37. Custom snippets preferences page

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.

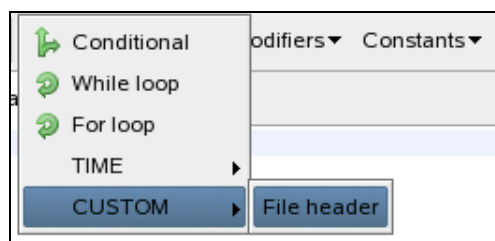


Figure 38. Custom snippets option

4.5 Droplets preferences page

As presented in section 3.2.4, SPELL 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.

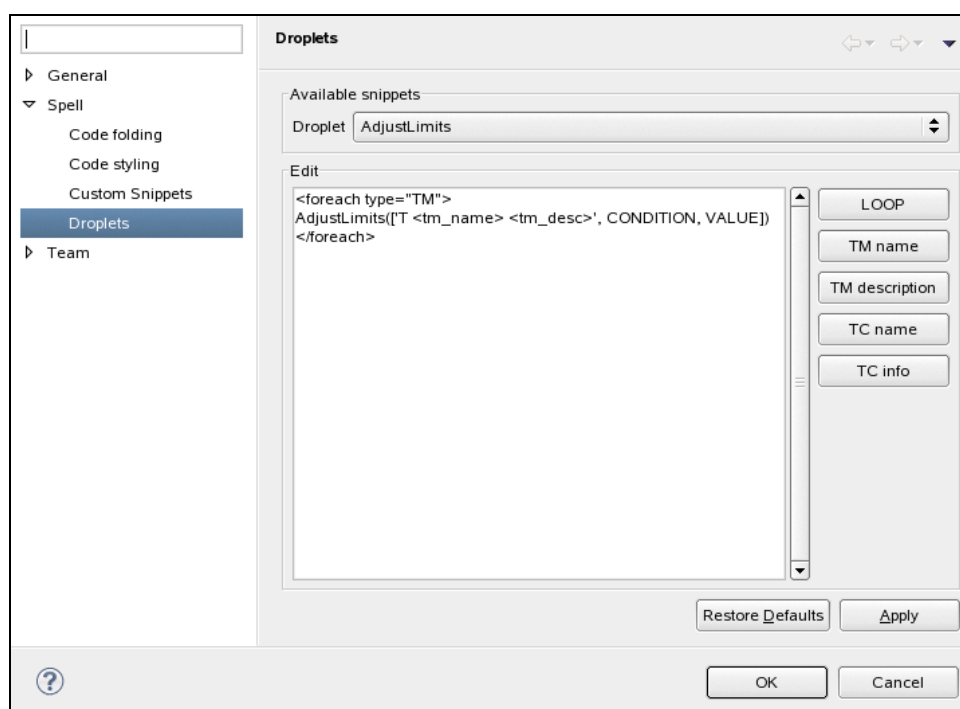


Figure 39. Droplets preferences page

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.

5 Version control system

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

5.2.1 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.

5.2.2 Update

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.

5.2.3 Commit

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.

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

5.2.4 Revert

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.

5.2.5 Lock

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.

5.2.6 Unlock

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.

5.2.7 Add to control version

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.

5.2.8 Show history

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.

5.2.9 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.

5.3 CVS inside SPELL DEV

SPELL DEV includes a version control system tool, called Subversion, allowing procedure developers to work concurrently over the same project. Actions explained in section 4.2 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 40. 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.

