Sif
Mjølner BETA Source Browser and Editor
Tutorial and Reference Manual

Mjølner Informatics Report
MIA 90-11(2.1)
August 1996

Contents

1. Introduction

Source browser structure editor and tool integration

Sif[1] is a general grammar-based editor, but it is especially useful for browsing and editing BETA programs. Modularity is in BETA handled by means of the fragment system [MIA90-2] used for combining fragments into a whole BETA program. Familarity with the fragment system is expected in this manual.

Sif contains a number of basic components: a source browser, a fragment group editor and a fragment form editor. Sif is integrated with the BETA compiler which gives a good support for locating and correcting semantic errors.

Sif functionality

Sif includes the following functionality:

- **Source Browser**

  The source browser makes it possible to browse in projects. A project can be a collection of files, a file directory or parts of the dependency graph of a BETA program. Links in the fragment structure like ORIGIN and INCLUDE can be followed easily.

- **Fragment Group Editor**

  The fragment group editor is a high-level editor used to manipulate the dependency graph of BETA programs, e.g. to create and delete whole fragment forms in fragment groups and to create and modify links like ORIGIN and INCLUDE between the fragment groups.

- **Fragment Form Editor**

  The fragment form editor is a structure editor that works inside fragment forms. Structure editing is a powerful technique for editing programs without introducing syntax errors. At the fragment form level Sif also provides useful browsing facilities, based on the syntactic and semantic structure of BETA programs.

- **Structure editing**

  The basic idea of structure editing is that the program is manipulated in terms of its logical structure rather than the textual elements such as characters, words and lines. The advantage of this approach is that only logically coherent parts can be inserted or deleted and thereby preserving the syntactical rules of the language at any time.

  Structure editing is especially useful for application-oriented languages intended for end-users, casual users and beginners that may have difficulties in remembering the concrete syntax. Also a program constructed by structure editing needs not be parsed, thereby saving time in the development phase.

- **Text editing**

  Structure editing has its greatest force at the higher levels of editing, i.e. for creating the overall structure of the program or for moving around large chunks of code. At the detailed level the textediting technique is more useful. Therefore the programmer may alternate freely between structure editing and textual editing in Sif. Any program part may be textually edited.

---

1 Sif is the wife of the god Thor. Sif is well-known for her golden hair.

- **Incremental parsing**

  Any program part that has been textually edited is immediately parsed. Only the text-edited part of the program is parsed.

- **Adaptive incremental prettyprinting**

  The editor includes an adaptive prettyprinting algorithm which prints the program such that it always fits within the size of the window or paper.

- **Abstract presentation and browsing**

  The editor is able to present a program at any level of detail. At the top-level of a program the user may get an overview of classes and procedures. It is then possible to browse through the classes and procedures to see more and more details. This mechanism is completely general since the user may decide the level of granularity. Printing a program at different abstraction levels provides a good basis for documentation.

- **Integration of documentation and comments**

  The user decides whether or not to display comments. The user also decides whether to display a comment as part of the program or in separate window. A pretty-print of a program which includes just the class and procedure headings (an abstract presentation) and corresponding comments may be produced. This makes it possible to extract an interface specification from the program including the explaining comments.

  Programmers are motivated for integrating code and documentation since comments are easy to hide. Large pieces of documentation need not to disturb the overview of the program. Conversely it is easy to extract a high-level presentation of the program including the comments.

- **Hypertext facilities**

  The editor includes hypertext facilities. The facility for handling comments is an example of a hypertext link between a program and a text piece. Another type of hypertext link is links from the use of a name in a program to the corresponding name declaration. Such semantic links are very useful especially when working with large programs. At the fragment group level, other examples like ORIGIN and INCLUDE links are links from SLOTs to their corresponding fragment forms and vice versa.

- **Metaprogramming system**

  The editor is built upon a metaprogramming system [MIA91-14], which is available to the user. The user has the possibility to program his or her own metaprogramming tools. It is possible to integrate such tools with the editor or simply to add functionality to the editor. This tailorability is provided by a flexible communication model and the object-oriented implementation language of the editor, which is BETA.

- **Grammar-basis**

  The editor is grammar-based, which means that it may support any language that can be described by means of a context free grammar. All the facilities mentioned above (except semantic links which require a semantic checker) are provided for any language that can be described in a context free grammar. Structure editing can not only be used on programs but any kind of documents with a formal or semi-formal structure. In the following the main focus will be on program editing in BETA. Editors have been generated for programming languages like BETA, SIMULA67, Pascal, Modula-2, for the query language SQL, for the specification languages SDL-92 and GDMO and finally for document types described in subsets of ODA and SGML. In addition the structure editor can be used to create or modify the grammars and prettyprint specifications for each language.

- **Integration with other tools in the Mjølner BETA System**
  Sif is integrated with the compiler, i.e. it is possible to perform semantic checking, code generation, assembling (if binary code is not generated directly) and linking inside Sif.

The purpose of this document is to describe how to use the structure editor Sif. The description is addressing the user who wants to use the structure editor for developing programs in one of the already supported programming languages, especially the BETA language.

More information

In [[MIA91-14] it is described how to generate a structure editor for a new language, and briefly how to add functionality to the editor or integrate the editor with another tool.

2. Basic User Interface Principles

2.1. The Mouse

The *Selection Button* is used to select arguments, to click on command buttons, to double-click and to select commands in all menus but one: the pop-up menu.

The *Pop-up Menu Button* is used to activate the pop-up menu, that is used during structure editing. The menu pops up when the mouse pointer is located in the codeviewer/editor window and the *Pop-up Menu Button* is pressed.

UNIX

On UNIX systems the left mouse button is used as the *Selection Button* and the right most mouse button is used as the *Pop-up Menu Button*. The middle mouse button has currently no use.

Windows NT/95

On Windows NT/95 systems the left mouse button is used as the *Selection Button* and the right most mouse button is used as the *Pop-up Menu Button.* The middle mouse button has currently no use.

Macintosh

On the Macintosh the mouse button is used as the *Selection Button* and Command-Click is used as the *Pop-up Menu Button.*

2.2. Selection

Selection is important because most operations in Sif are performed in relation to the current selection, which will always be marked in reverse video. There are different ways of selecting parts of a fragment.

When you position the cursor somewhere in the *Code editor* and click the *Selection Button* the nearest surrounding structure will be selected and marked as the current selection.

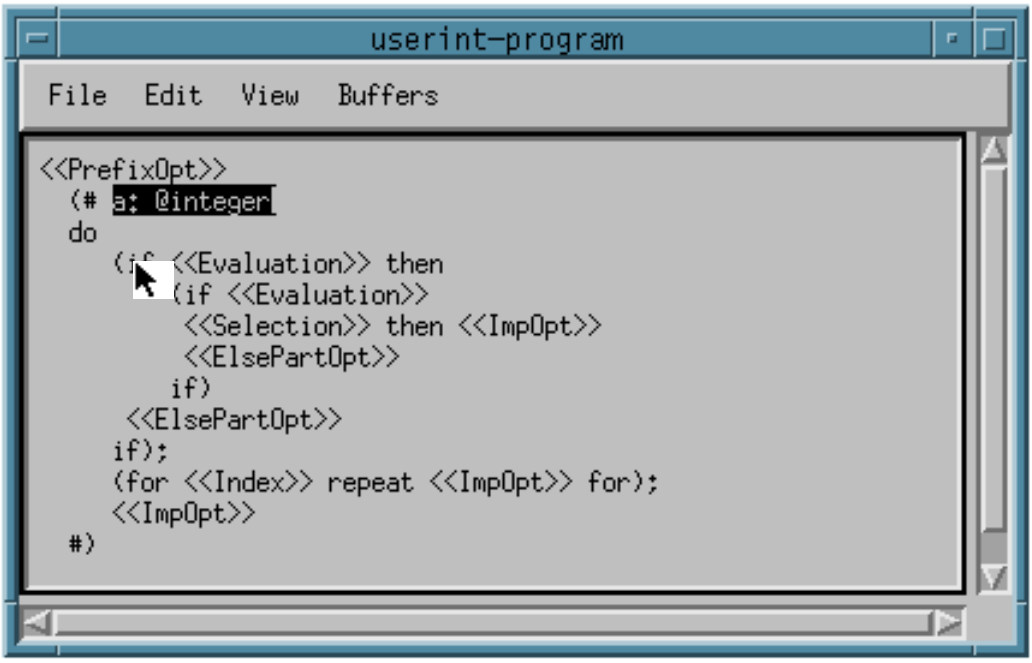
Some examples of how to select:

To select a name:

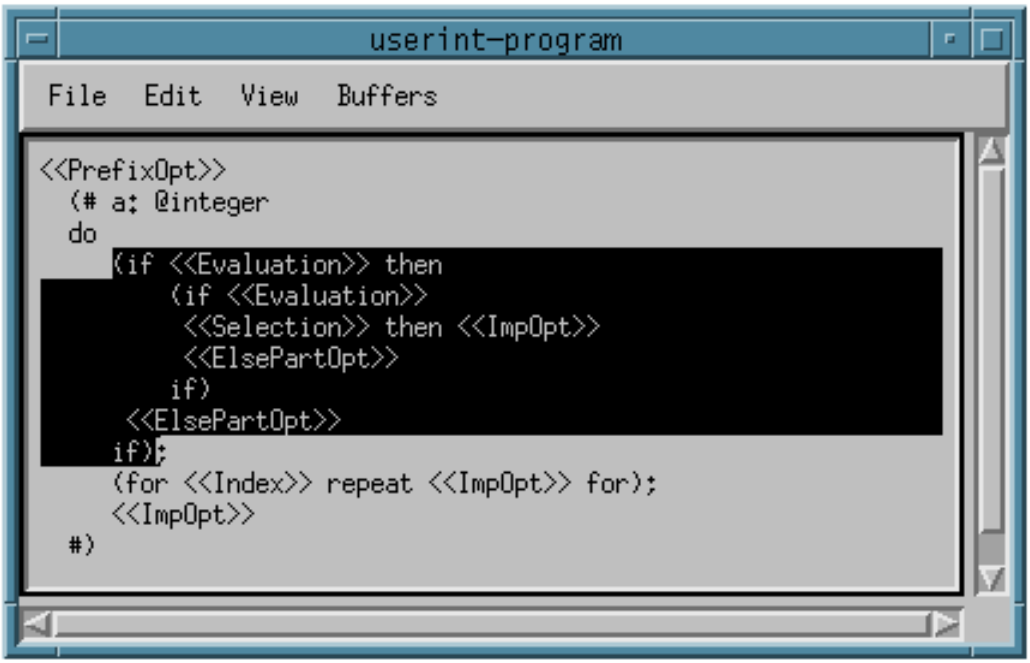
just place the cursor in the name and click the *Selection Button.*

To select a complete structure containing keywords :

place the cursor at one of the keywords and click the *Selection Button*:



Then the resulting current selection becomes:



To select all elements in a list:

place the cursor at one of the list separators e.g. a ';', and click the *Selection Button*.

To select a sublist:

If a sublist of elements is going to be the current selection, dragging selection is a convenient technique. Dragging selection means clicking with the *Selection Button* at the starting point of the sublist and keeping the button down while moving the mouse to the ending point of the sublist. At this point the button is released and the current selection will be set to the selected sublist.

In general the smallest complete enclosing structure will be selected.

3. Source Browser Tutorial

3.1. How to Get Started

The source browser that is part of Sif is activated by writing

```
sif
```

at the command line (Unix or Windows NT/95) or by double-clicking on the Sif icon (Macintosh).

Activating Sif the following window appears:

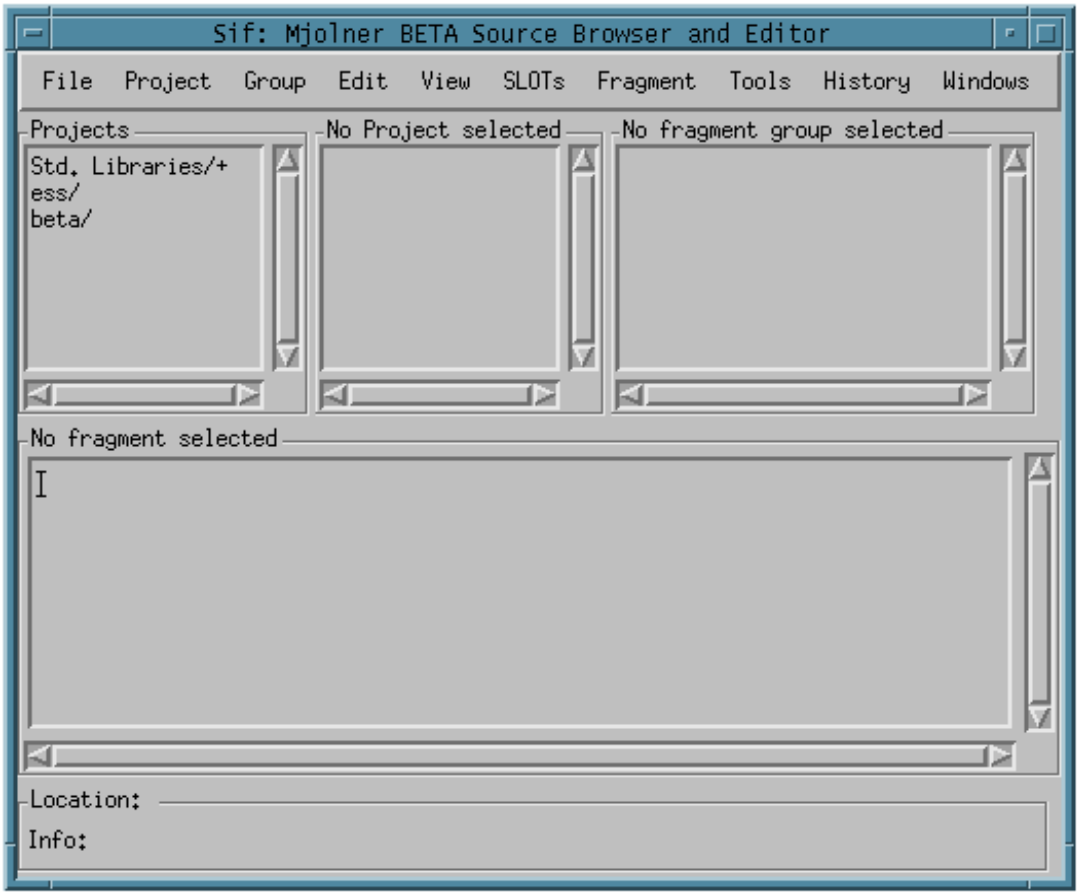


Fig. 1

This window is called a *source browser window* or just *browser window* and has 4 important panes. Followed clock-wise from the upper left pane, the panes are described below.

Pane 1 is called the *project list pane*, it contains a list of projects.

Pane 2 contains a list of fragment groups in the project selected in the project list pane. It is called the *fragment group list pane* or just *group list pane.*

Pane 3 contains a list of properties and fragment forms of the fragment group selected in the group list pane. It is called a *fragment group viewer/editor* or just *group viewer/editor*.

Pane 4 displays the BETA code of the fragment form selected in the group viewer. It is called the *code viewer/editor*[2]. This code viewer will be seen in many different windows and its functionalities are described later.

Browsing can be done at basically 3 levels: the project level, the group level or at the code level.