5.3.1 SPELL 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.

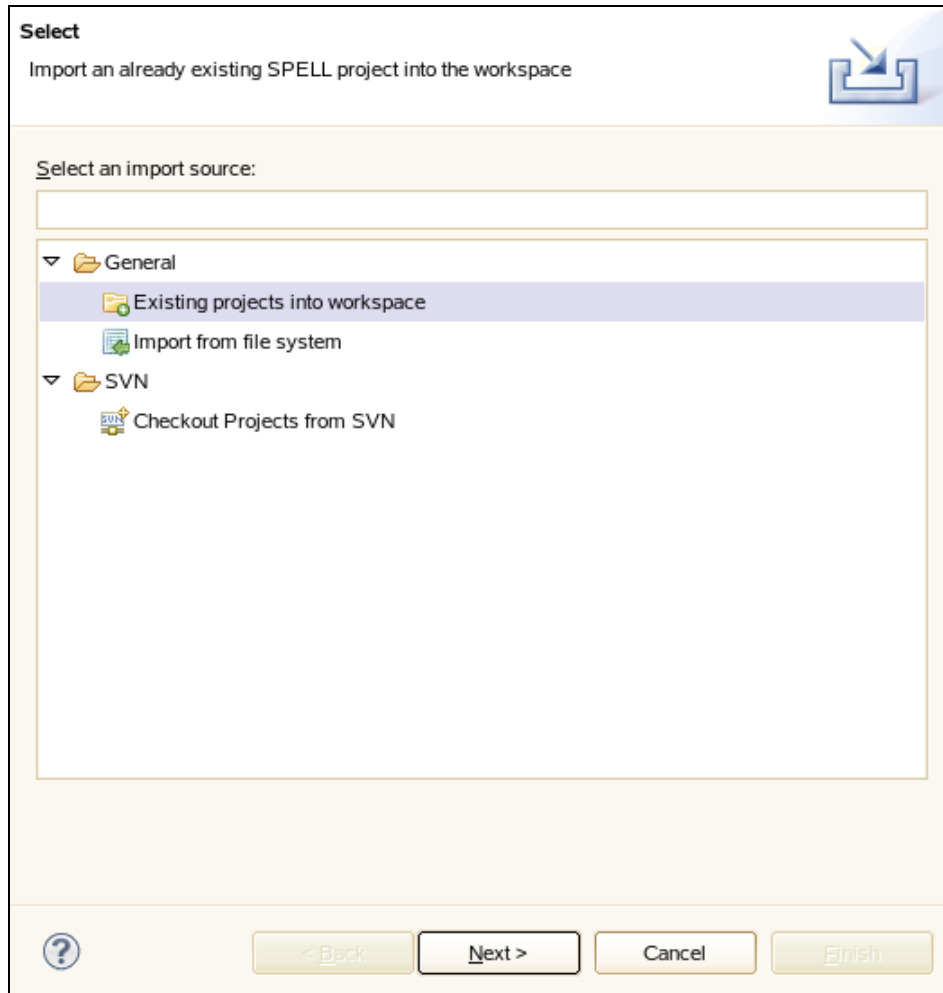


Figure 41. Checkout project option

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.

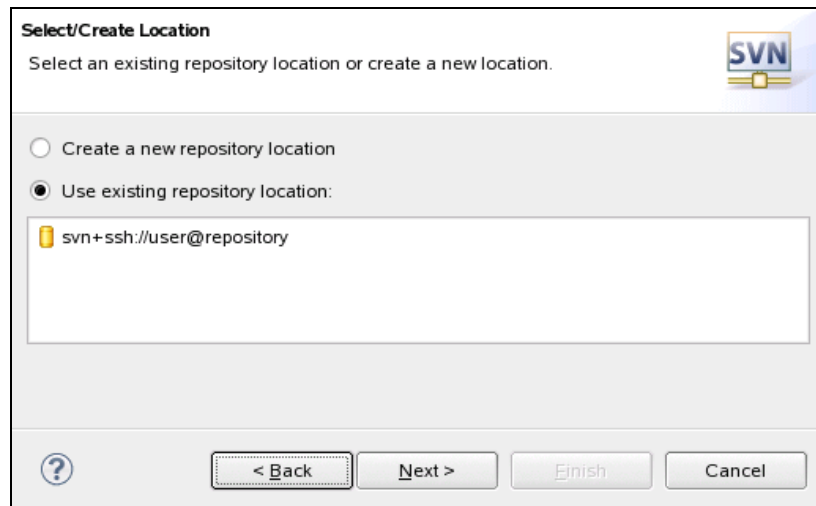


Figure 42. Checkout project from existing repository

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.

5.3.2 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.

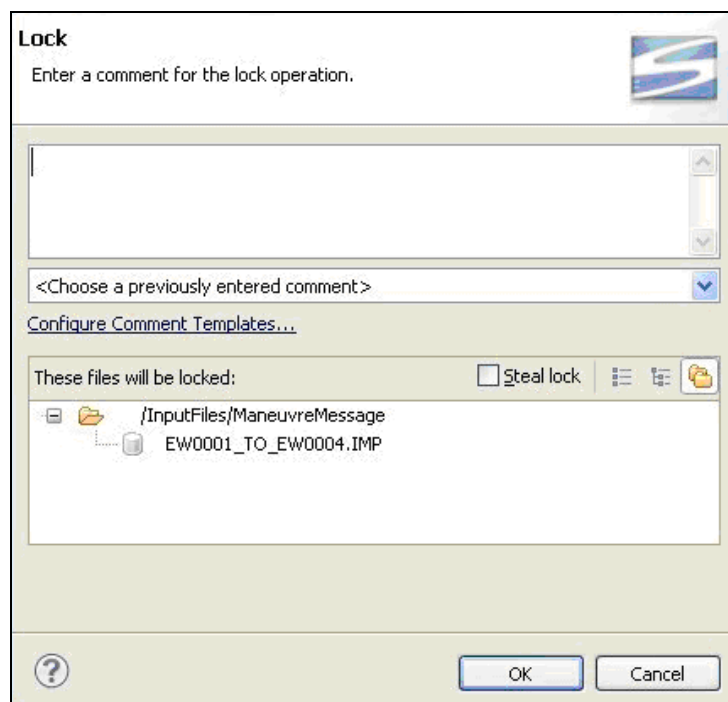


Figure 43. SPELL DEV lock dialog

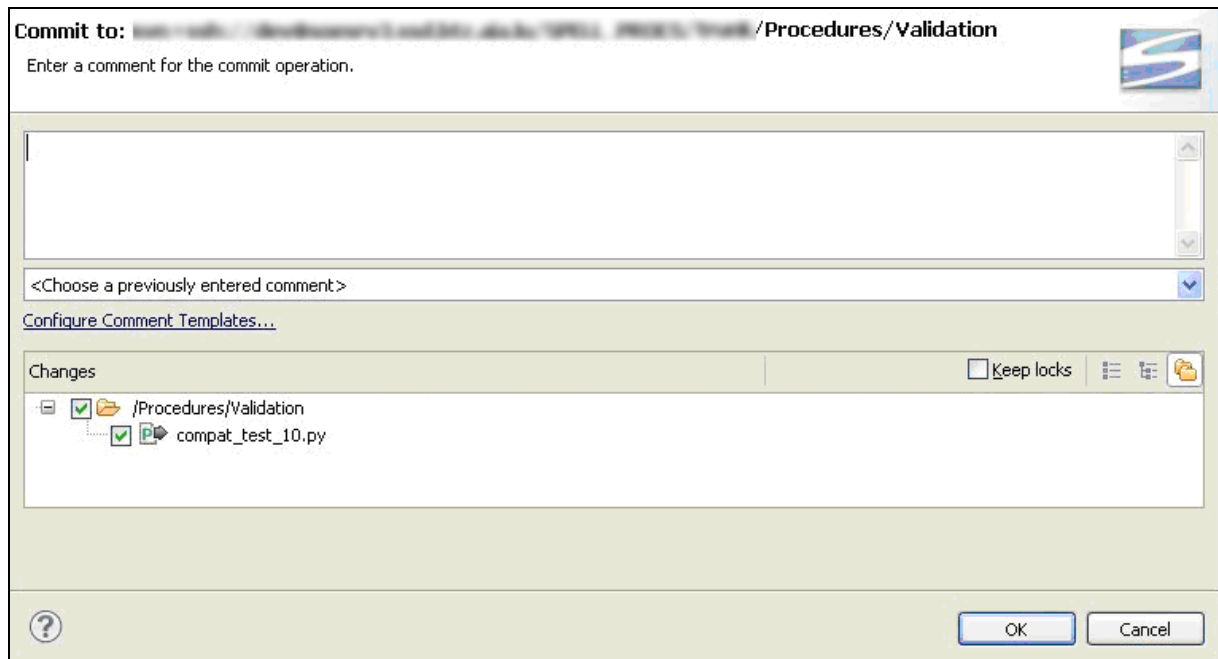


Figure 44. Commit dialog

5.3.3 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 the select the Mark Resolved option. When a conflict is marked as resolved, .mine and .rXXXX files are removed automatically.