3.2. Browsing at Project Level

Double-clicking on the `Std. Libraries` project in the project list pane will give the following result:

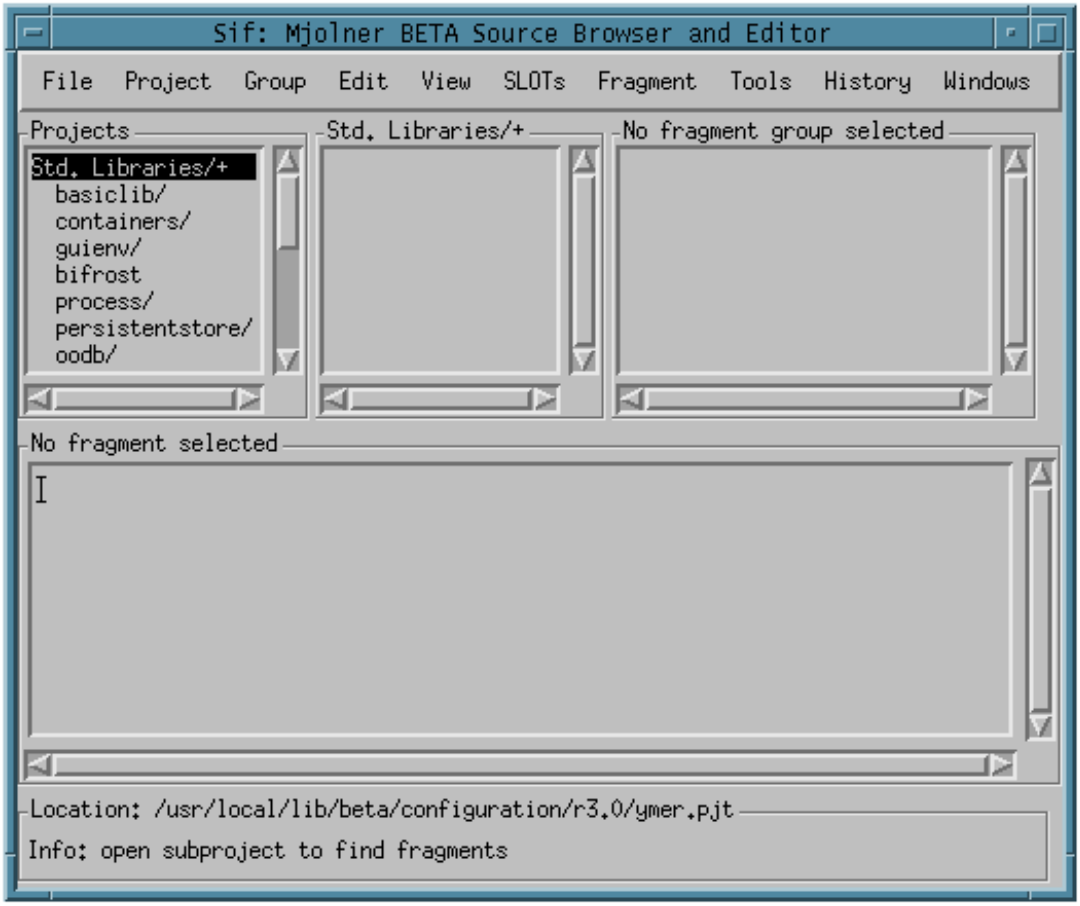


Fig. 2

`Std. Libraries` is the standard project that contains the basic libraries of the Mjølner BETA System. The contents of the `Std. Libraries` project is a list of file

---

2 The contents of the group viewer/editor can of course also be considered as part of the BETA code, but this is actually "code" written in the fragment language.

directories. By selecting the container directory its files (fragment groups) will be shown in the group list pane:

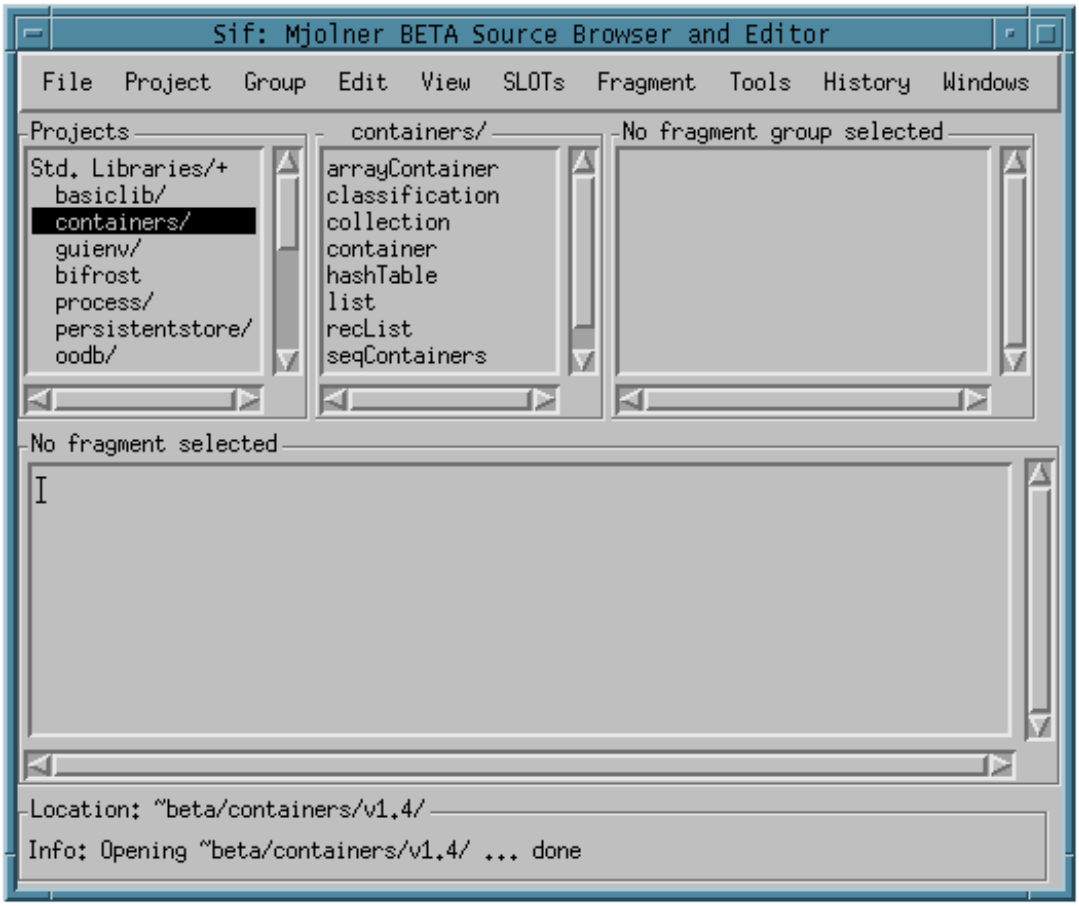


Fig. 3

Now by selecting the `classification` fragment group its properties and fragment forms will be presented in the group viewer. Since `classification` only contains 1 fragment form it is automatically presented in the code viewer:

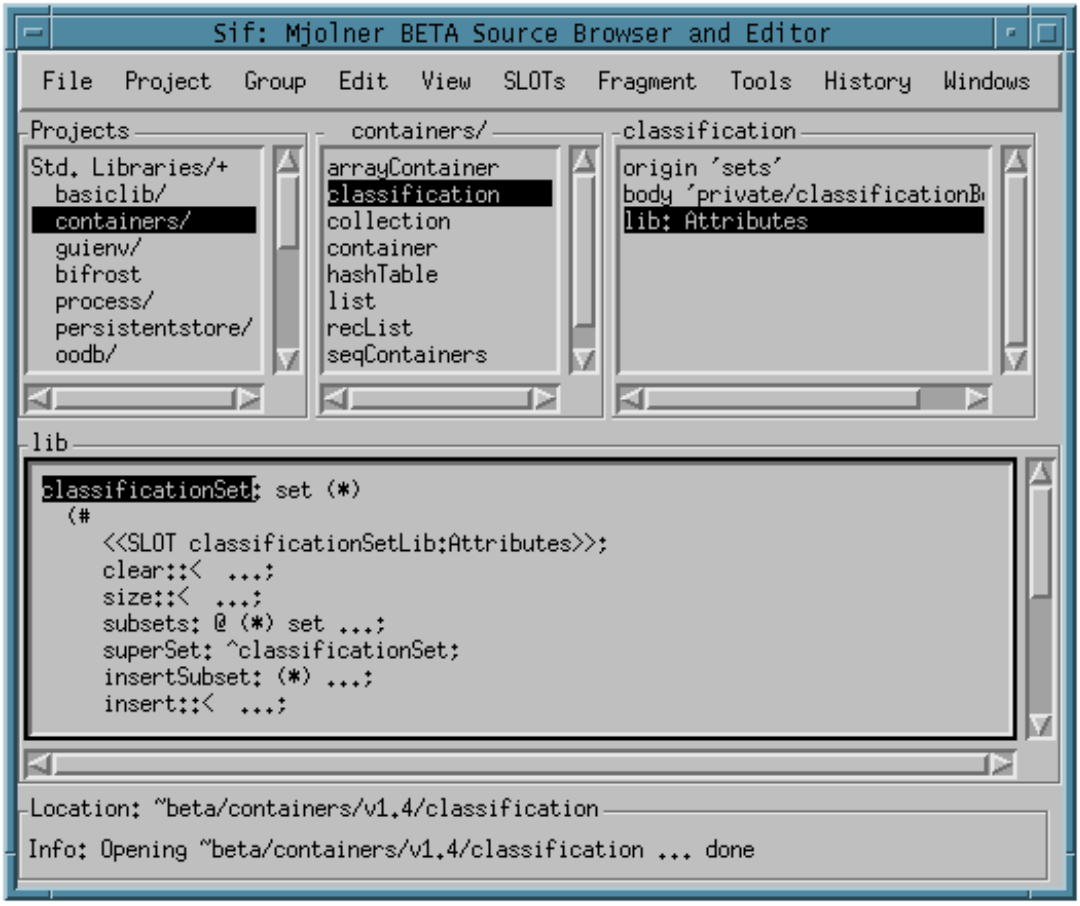


Fig 4.

3.3. Browsing at Group Level

Browsing at the group level is provided in the following way:

1) If an ORIGIN, BODY, MDBODY or INCLUDE entry is selected in a group viewer, the link to the specified group can be followed by double-clicking on it. The contents of the group viewer is replaced with the specified fragment group (if it exists). Shift-double-click gives a separate source browser.

2) If a fragment group presented in the group viewer only contains one fragment form it is automatically presented in the code viewer. When a fragment form is selected in the group viewer (by just clicking at it) it will be presented in the code viewer. Shift-double click gives a separate code viewer/editor.

3) The way links between SLOTs and fragment forms can be followed (is described separately under Fragment and SLOT Links)

The screen dump below is the result of double-clicking on the ORIGIN property in the group viewer of Fig. 4.

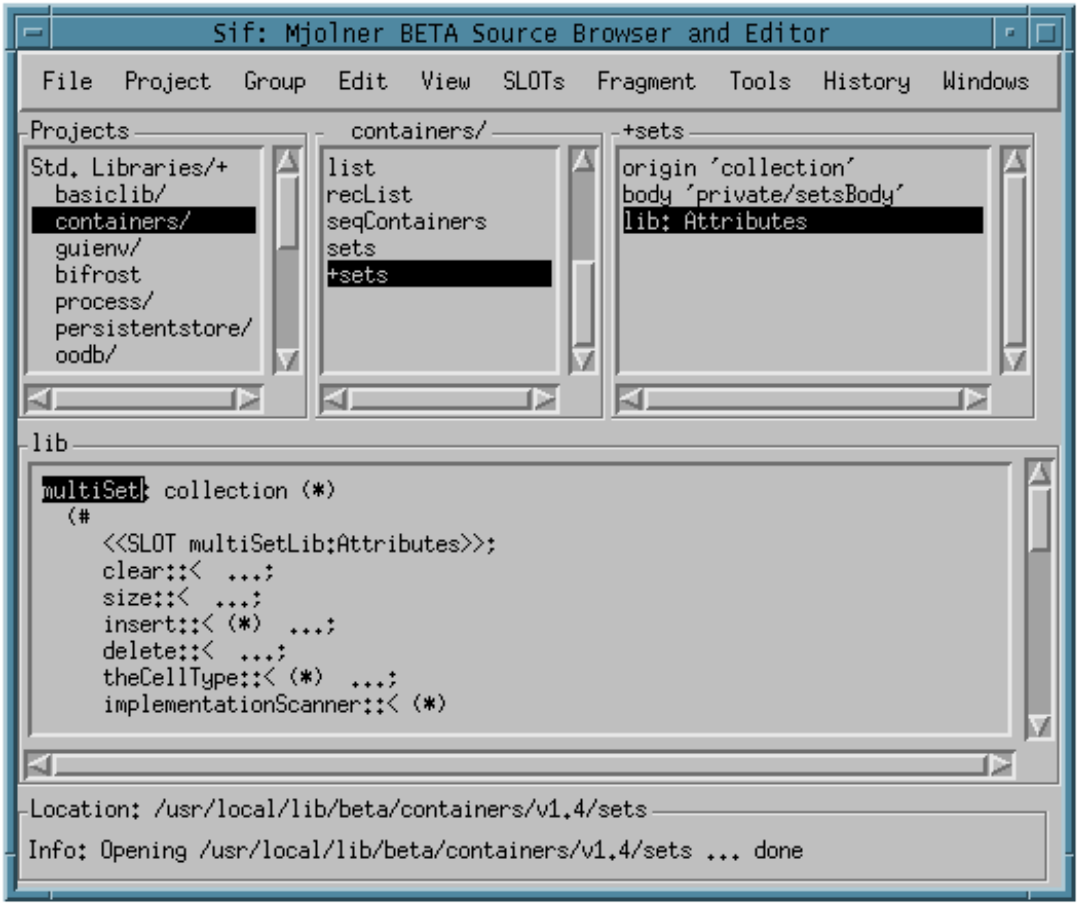


Fig. 5

3.4. Browsing at Code Level

3.4.1. Abstract Presentation

When a fragment is shown in the code viewer, abstract presentation is used. Instead of showing the program in full detail, 3 dots (**...**) are shown for certain syntactic categories. In the presentation of BETA programs these syntactic categories are `Descriptor`, `Attributes` and `Imperatives`.

Contraction

These 3 dots are called a *contraction*. By double-clicking on a contraction the next level of detail is shown. E.g. when double-clicking on the 3 dots belonging to `insertSubset` in Fig. 4, the result will be:

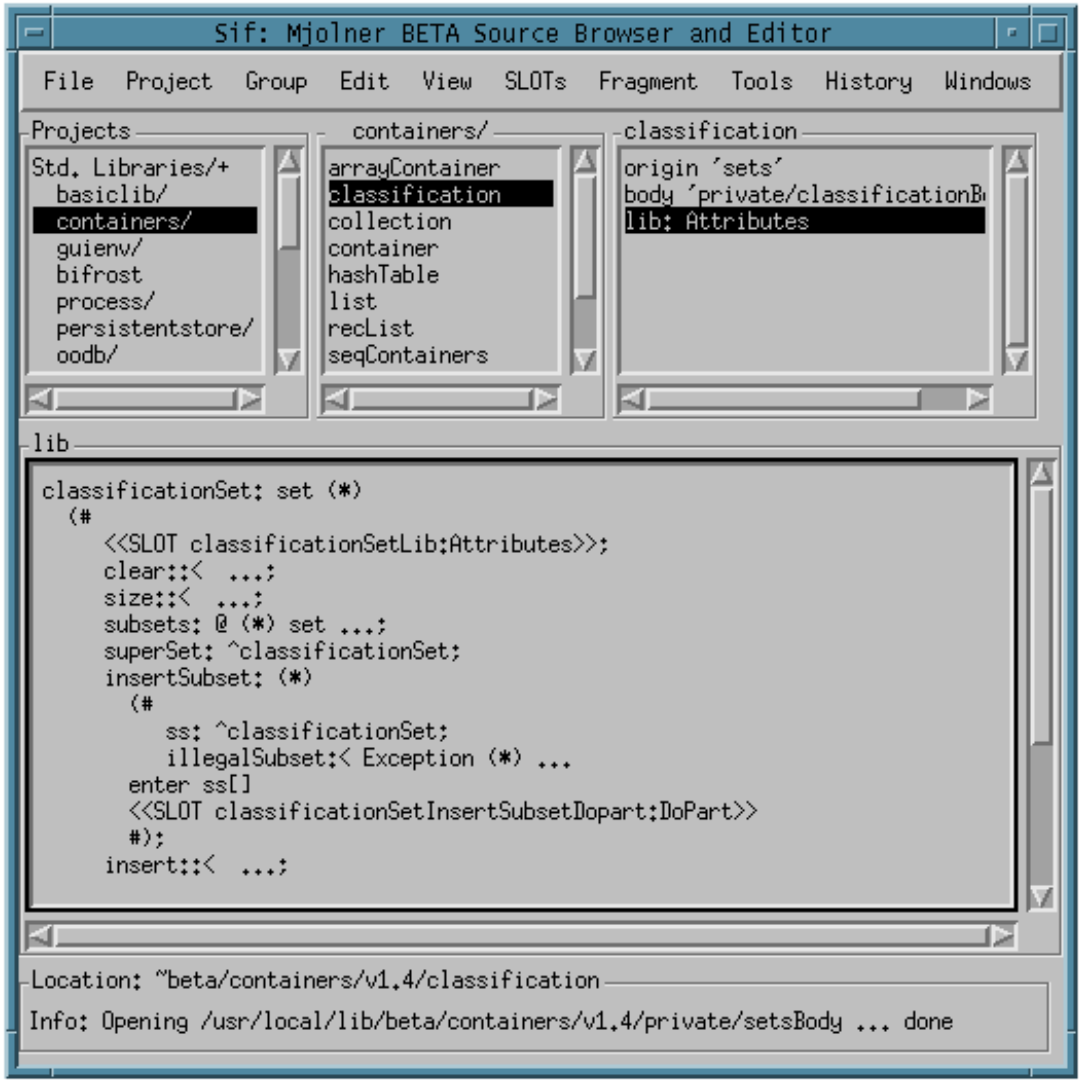


Fig. 6

Abstract presentation

Abstract presentation can be used to browse around in the program or to produce documentation. By means of the commands in the **View** menu or by double-clicking an appropriate abstraction level can be chosen. This presentation can then be written on a text file by means of the **Save Abstract...** command in the **Group** menu.

3.4.2. Semantic Links

This facility is only relevant for BETA programs. If the program has been checked it is possible to browse around in the semantic structure of the program. If, for example, a name application is selected in the code viewer, the corresponding name definition can be found using the **Follow Semantic Links** command of the **View** menu or by simple double-clicking on the name application.

If the definition is in another fragment, a code viewer is opened on that fragment. Fig. 7 is the result of double-clicking on `set` in the first line of the code viewer in Fig. 6.

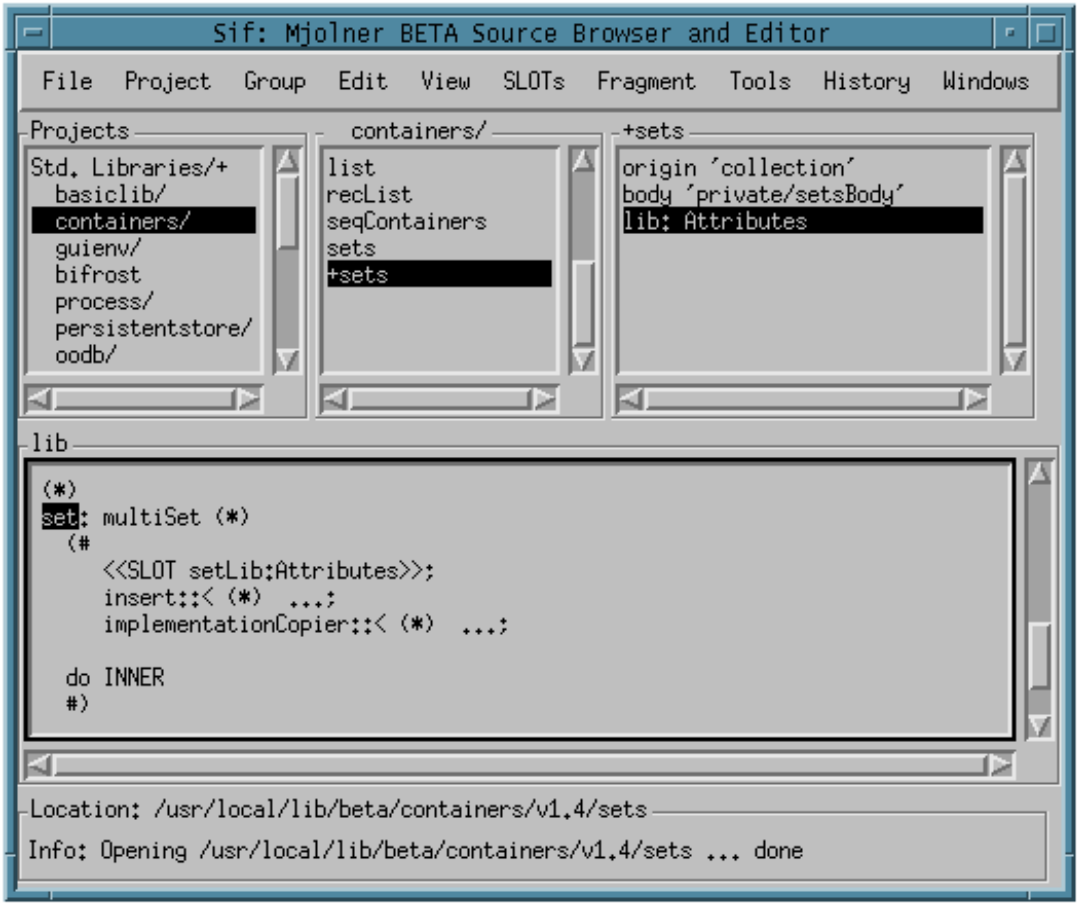


Fig. 7

If the program has not been checked, no semantic links are available. Or if the program has been modified, the semantic links may become inconsistent and the program then has to be re-checked. If this command is used and the program needs to be re-checked and a dialog box, that offers re-checking, is popped up.

3.4.3. Comments

In order to compress information in the windows, comments are not shown in the code viewer window, but instead a so-called *comment mark* **(*)** is shown. By double-clicking on such a comment mark, the comment mark is expanded and the whole comment is shown. Fig. 8 is the result of double-clicking on the comment mark after `multiSet` in Fig. 7:

Comment mark (*)

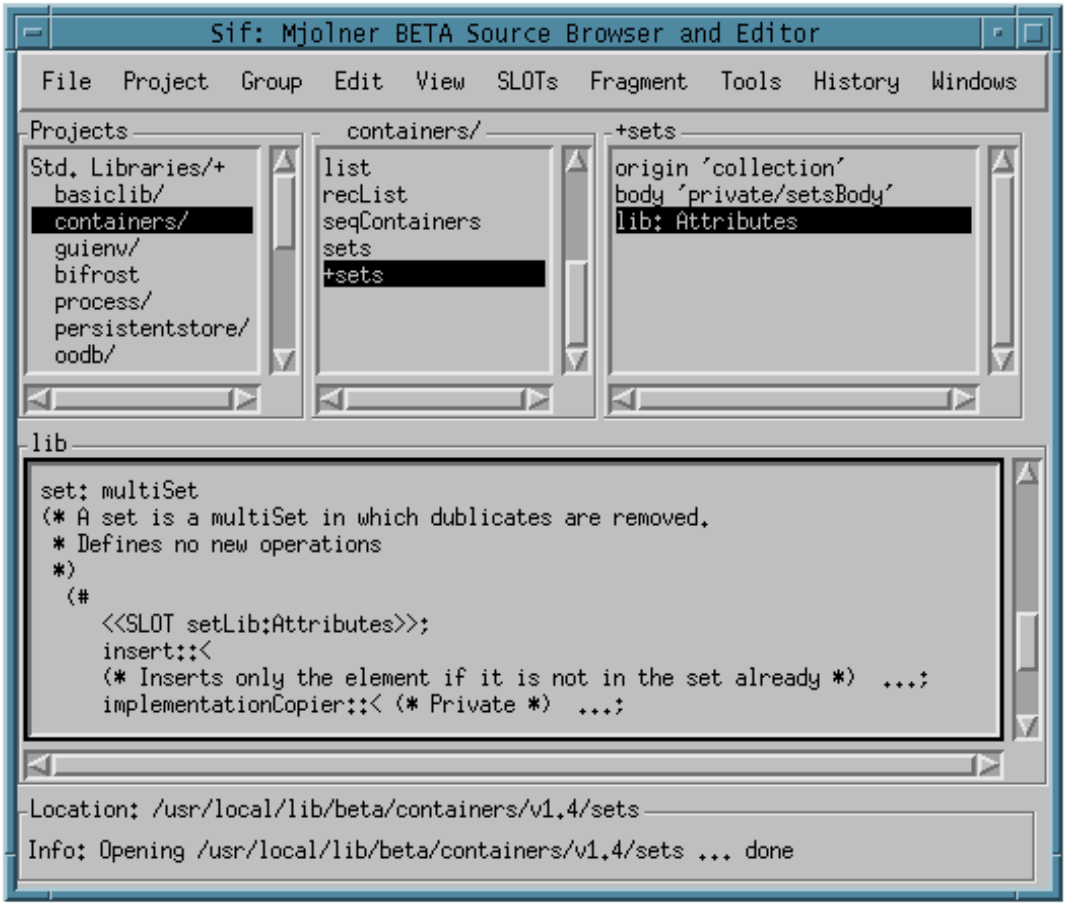


Fig. 8

Likewise the expanded comment can be collapsed to a comment mark again by double-clicking on it.

By shift-double-clicking on the comment mark the comment will be shown in a separate text editor window. The following window shows the comment associated with the `multiSet` pattern:

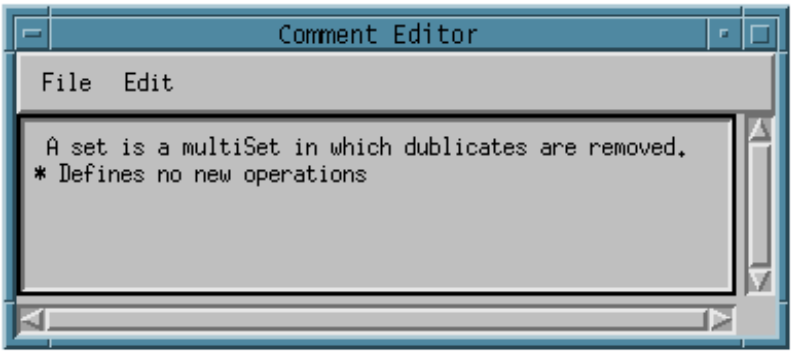


Fig. 9

The **Show Node** command in the **Edit Menu** of the comment window makes it possible to “find back”, i.e. navigate to the node this comment is associated with.

3.4.4. Fragment and SLOT Links

Follow link to fragment form

Given a SLOT definition, the link to the binding of the definition (i.e. a fragment form with the same name and type) can be followed.

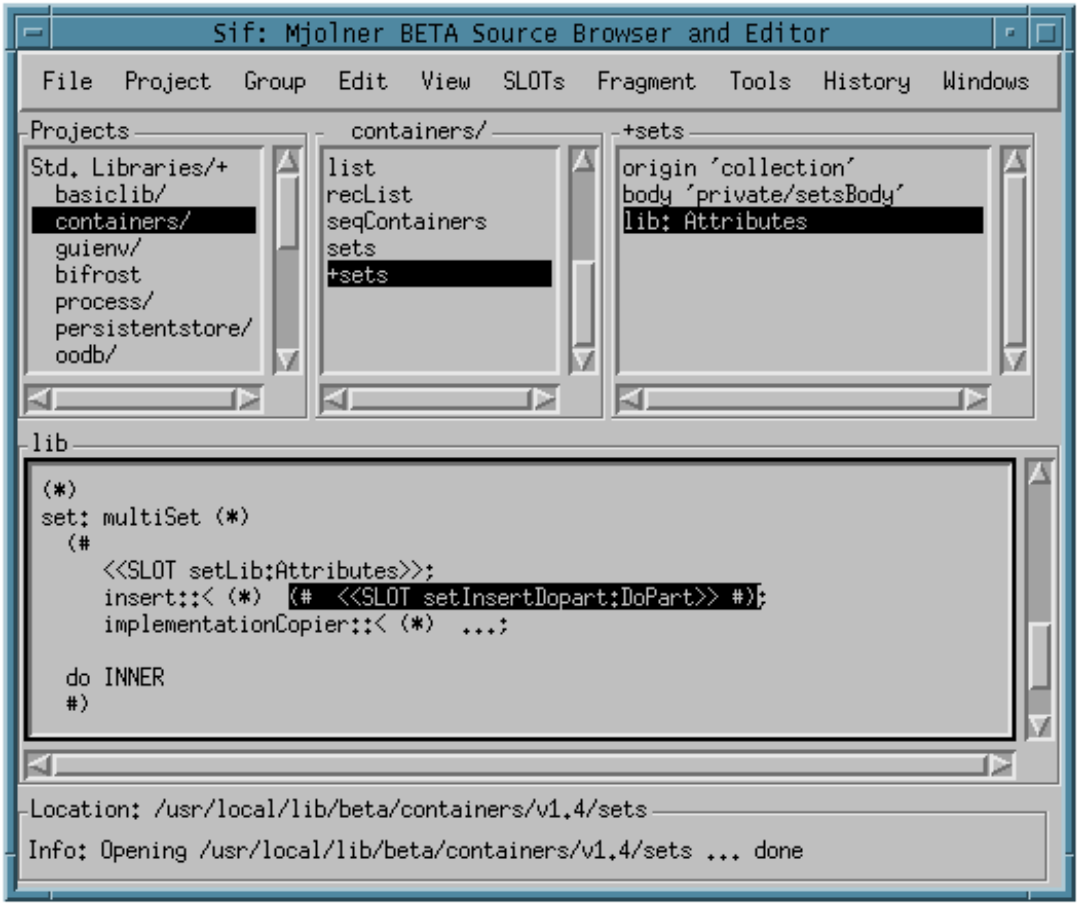


Fig. 10

This is done either by selecting the SLOT definition in a code viewer and the **Follow Fragment Link** command in the **View** menu or simply by double-clicking on the SLOT definition. The editor will now search for a fragment form with the same name in the BODY and MDBODY hierarchy of the group that the current fragment is part of. Doing this in the example above will result in the fragment shown in Fig. 11. If Shift-double-click is used a separate code viewer window is shown.

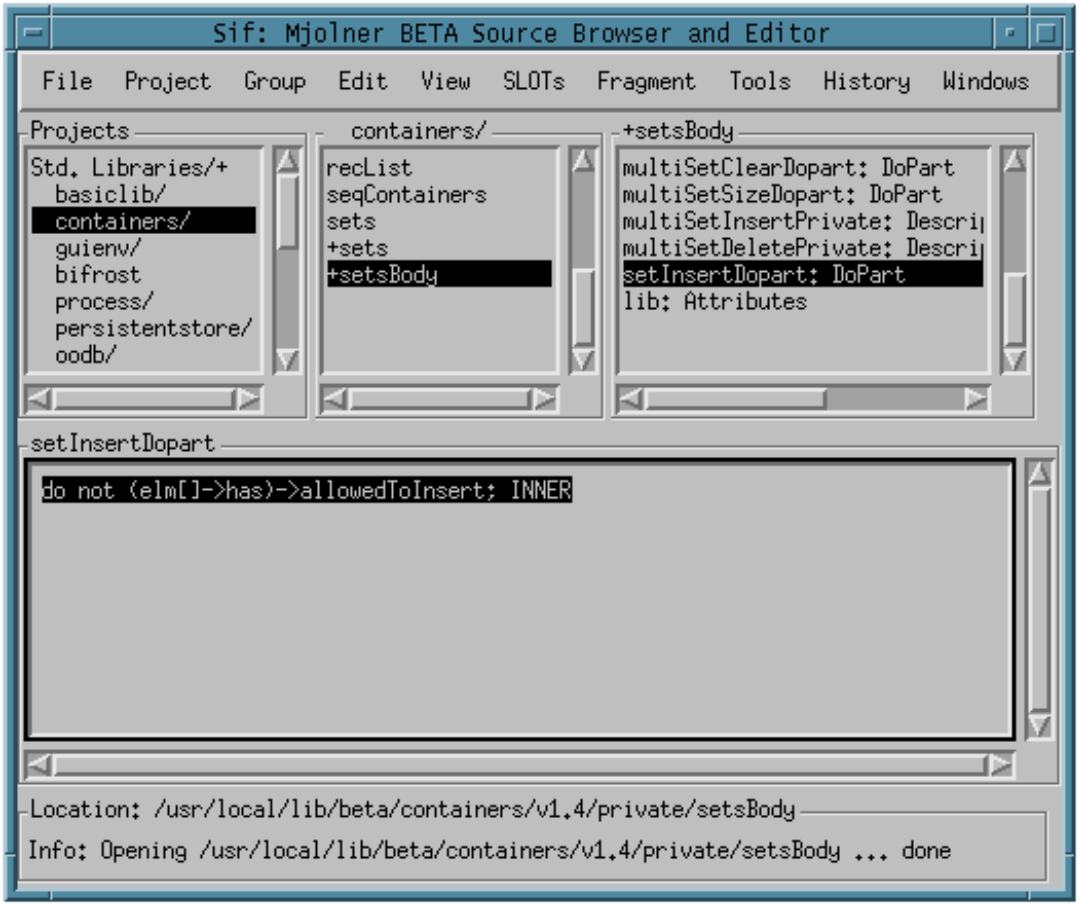


Fig. 11

Notice that during an editing session the fragment form may not be found since the program may be incomplete. Another restriction is that binding of `Attributes` SLOTs only can be found in this way if they are bound in the BODY and MDBODY hierarchy. If not, there is no automatic way of finding the bindings.

Follow link to SLOT definition

Given a fragment form, the link (possibly dangling) to the SLOT definition with the same name can be followed. This is done by selecting a fragment form and using the **Follow Link to SLOT** command in the **View** menu. The editor searches after a SLOT definition along the ORIGIN chain until it finds it or reaches the top, i.e. `betaenv`.

3.4.5. Searching

By means of the **Search** in the **View** Menu it is possible to search for a substring of a lexem, i.e. a name definition, a name application or a string.

4. Editor Tutorial

The editor consists of basically two types of editors: a *group editor* and a *code editor*.

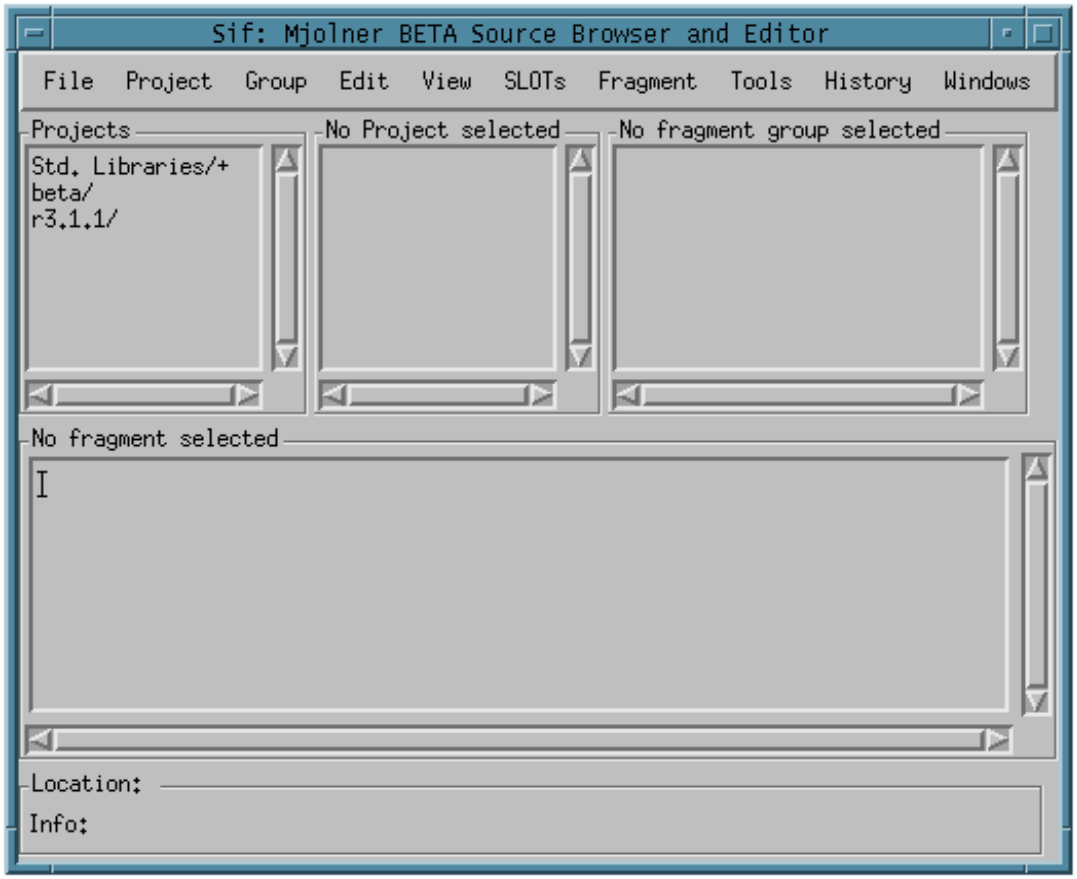


Fig. 12

The Group editor **Group editor**

The group editor is only relevant if the language in question is BETA. It is used for browsing or editing BETA programs. It is used to present and modify the structure of a group, i.e. the properties (ORIGIN, BODY, MDBODY, INCLUDE etc.) and fragments (Descriptor forms, DoPart forms or Attributes forms).

The Code editor **Code Editor**

The code editor provides structure editing on each fragment form.