5.3.4 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 wan to compare.
2. When the context menu appears, put the mouse over the Compare with option, and the 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.

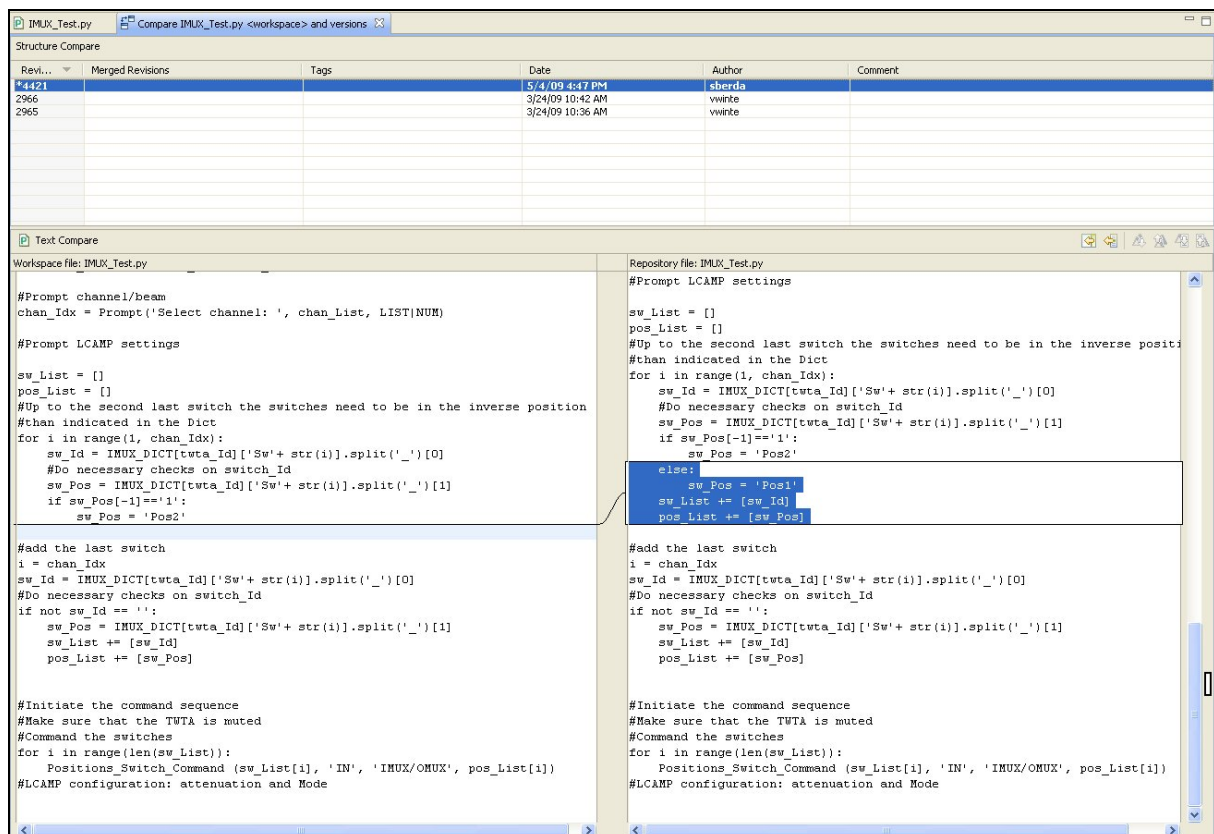


Figure 45. Compare with editor

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.

5.3.5 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.