4.1. Creating a New Program

By selecting the **New BETA Program...** of the **Group** menu the following standard dialog pops up:

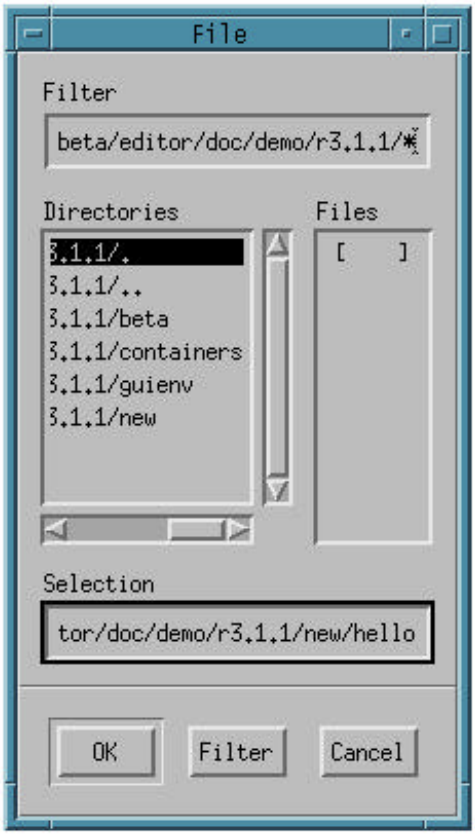


Fig. 13

After specifying the path and the name of the fragment group to be created, a minimal BETA program is presented in the browser window:

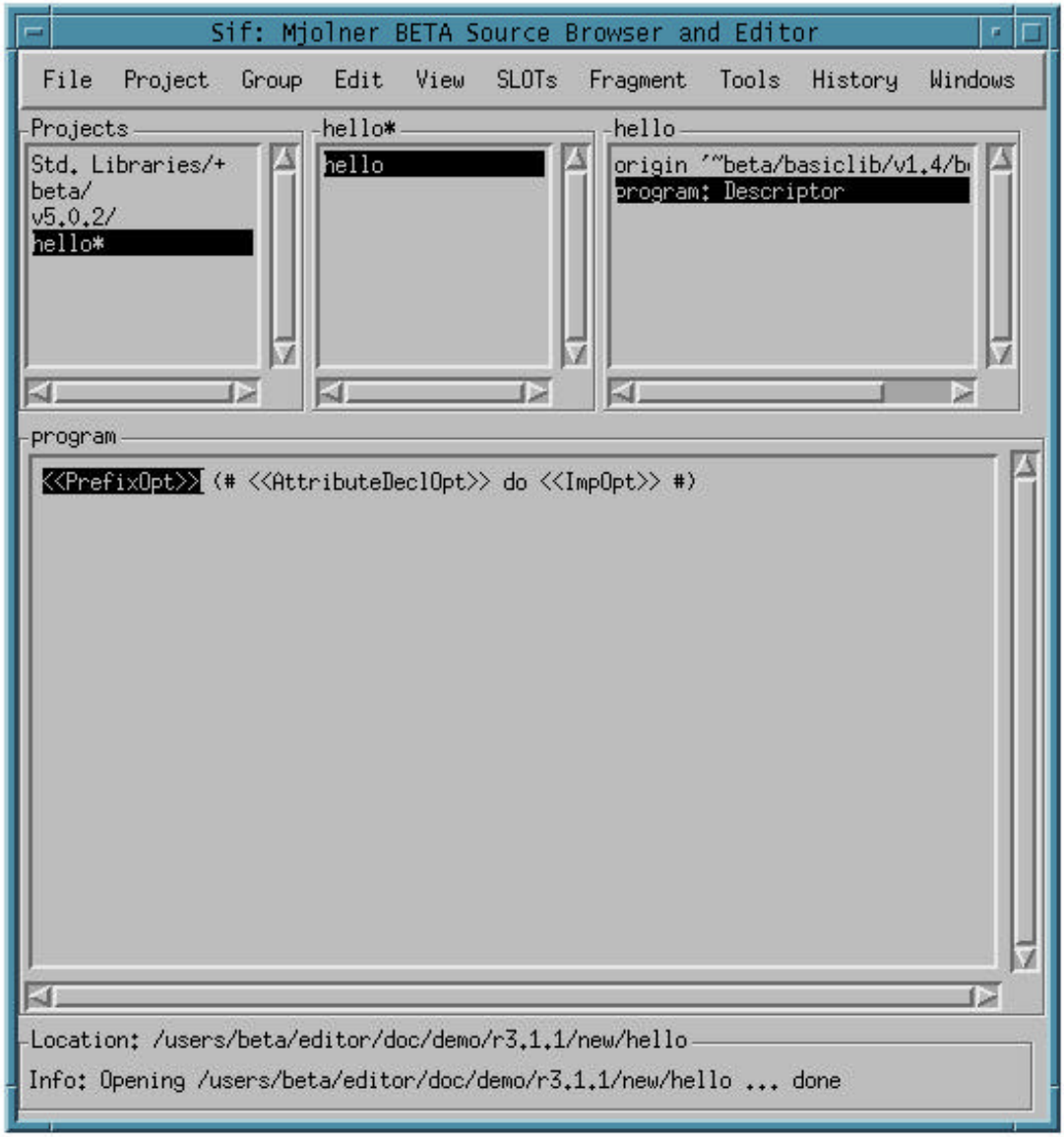


Fig. 14

4.2. Editing at Code Level

4.2.1. Code editor

The code editor provides structure editing on each fragment form. In the following a separate code editor is chosen by shift-double-clicking on the `program` fragment form in the group editor of Fig. 14.

4.2.2. Structure Editing

The basic idea of structure editing is that the program is manipulated in terms of its logical structure rather than the textual elements such as characters, words and lines. The advantage of this approach is that only logically coherent parts can be inserted or deleted and thereby preserving the syntactical rules of the language at any time.

Expanding nonterminal

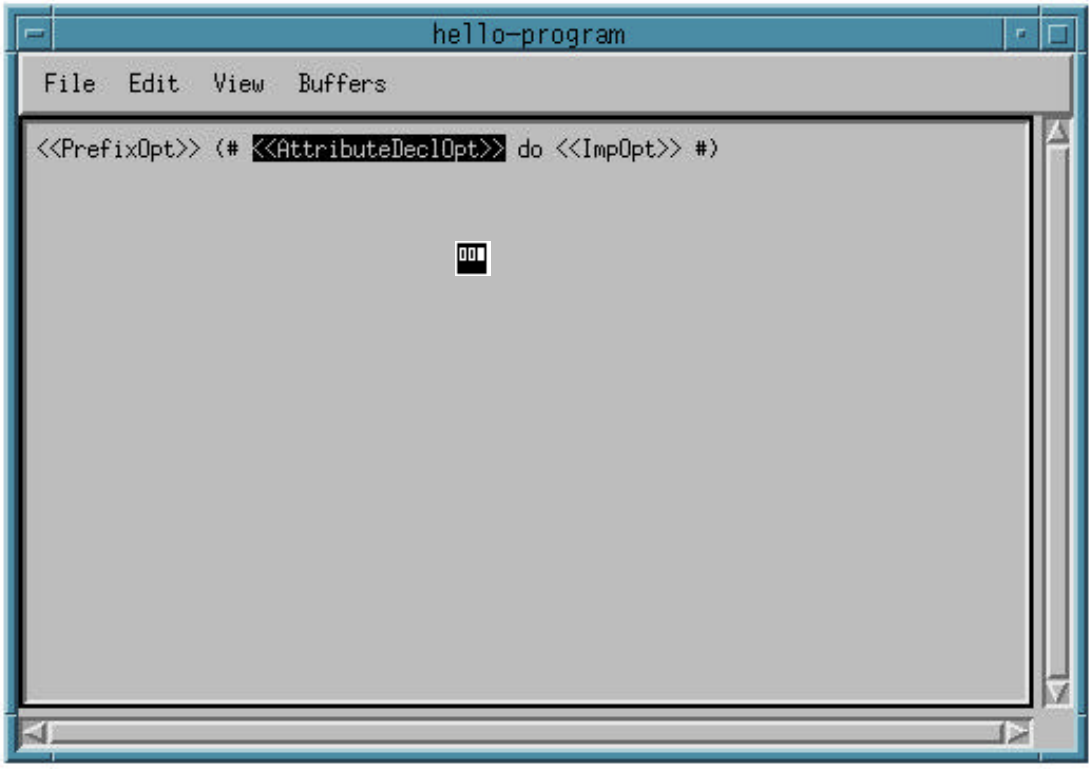


Fig. 15

The window above shows an example where a template for a BETA pattern has been derived. The template includes placeholders (nonterminals) and keywords (terminals). If a nonterminal is selected the mouse pointer is changed to an icon that indicates that the rightmost button of the mouse must be pushed to pop-up a menu (only Unix version, in general the *Pop-up Menu Button*. is used, see *Basic User interface principles*). In the example, the `<<AttributeDeclOpt>>` placeholder has been selected and the legal declarations (according to the BETA grammar) is shown in the pop-up menu.

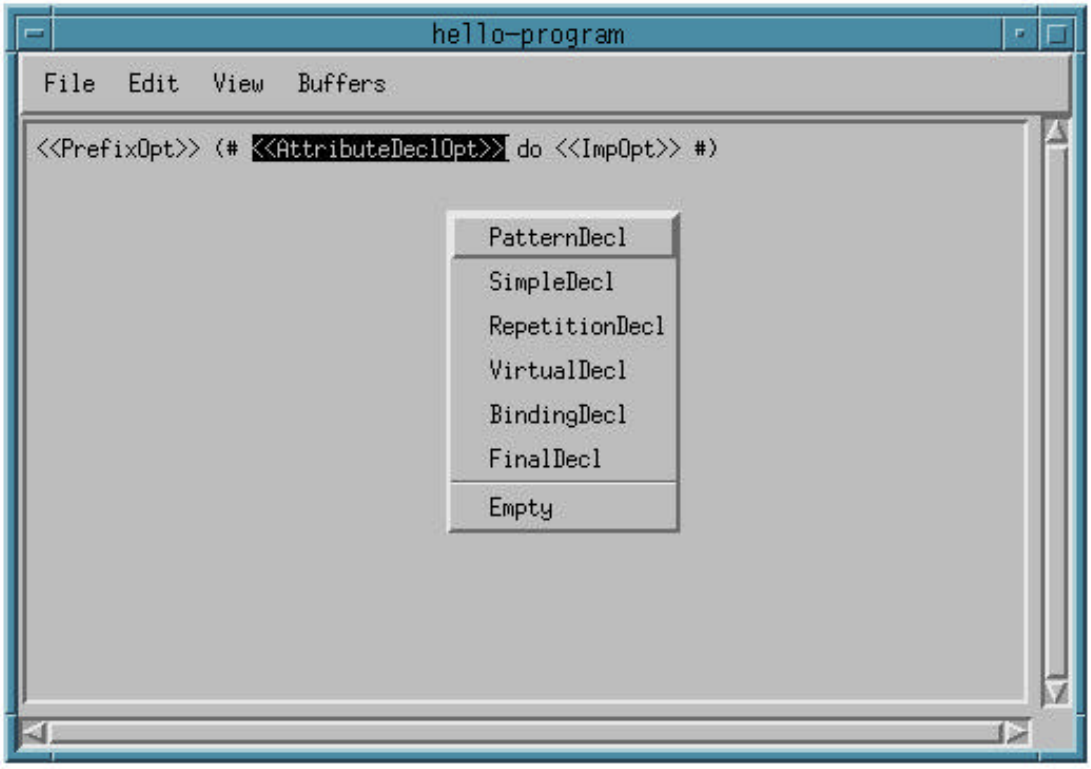


Fig 16

The `PatternDecl` entry is selected and the result is:

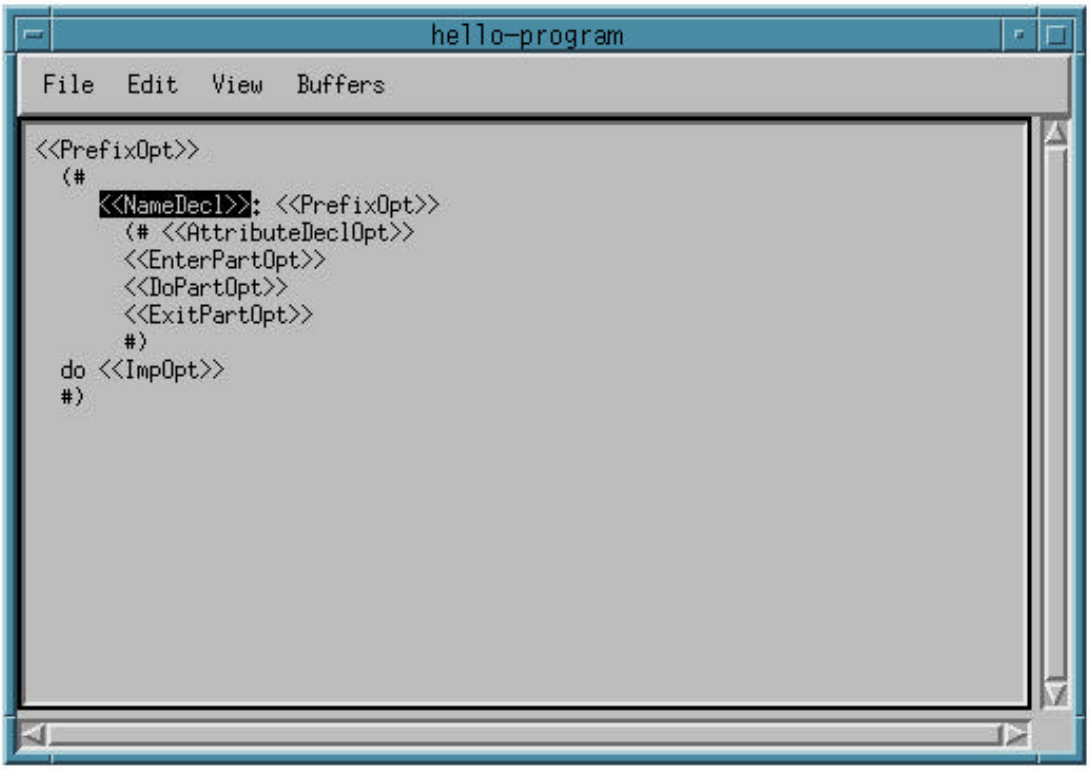


Fig. 17.

4.2.3. Text Editing and Parsing

Structure editing has its greatest force at the higher levels of editing, i.e. for creating the overall structure of the program or for moving around large chunks of code. At the detailed level the textediting technique is more useful.

Text editing can at any time be used as an alternative to structure editing. Text editing is activated either by just starting to type or by selecting the **Textedit** command in the **Edit** menu. If you start typing at the keyboard, the typed characters will replace the current selection in the code editor window. Text editing mode may alternatively be entered without deleting the current selection, by means of the **Textedit** command or by pressing the <space> key. In that case the text cursor will be positioned in the start of the current selection.

Text editing mode can be terminated by selecting the **Parse Text** command in the **Edit** menu, or simply by selecting outside the text editing area. The possibly modified text will immediately be parsed and any parse errors will be reported (but only one at a time). Note that semantic checking is **not** done by the editor. In this example a parse error is detected

In Fig. 17 the placeholder <<`NameDecl`>> is selected and the name `hello` is typed (See Fig. 18). After that the <<`DoPartOpt`>> is selected and the text

```
'hello world -> putlin
```

is typed. When textediting is exited for example by clicking outside the textediting area, the syntax error is immediately reported:

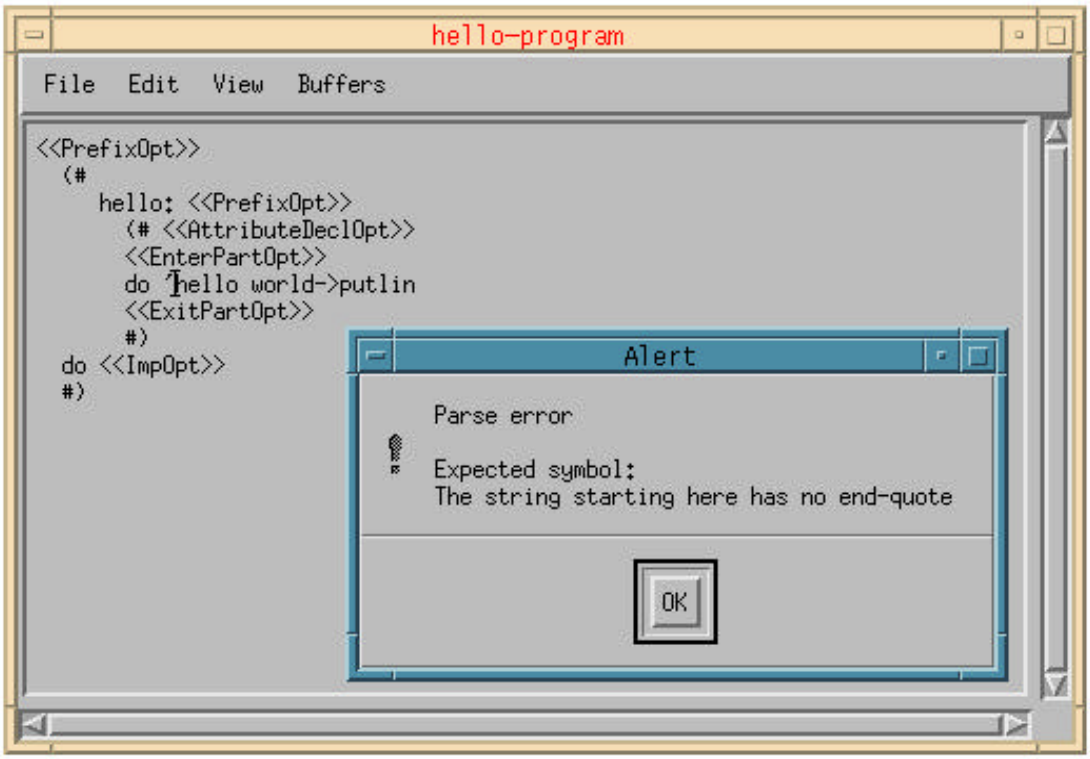


Fig. 18.

4.2.4. Checking

After correcting the syntax error the program looks like below:

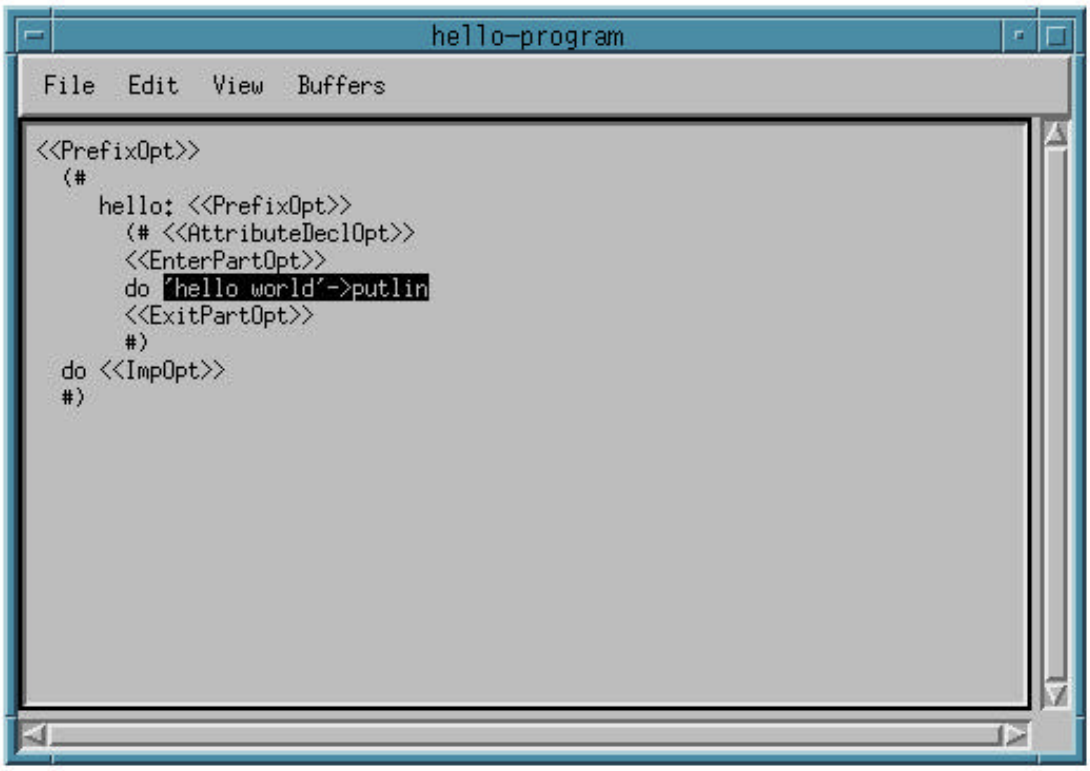


Fig 19

Now we want to call the checker. This is done by means of the **Check Current** command in the **Tools** menu. But the compiler does not accept unexpanded nonterminals. Therefore the following dialog is popped up:

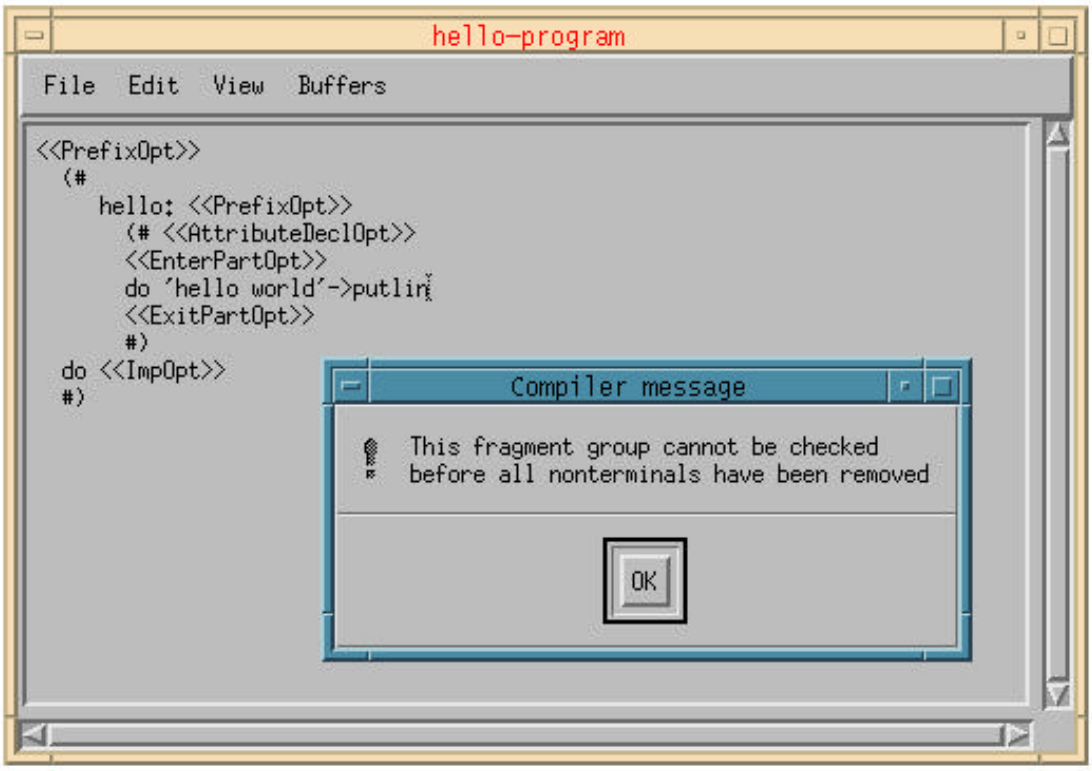


Fig. 20

Notice that in this example all nonterminals are optionals (is indicated by the Opt suffix). An easy way to remove unexpanded optionals is to select the whole program and use the **Remove Optionals** in the **Edit** menu. The result is:

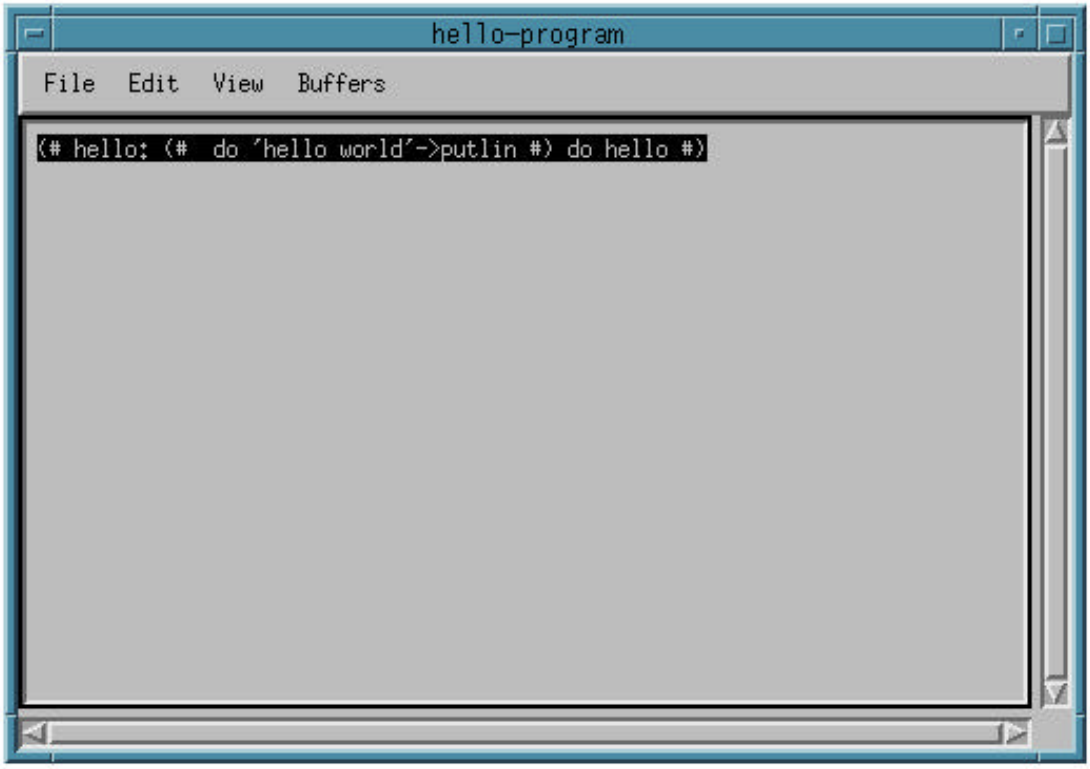


Fig. 21

Now the checker is called again and the semantic error is detected and shown by means of the semantic error viewer:

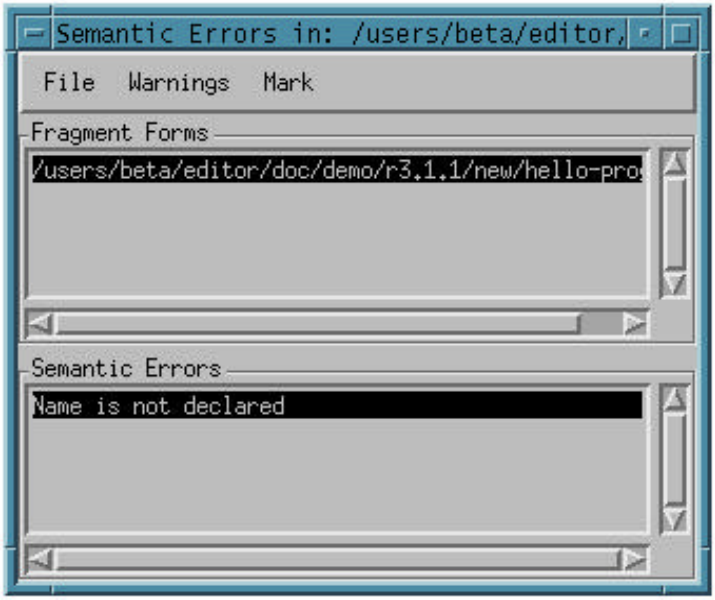


Fig. 22

In this case there is only one semantic error, but in general the Fragment Forms pane will contain a list of fragment forms with semantic errors and the Semantic Errors pane will for each fragment form in the upper pane show a list of semantic errors. By selecting in the two panes the different semantic errors can be inspected. By clicking in the Semantic Errors pane the code editor will select the corresponding structure. The first semantic error will always be selected automatically. In the example the following selection is made:

Fig. 23

4.3. Modifying a Program

Consider the program of Fig. 23. We now want to add an enter part that should enter a text to be printed after the `'Hello world'` string. To add an enter part you can either select for example the descriptor of the `hello` pattern and type the enter part using text editing or you can select the for example the whole pattern declaration and use the **Show optionals** command in the **Edit** menu. The latter has been done below:

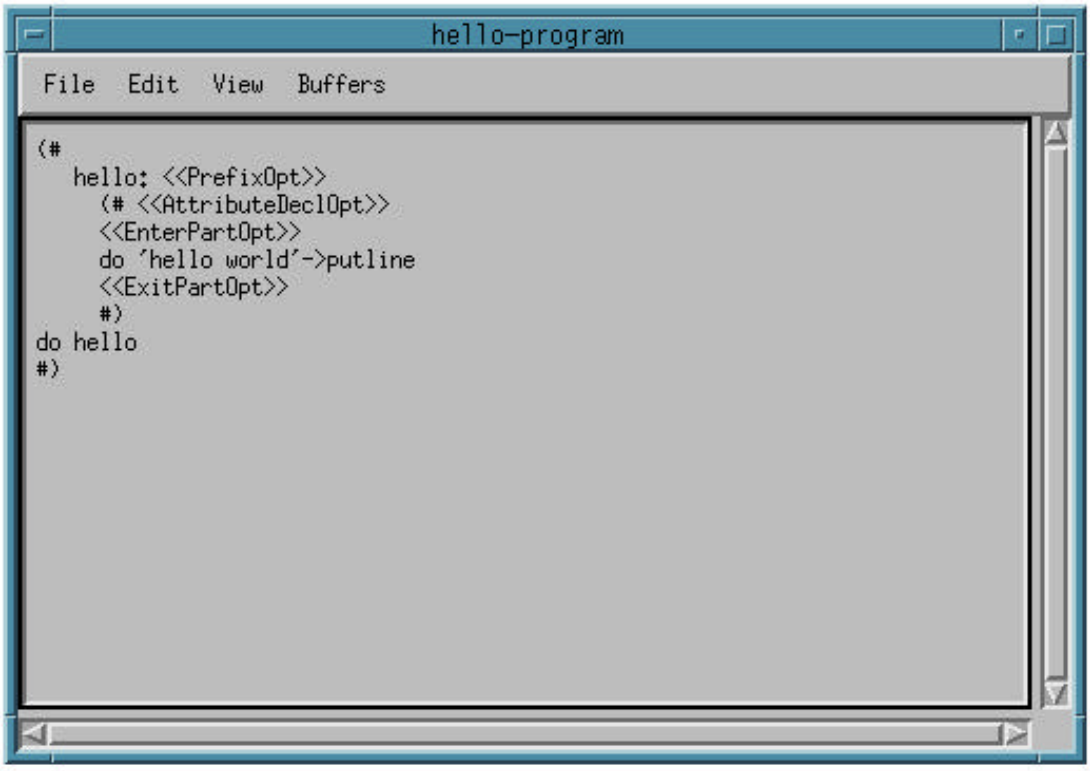


Fig. 24

Using textediting we select the `EnterPartOpt` nonterminal and type: `enter t` and then we select somewhere inside the `AttributeDeclOpt` nonterminal and type: `t: @text`. Now we want to add an imperative after the putline. To insert a new list element in a list element is selected and the **Insert After** or **Insert Before** command of the **Edit** menu is used. An alternative to **Insert After** is to press the <cr> key.

In Fig. 24 the imperative `'hello world'->putline` is selected e.g. by clicking on the arrow (->). Then the <cr> key is pressed and the result is:

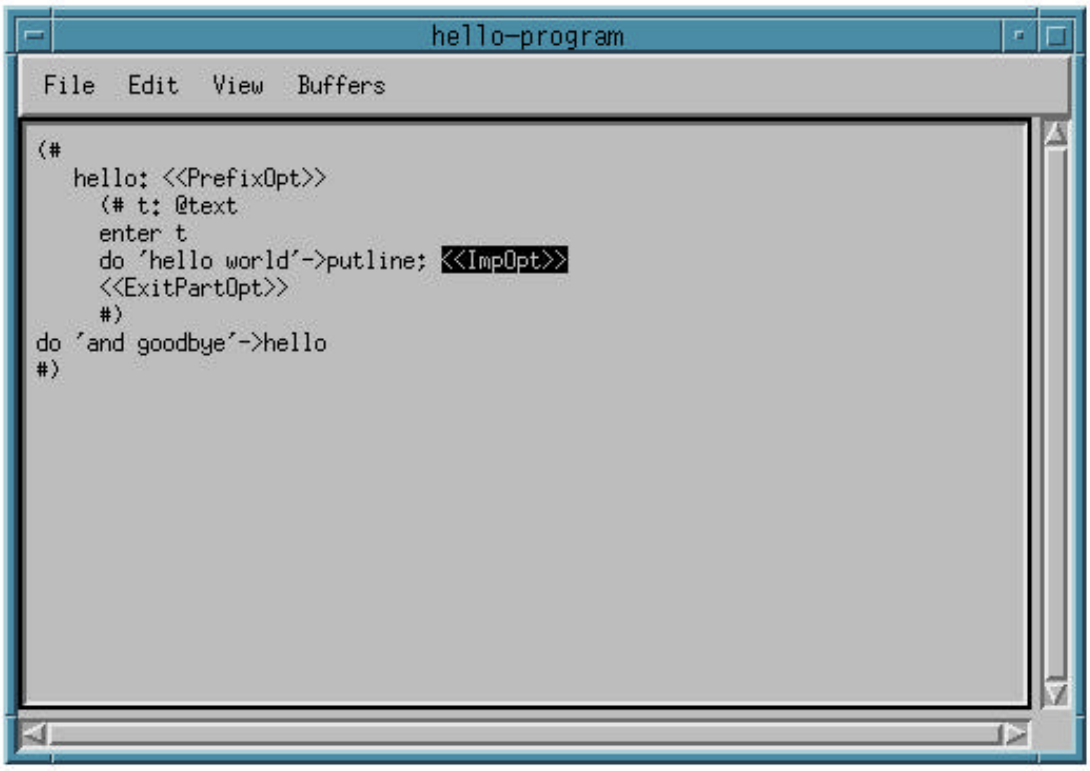


Fig. 25

The final result is:

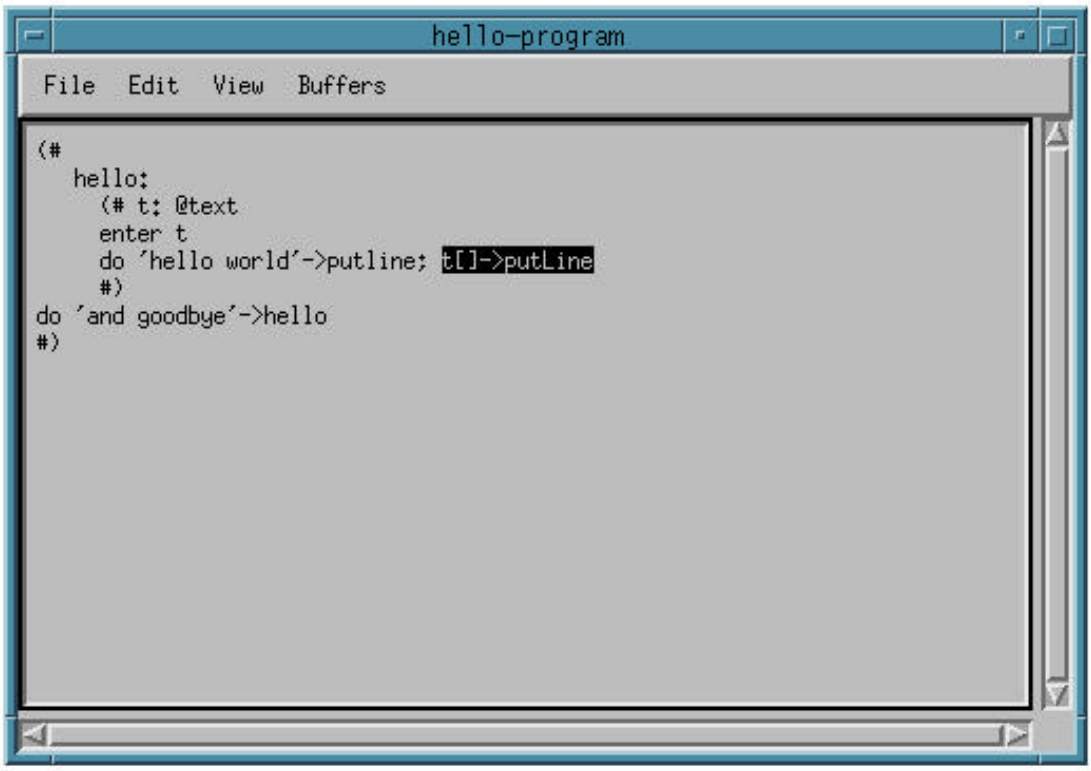


Fig. 26

4.4. Editing at Group Level

4.4.1. Group Editing

Manipulation of the fragment group structure is done in the group editor. It is possible to insert or delete fragment forms, modify the names of the fragment forms, or edit the properties of the fragment group e.g. the ORIGIN, INCLUDE and BODY properties. Editing of properties is done using a property editor that is a structure editor on the properties.