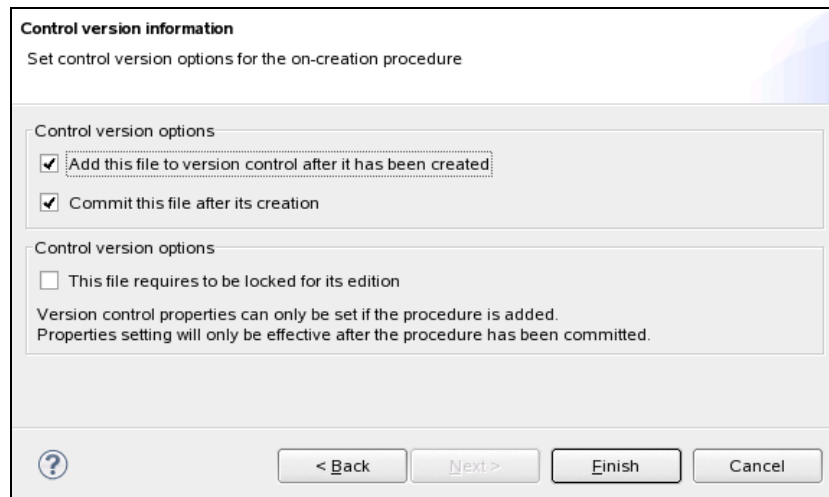


Figure 46. Control version page in the New Procedure wizard

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.

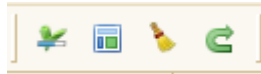


Figure 47: semantics check toolbar

The toolbar (see image above) contains four buttons:

- Trigger semantic check
- Generate semantic check report files
- Clean semantic check results
- Re-parse user library

Same options appear in the Semantics menu:

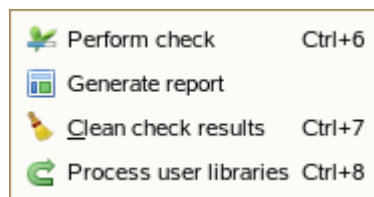


Figure 48: semantics check menu

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.

When a project is selected, all procedures within the project are checked. The same applies when a folder is selected. It should be taken into account that performing a semantic check on a whole project can take quite long to finish. It is advisable then to trigger the check on small sets of procedures (or single procedures) at a time.

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 SPELL procedure editor with some semantic markers on it:

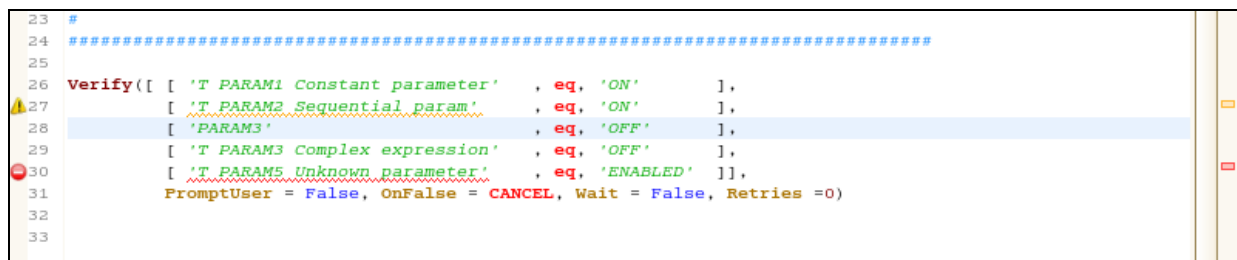


Figure 49: editor with semantics markers

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:

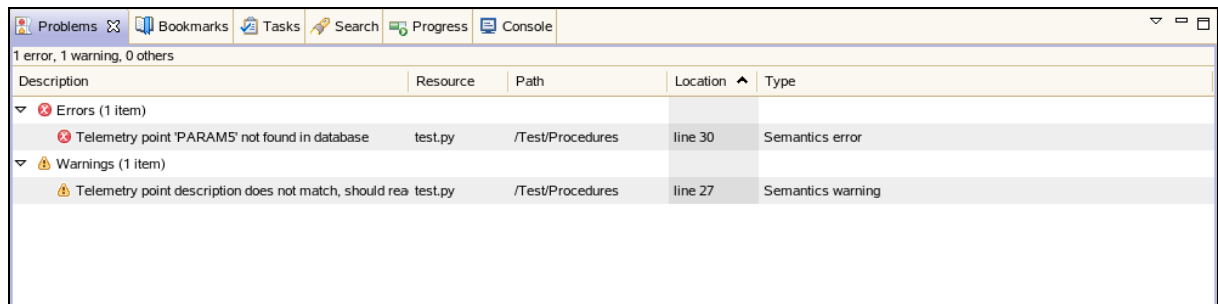



Figure 50: Problems View showing the semantics problems

3 July 2011	SPELL Development Environment Manual USL/SPELL Software version 2.0.5 UGCS-USL-SPELL-DEV-SUM_10_003_1.3.doc	
Page 44 of 44		

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.