4.4.2. Fragmenting

The fragment system provides facilities for splitting a BETA program into several parts in order to support separation of interface and implementation, variant configuration or separate compilation. The code editor supports this kind of fragmenting.

If, for example, a fragment form is going to be divided into an interface part and an implementation part the, **SLOTs** menu is very useful. The DoPart of a procedure pattern can, for example, be replaced by a SLOT definition and the removed part can be inserted into another fragment group (the implementation part). Consider the following example, where the DoPart of the procedure pattern `hello` is selected.

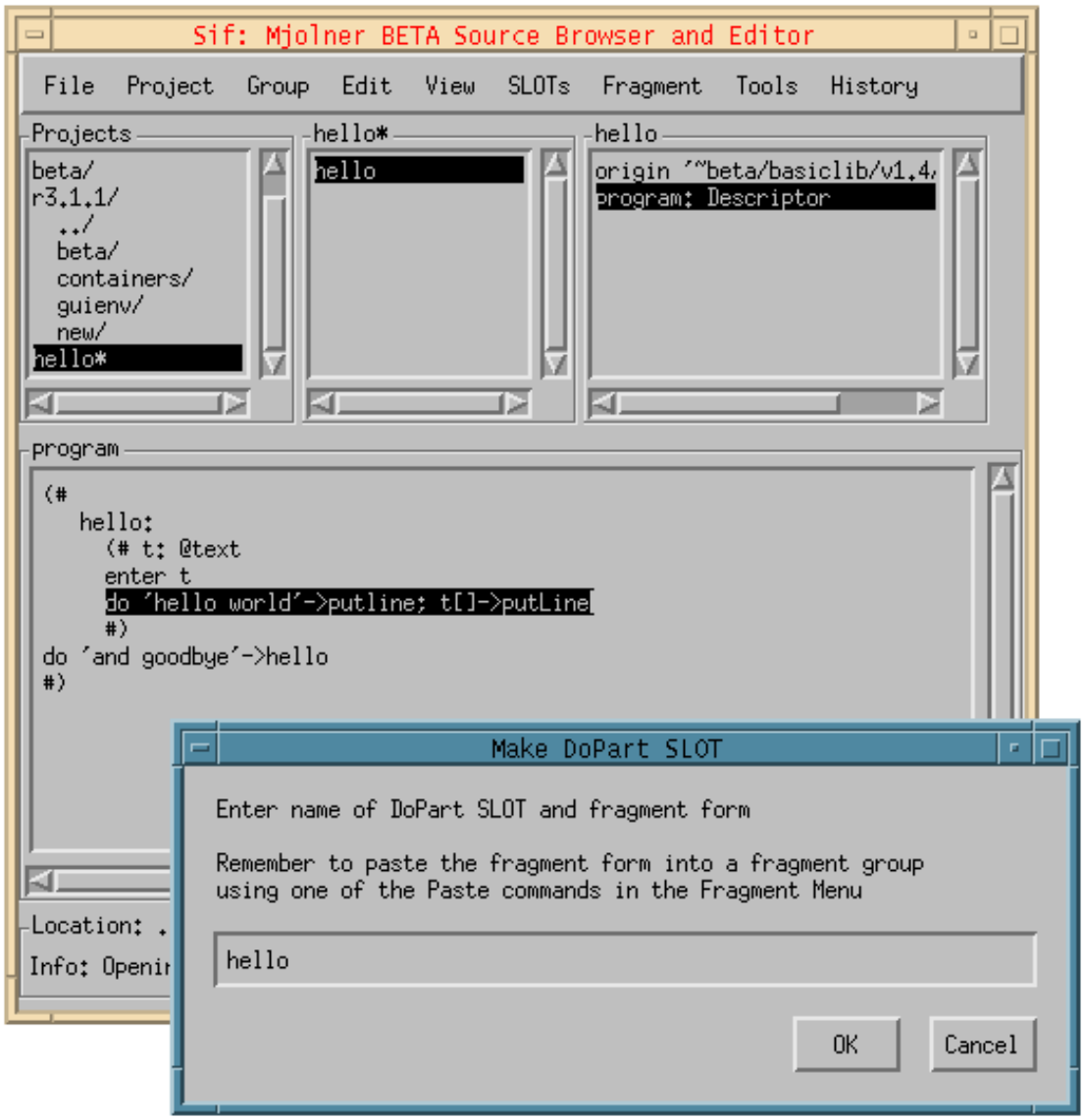


Fig. 27

By means of the **Make Dopart SLOT** in the **SLOTs** menu, the DoPart can be replaced by a SLOT definition. The name of the SLOT is prompted for. The default name is the name of the enclosing pattern. The removed doPart is inserted in a new fragment form that is stored on a so-called *fragment clipboard*. As mentioned in the dialog the created DoPart fragment form must be pasted into a fragment group (this can be done automatically see later). The result is (for simplicity we have pasted the fragment form into the same fragment group, this is not the usual case):

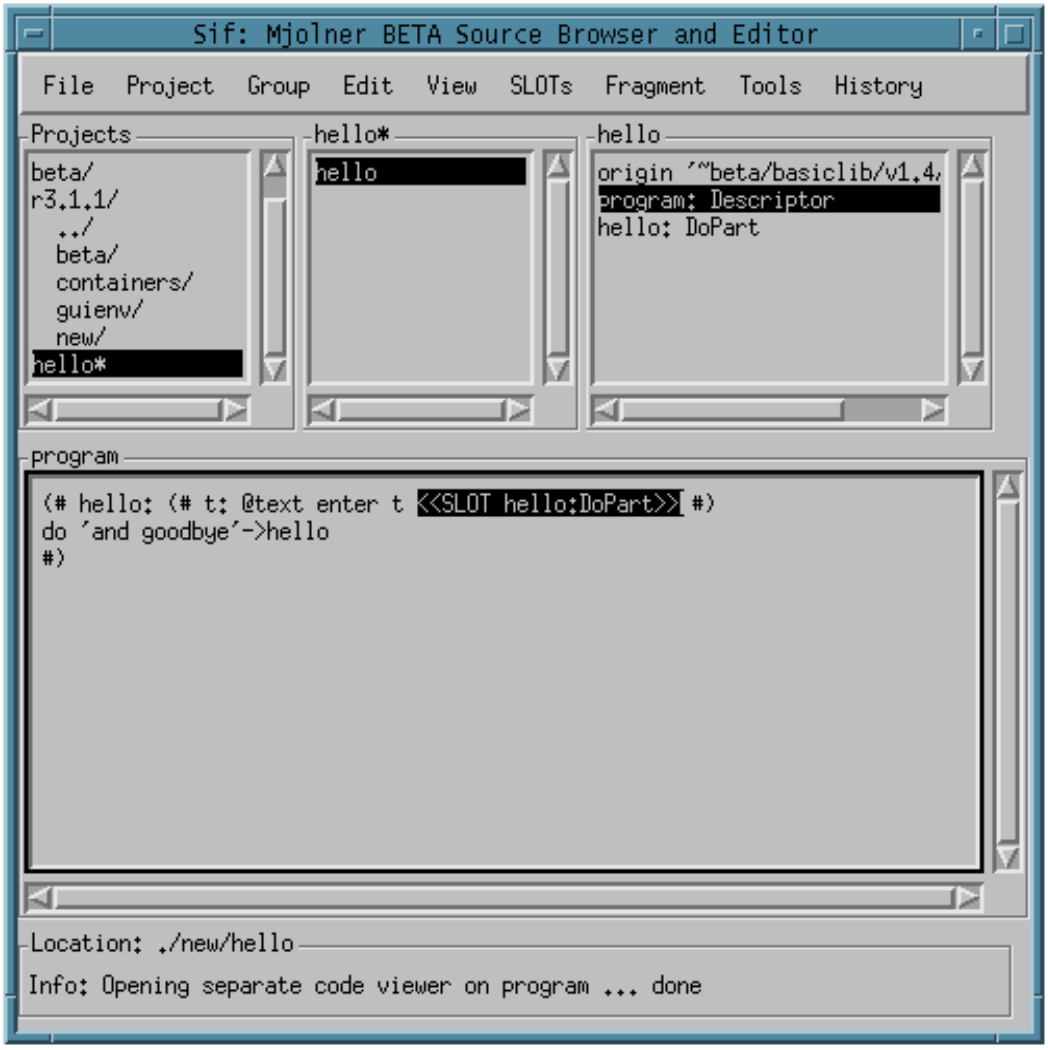


Fig. 28

This was the most simple way of moving a moving a DoPart into a implementation file and inserting a SLOT in the interface file. The implementation file can also be specified using the **Set Current as Implementation File** command in the **Fragment** menu. In this case the created fragment form is automatically pasted into the implementation fragment group. The **Hide Implementation...** command in the **SLOTS** menu is a powerful command that traverses the current selection in a code editor and for each DoPart you are asked whether it should be put into a DoPart fragment form. In the example below the whole program has been selected and the **Hide Implementation...** command is called.

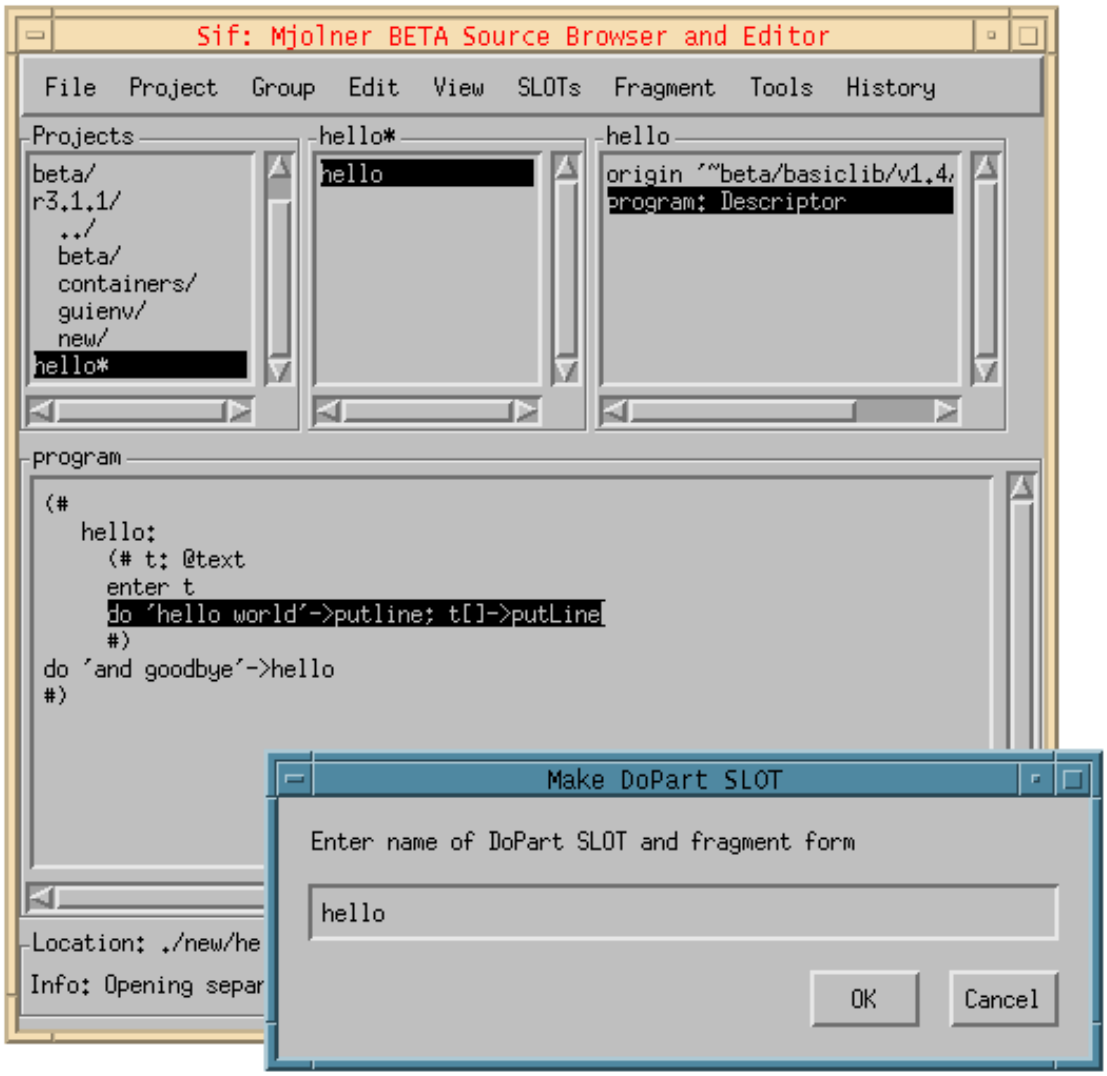

Sif: Mjolner BETA Source Browser and Editor
File Project Group Edit View SLOTs Fragment Tools History
Projects
beta/
r3.1.1/
../
beta/
containers/
guienv/
new/
hello*
hello*
hello
hello
origin '~beta/basiclib/v1.4/
program: Descriptor
program
(#
hello:
(# t: @text
enter t
do 'hello world'->putline; t[]->putLine
#)
do 'and goodbye'->hello
#)
Location: ./new/he
Info: Opening separ
Make DoPart SLOT
Enter name of DoPart SLOT and fragment form
hello
OK
Cancel

Fig. 29

4.5. Work Space

Consider the following interface file:

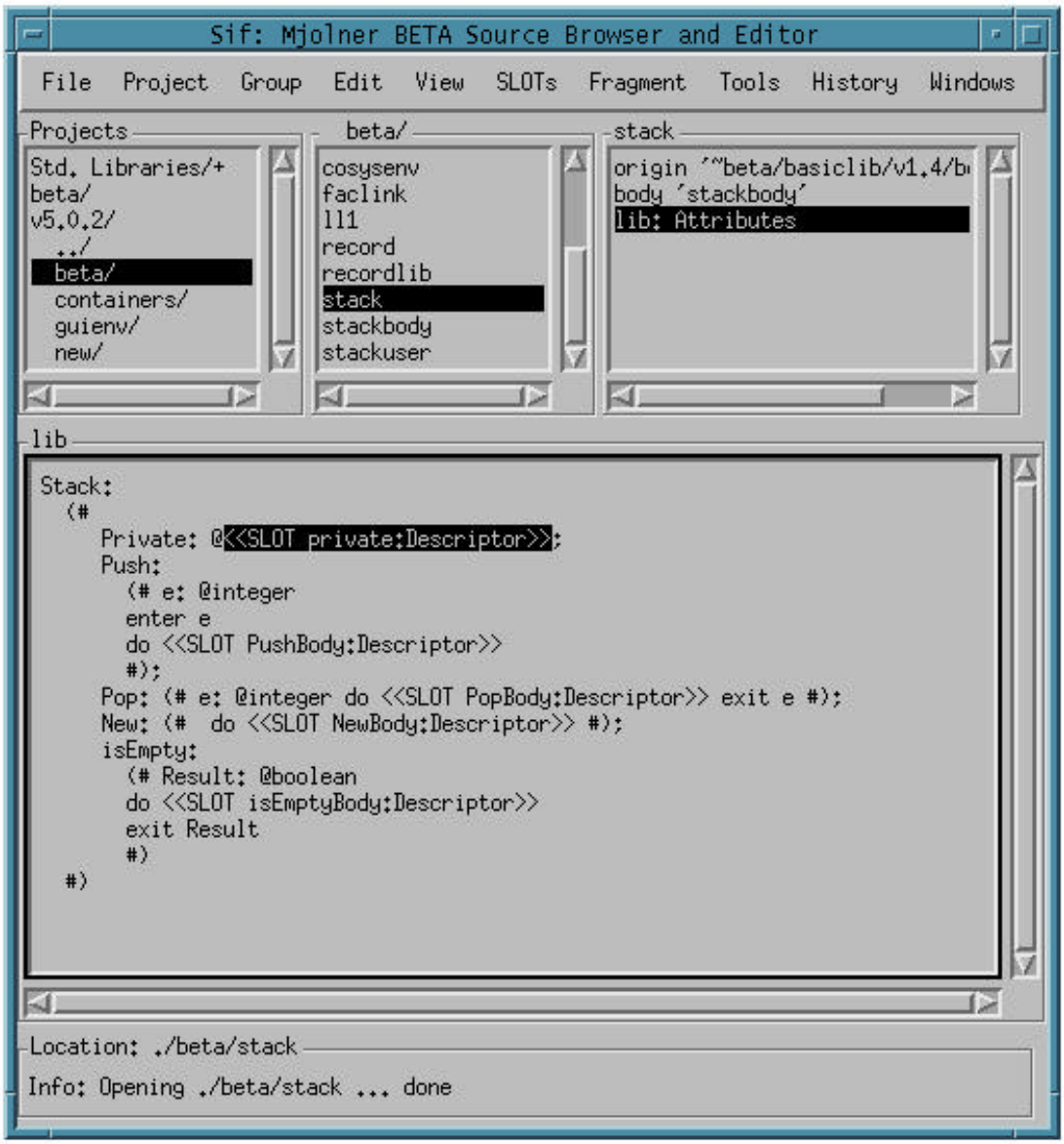


Fig. 30

Each fragment form in the implementation file can be shown one at a time in the browser either by double-clicking on the SLOTs or by double-clicking on the BODY property in the group editor window to get the implementation file and then selecting them one at a time:

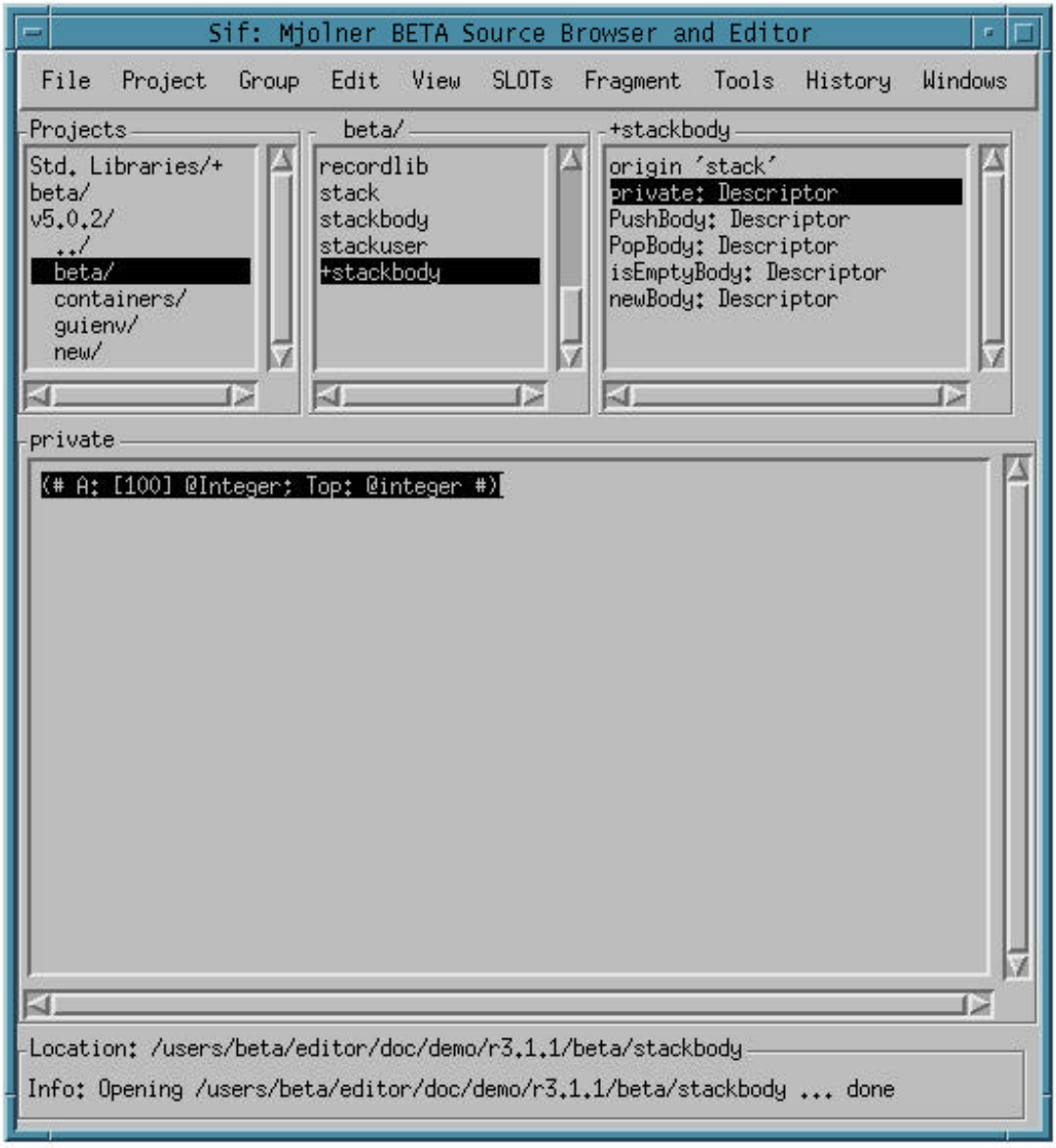


Fig. 31

but an easier way is to get all or a subset of the fragment forms in a so-called work space window by using the **Workspace** command of the **Windows** menu. The **Import** menu is used to open a code editor on all or a subset of the fragment forms in the workspace window:

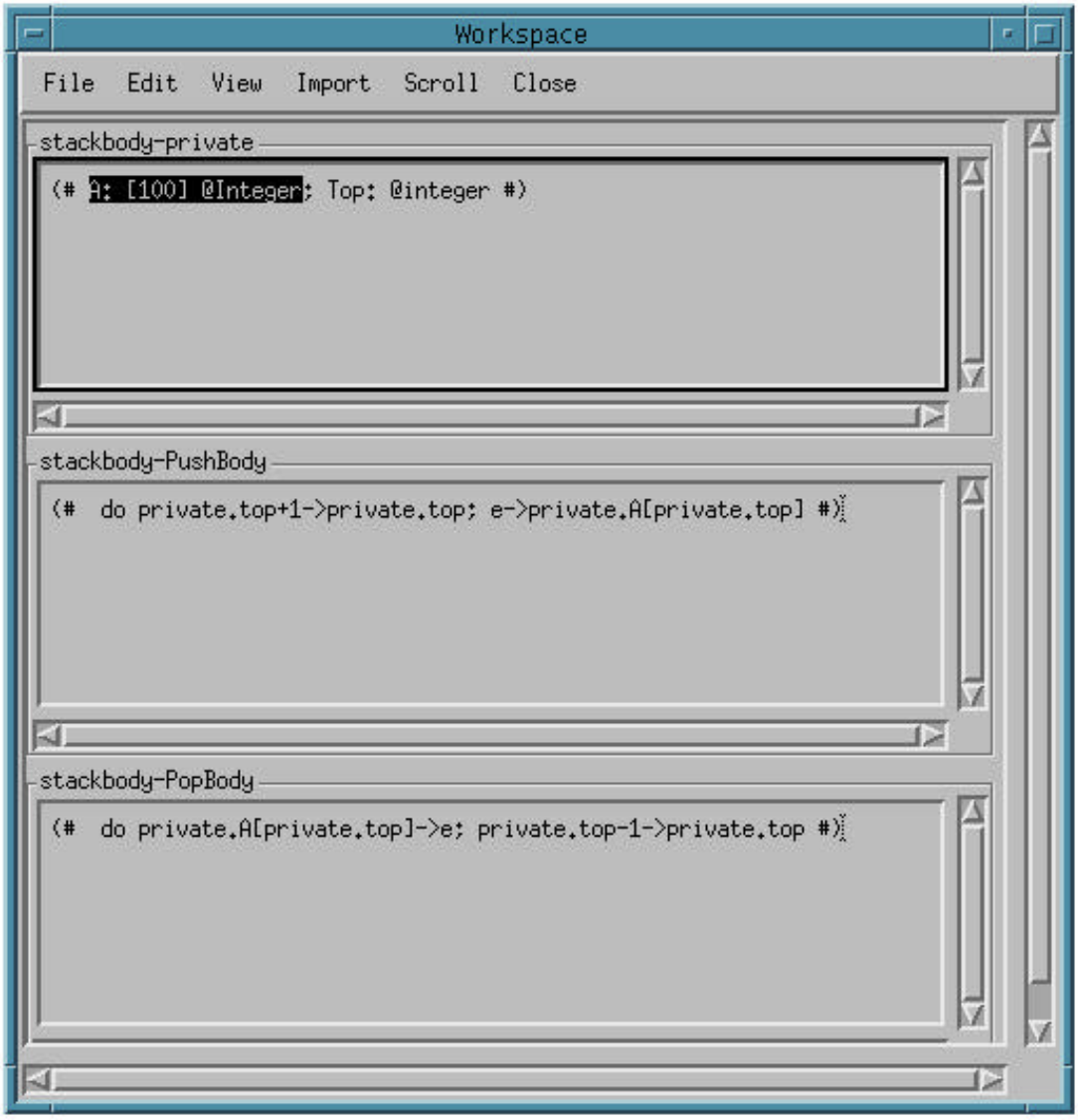

Workspace
File Edit View Import Scroll Close
stackbody-private
(# A: [100] @Integer; Top: @integer #)
stackbody-PushBody
(# do private.top+1->private.top; e->private.A[private.top] #)
stackbody-PopBody
(# do private.A[private.top]->e; private.top-1->private.top #)

Fig. 32

5. Source Browser Reference Manual

5.1. How to Get Started

The source browser that is part of Sif, is activated by writing one of the following (UNIX):

Activating Sif

1) sif
2) sif myFragmentGroup
3) sif myFragmentGroup.bet
4) sif myFragmentGroup.ast

In each case a source browser window appears on the screen.

1) If there are no arguments to Sif the project list pane will show the users home directory and possible projects defined in ~/.ymer.pjt (See Project definitions in this manual)

2)-4) If the fragment group only has one fragment form, it will a will automatically be opened in the code viewer pane see below.

The structural representation of programs is abstract syntax trees (ASTs), that are stored on .ast files (or .astL files on the PC). The relations between the textual form and the structural form are handled in the following way: if the text file (e.g. the `.bet` file) is newer than the .ast file, theuser is asked whether the text file should be parsed automatically. If there are parse errors these can be corrected in a parse error editor.

5.2. Source Browser Windows

This short description describes the browser which is part of the Mjølner BETA System. This browser is used in a number of tools of the system (Sif, Freja, Frigg, Valhalla). This description only describes the browser functionalities, and not the application specific functionalities (such as the Sif structure editing functionalities).

We will here present the most important user interface components of the source browser. Tools integrated with the source browser will share these user interface components. These components are therefore described here, such that the other tools need not describe them.

We will describe the windows, menus and other interaction possibilities in the following sections.

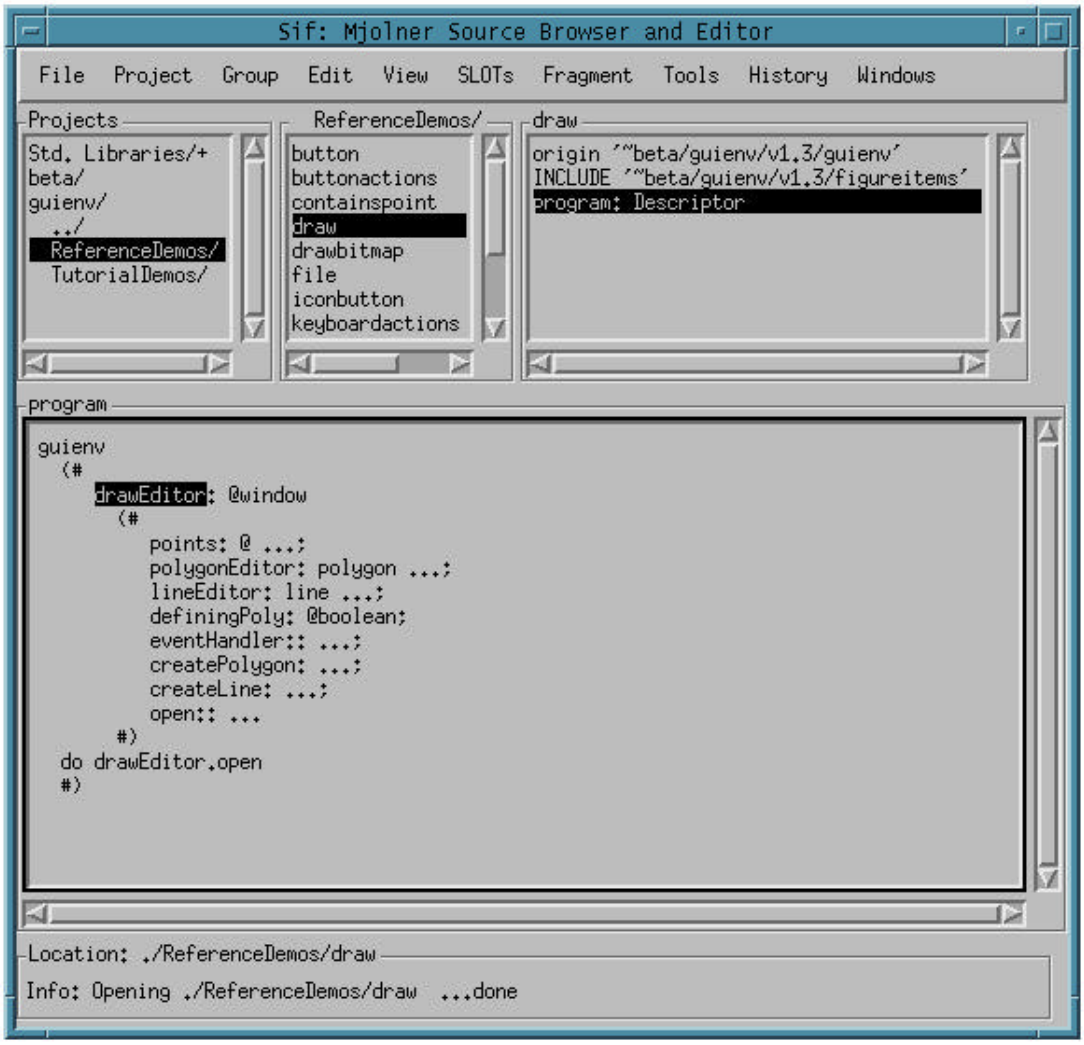
The source browser interface consists of a number of different window types:

- browser window
- workspace window
- separate code viewer window
- subviewer window
- parse error editor window
- two types of semantic errors windows
- help window

where the browser window is the main window, giving access to the main browsing facilities and to the other windows. These windows will be described in the following.

5.3. Browser Window

The browser window appears like:



The source browser window or has 4 important panes. Followed clock-wise from the upper left pane, the panes are described below.

Pane 1 is called the *project list pane*, it contains a list of projects.

Pane 2 contains a list of fragment groups in the project selected in the project list pane. It is called the *fgment group list pane* or just *group list pane*.

Pane 3 contains a list of properties and fragment forms of the fragment group selected in the group list pane. It is called a *fragment group viewer/editor* or just *group viewer/editor*.

Pane 4 displays the BETA code of the fragment form selected in the group viewer. It is called the *code viewer/editor*[3]. This code viewer will be seen in many different windows and the functionalities of the viewer is described later.

Below the code viewer/editor pane there are two info panes. *Location* displays path of the selected fragment group in the group list pane - or in the project list pane, if nothing is selected in the group list pane.

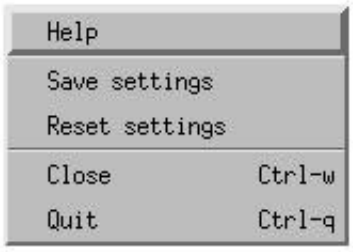
Info displays information, such as "no fragments in the selected project"' or what the source browser currently is doing.

Browsing can be done at basically 3 levels: the project level, the group level or at the code level.

5.3.1. Menus

The following sections describe the browser window menus offered by the source browser. The different tools that are integrated with the source browser might supply additional menus. Please refer to the individual tool manuals for information on which menus they supply for the browser window.

5.3.2. File Menu



Help

Displays a window, containing a series of manuals. The different manuals can be selected in the **Manuals**menu of this window.

Save settings

Enables you to save the current contents of the project list pane on a project-file $USER/.ymer.pjt. This project is then automatically loaded next time you invoke a source browser application - in this way you can save your configuration. It also saves/creates another file, $USER/.ymer.rc containing information on the locations of the BETA compiler, and other tools in the **Tools** menu (only in the stand-alone source browser application Ymer).

Reset settings

Enables you to clear the project list pane and reset it to the last settings (or the default settings, if no $USER/.ymer.[pjt|rc] files are found.

---

[3] The contents of the group viewer/editor can of course also be considered as part of the BETA code, but this is actually "code" written in the fragment lanuage.

Close

Closes the window (and terminates the application if this is the last browser window).

Quit

Closes all browser windows, and terminates the application or . In both **Close** and **Quit** source browser applications will examine if any files have been changed, and prompt for whether these files should be saved before closing/quitting.

5.3.3. Project Menu

New...

Is an unimplemented menu-item (that is, it is always disabled - will be implemented in a later version)

Open...

Opens a standard file dialog, and you can now select (a) a project-file (must have extension .pjt), (b) a directory, and (c) a .bet/.ast/.astL file. You will then in the project list pane find a project of the corresponding type A, B, or C (see Project Definitions). All other selections will be ignored.

Reload

Is used to reload the contents of this project (e.g. if this project represents a directory, and the contents of this directory has changed).

Save

Is yet another unimplemented menu-item (that is, it is always disabled - will be implemented in a later version). Is intended to enable the saving of changes to a project-file project.

Save As...

Is yet another unimplemented menu-item (that is, it is always disabled - will be implemented in a later version). Is intended to enable saving an existing project under a new name (and location).

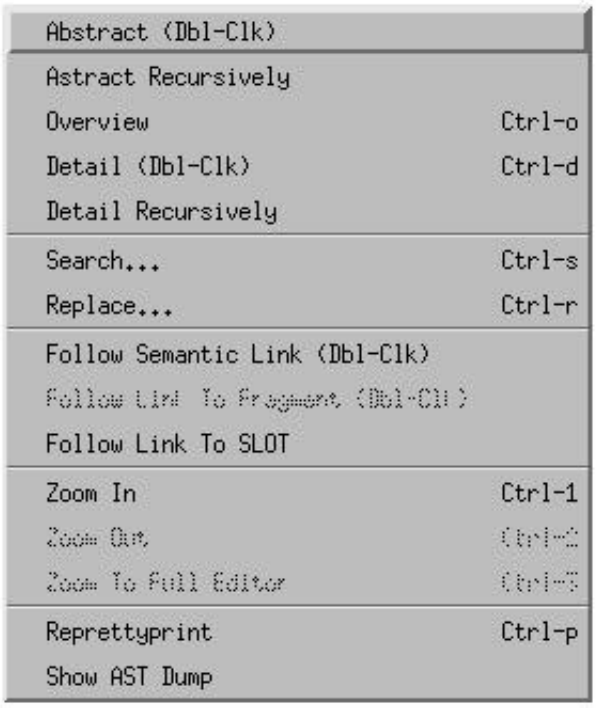
Rename...

Displays a dialog in which you can type a new name for the selected project.

Close

Closes the selected project, and removes it from the project list pane.

5.3.4. View Menu



One important quality of Sif is the possibility to present documents at different abstraction levels. Abstract presentation gives a good overview over a document by suppressing irrelevant details. Abstract presentation is provided by means of *contractions*. A contraction is a special presentation of an sub-AST, where only the root is presented. See also *Abstract Presentation* in *Source Browser Tutorial*. If the construct has an associated comment, the comment marker is included in the contraction. Browsing in the document can be done by detailing, i.e. opening the contractions selectively. This is done either by the command **Detail** or by double-clicking. Abstract presentation can be used to produce documentation such as a interface specifications by saving the document on textual form at different abstraction levels, using the **Save Abstract...** command.

Abstract

Contracts the current selection, i.e. presents the current selection as a contraction instead of the full "text". This can also be done by double-clicking.

Abstract Recursively

Traverses the current selection recursively and contracts those constructs that have a certain syntactic category. This command has only effect if abstraction levels have been specified in the grammar for the particular language. The grammar may specify a list of syntactic categories that automatically shall be contracted. For BETA the syntactic categories are `Descriptor`, `Attributes`, and `Imperatives, when opening the files in the viewer/editor`. For more information on specifying abstraction levels see [MIA91-14].

Overview

Like **Abstract Recursively** but the current selection is reestablished after the **Abstract Recursively** command, i.e. the path from the root to the current selection is detailed. This command is useful if you want to see the context of the current selection.

Detail

If the current selection is a contraction, this operation opens the contraction, i.e. presents the next level of text. A contraction can also be opened by double-

clicking on it. If the current selection is not a contraction any visible contraction in the current selection will be opened.

Detail Recursively

Recursively details the contractions in the current selection.

Search...

Makes it possible to search for a substring of a lexem, i.e. a name definition, a name application or a string in the current fragment form. A dialog pops up.

Replace...

Extends the search dialog with a replace text field.

Follow Semantic Link

During the checking phase a number of semantic links are inserted in the AST. These links are available to the user. Using this command on a name application the semantic link will be followed and the current selection is changed to the appropriate node in the current or another fragment form.

Follow Link to Fragment

Given a SLOT definition, a binding of the definition (i.e. a fragment form with the same name and type) can be searched for. This is done by selecting the SLOT definition and using this command. The editor will now search for a fragment form with the same name in the BODY and MDBODY hierarchy of the group that this fragment is part of. Notice that the binding of **Attributes** SLOTs may only be found in this way if they are bound in the BODY and MDBODY hierarchy.

Follow Link to SLOT

The editor searches after a SLOT definition with the same name as the current fragment form along the ORIGIN chain until it finds it or reaches the top, i.e. `betaenv`.

Zoom In

Changes the contents of the code viewer to the current selection, i.e. the root of the AST shown in the editor (the editor root) is changed to the sub-AST, that corresponds to the current selection.

Zoom Out

Extends the contents of the code viewer to show one level more of the structure, i.e. the editor root is changed to its father.

Zoom To Full Editor

Extends the contents of the code viewer to show the whole fragment form, i.e. the editor root is set to the whole AST of the fragment form.

Reprettyprint

Rebuilds the textual presentation of the AST and reestablishes the current selection. This command can be used if the contents of the code viewer need to be refreshed.

Show AST Dump

This command is useful for users of the metaprogramming system. It prints in the console window a dump of the sub-AST that corresponds to the current selection

Editing and contractions

Structure editing on parts of the document that contain contractions is done exactly as described earlier. You can for example cut, copy and paste constructs that contain contractions. When text editing constructs that contain contractions, some parser

technical information is included in the contractions before the text editor is activated. This information is necessary to avoid loosing contractions during text editing. Do not modify the contents of contractions during text editing. A contraction in text editing mode looks like:

Contraction in text editing mode

```
<<@4711: Descriptor>>
```

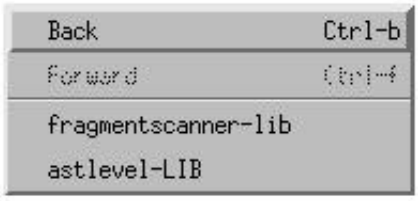
where `@4711` is an internal address of the sub-AST that is contracted, i.e. not presented, and `Descriptor` is the syntactic category of the contracted sub-AST.

Following links to other fragment forms

If the destination fragment form is different from the source fragment form the destination fragment form is shown in the same code viewer or in a separate code viewer, depending on the code viewer of the source fragment form. If the code viewer of the source fragment form is

- part of the browser window, the contents of the code viewer window is changed
- a separate code viewer, the contents of the code viewer window is changed
- a subviewer, a new subviewer is opened
- part of a work space window, a new code viewer is inserted in the work space window.

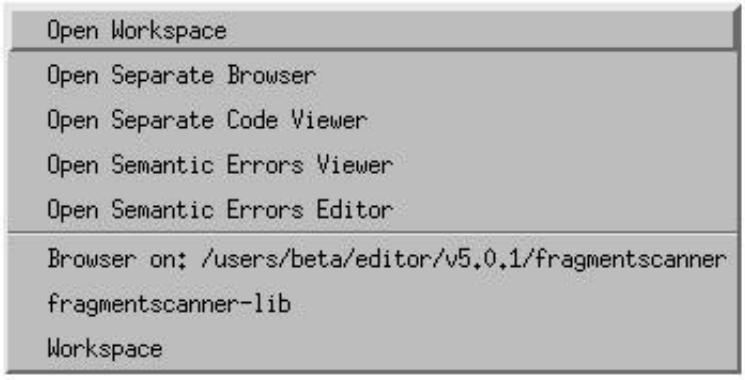
5.3.5. History menu



Back and **Forward** moves you along the path you have previously examined.

Furthermore, this menu contains one entry for each fragment form, you have displayed in the code viewer. This is used for speedy access to previously visited fragment forms.

5.3.6. Windows menu



Open Workspace

Opens a workspace window (see later more on this).

Open Separate Browser

Opens a new browser window, identical to this one. This window is controlled by the same source browser application instance. Makes it possible to browse two different "places" at once.

Open Separate Code Viewer

Opens a separate window just like the code viewer, displaying the currently selected fragment form in the group viewer.

Open Semantic Errors Viewer

Will open a semantic errors viewer on the selected group, if there are any semantic errors in that group. See later for more information on the semantic errors editor.

Open Semantic Errors Editor

Will open a semantic errors editor on the selected group, if there are any semantic errors in that group. See later for more information on the semantic errors editor.

The rest of the menu contains one entry for each window opened by the source browser. Selecting one of these items will bring the window to the front (and wriggle it)

5.3.7. Double-click

As already mentioned, you can display a fragment form in the code viewer by selecting it in the group viewer. But that's not all you can do in the group viewer.

Double-click can is used heavily in the sourcebrowser for browsing.

If you double-click on an ORIGIN, INCLUDE, BODY or MDBODY property, in the group viewer, the fragment group referred to in the property will be included in the group list pane and selected there.

If you double-click on a MAKE property, a new window appear with a Lidskjalv text editor for editing the MAKE file.

In the code viewer, double-click is also used for browsing. Double-clicking on the current selection will perform different actions. If the current selection is

- a name application, the semantic link will be followed (if available). If the definition is located in another fragment form the contents of the code viewer will be changed to show that fragment form.
- a SLOT definition, the fragment link will be followed (if available) and the destination fragment form will be shown in the same code viewer
- a comment mark (*), the full comment will be shown instead
- a expanded comment (* ... *), the comment mark will be shown instead
- a contraction (...), the contraction is opened, i.e. the next detail level is shown
- anything else the current selection will be contracted (...)

5.3.8. Shift-double-click

Shift-double-click is used for browsing like double-click, except that separate windows are opened.

In the project list pane, the group list pane and the group viewer, you can open a separate window by pressing <shift> down while double-clicking on an element in one of the lists.

In the project list pane, you will get a new separate browser window, with the subprojects of this project as top-level projects.

In the group list pane, you will get a new separate browser window with the [domain] and [extent] projects as top-level projects (See Project Definitions).

In the group viewer, you will get a separate code viewer on the selected fragment form. If the currently selected element in the group viewer is not a fragment form (i.e. it is a property), the effect is different. If the selected is an ORIGIN, INCLUDE; BODY or MDBODY property, a new separate browser window with the specified fragment group as top-level project. If the selected is a MAKE property, a text editor is opened, enabling textual editing of the Make file specified. All other properties are ignored. In the code viewer, shift double-click will do the same as double-click (except that the "result" will be presented in a separate window. Shift double-click will open different kind of separate windows depending on the current selection. If the current selection is:

- a name application, the semantic link will be followed (if available) and the destination fragment form will be shown in a separate code viewer
- a SLOT definition, the fragment link will be followed (if available) and the destination fragment form will be shown in a separate code viewer
- a comment, the comment will be shown in a separate window
- anything else a subviewer window with the current selection as its contents will be opened.

5.3.9. Search Facilities

There is a search facility in the 3 upper panes If you are having keyboard focus in one of these panes - usually by just clicking inside them, you can start search by pressing <Ctrl>i (^i for short - "i" is for "interactive search"). You can now type a string, and the source browser will immediately select the first item in the list with the typed string somewhere in the name.

You can get the next fragment form, matching the string, by again pressing ^i. Repeated ^i will repeatedly find the next match, starting from the top of the list again, if no more matches are found.

You can stop the search by pressing <cr>.

If you start the string by a "^" (caret), the string *must* be a prefix string, i.e. the name *must* start with the string.

5.3.10. Changing the Size of the inner Panes

If you position the cursor between any of the panes, a cross-cursor appears. By pressing one of the mouse button you can move the separation between the panes, making more space for the one (and naturally consequently less for the other).

5.3.11. Zooming in the different inner Panes

If you position the cursor at the visible borders of the panes, and double-click, you can make the particular pane zoom to take over major parts of the entire browser window.

If you zoom the info pane, you will be able to see all messages that have been written in the Info area.

5.3.12. Information in the Info Pane

Besides the already mentioned information in the Info area, you will occasionally see status information to the right of the pane. This status information is shown as small

boxes containing state information. There are three types of state information presently:

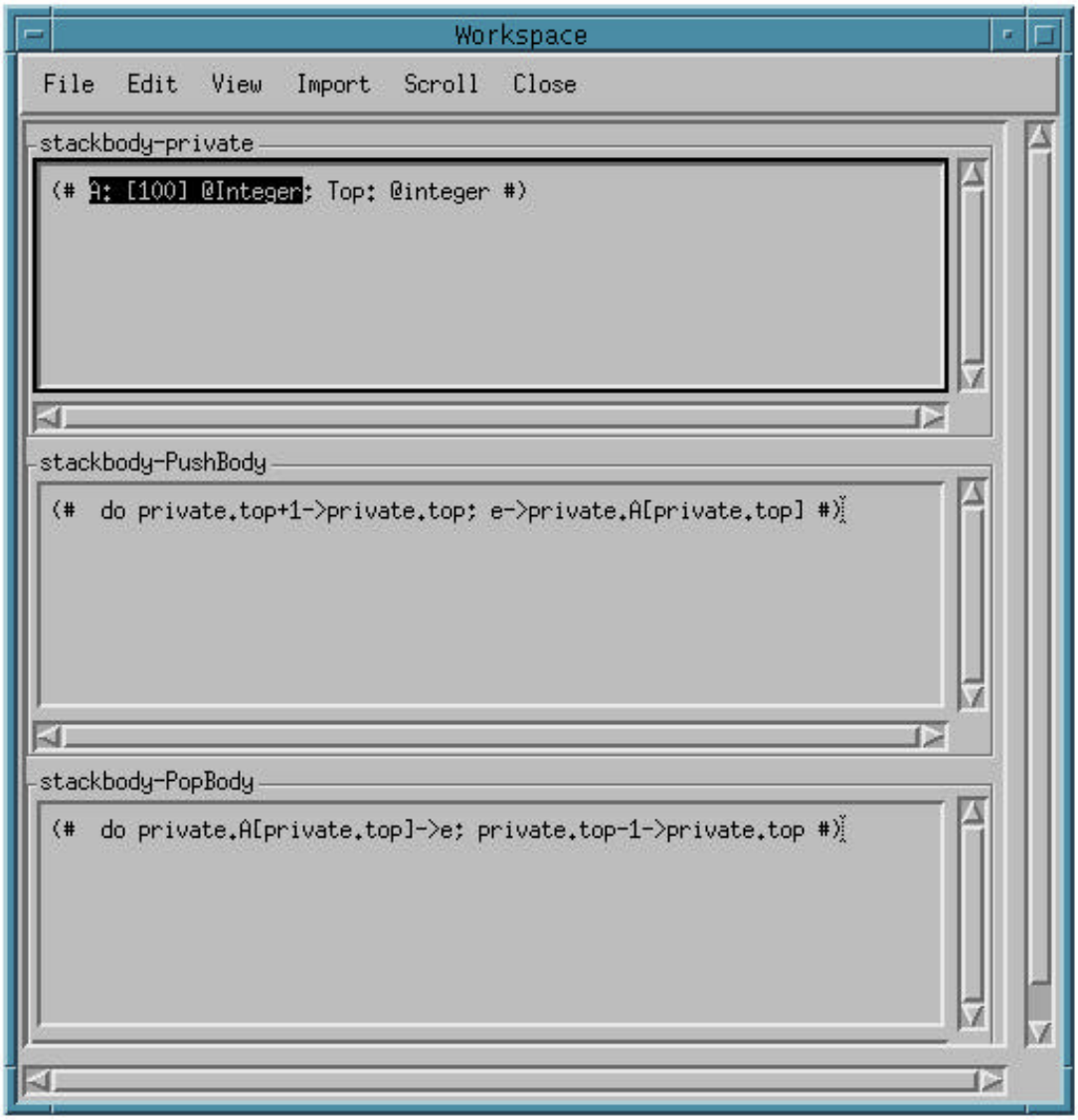
'%' indicating that the current fragment group is locked (cannot be edited)

'*' indicating that the current fragment group has been changed

c' indicating that the current fragment group has been checked but a semantic checker (usually the BETA compiler), and that semantic information therefore is available for semantic browsing.

5.4. Workspace Window

A workspace window (you can create as many as you want) is a window in which you can view (and possibly edit) fragment forms from different fragment groups (or the same fragment group) together. The workspace window looks like:



When a fragment form has been imported to the workspace through the Import menu, it will be made visible in a view identical to the code viewer in browser window. The workspace may contain as many such views as you wish. The outer scroller enables

you to scroll among the fragment forms in the workspace (try it - it is difficult to describe short).

As in the browser window, you can place the cursor between two views in the outer scroller, and thereby make the one smaller and the other bigger (and zoom the panes, as described in section 5.3.11). But in the workspace you can even more. If you press down <shift> while pressing the mouse button, you will *only* resize the view *above* the cursor - this is handy for making one view bigger without affecting the other views.

5.4.1. Menus

The above menus are those supplied by the source browser by default in a workspace window. Integrated tools may define additional workspace window menus - please refer to those manuals for details.

5.4.2. File Menu

The **File** menu just contains a **Close Ctrl-w** entry to enable closing the workspace.

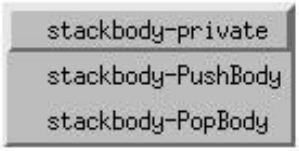
5.4.3. View Menu

The View menu is identical to the View menu in the browser window.

5.4.4. Import Menu

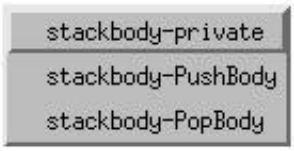
The **Import** menu will always contain at least one entry. This entry is the name of the name of the currently selected fragment group in the group list pane in the browser window. If selected, all fragment forms in that fragment group will be displayed in the workspace. Following this permanent entry, a variable number of entries are shown. Each entry corresponds to one fragment form in the currently selected fragment group in the group list pane in the browser window. If you select one of these entries, the corresponding fragment form will be displayed in the workspace. In the example above all but isEmptyBody and newBody is shown because the others are already imported.

5.4.5. Scroll Menu



The **Scroll** menu will contain one item for each fragment form in the workspace. If selected, the scroller will scroll such that the fragment form view is visible. In the example above these 3 fragment forms have been inserted.

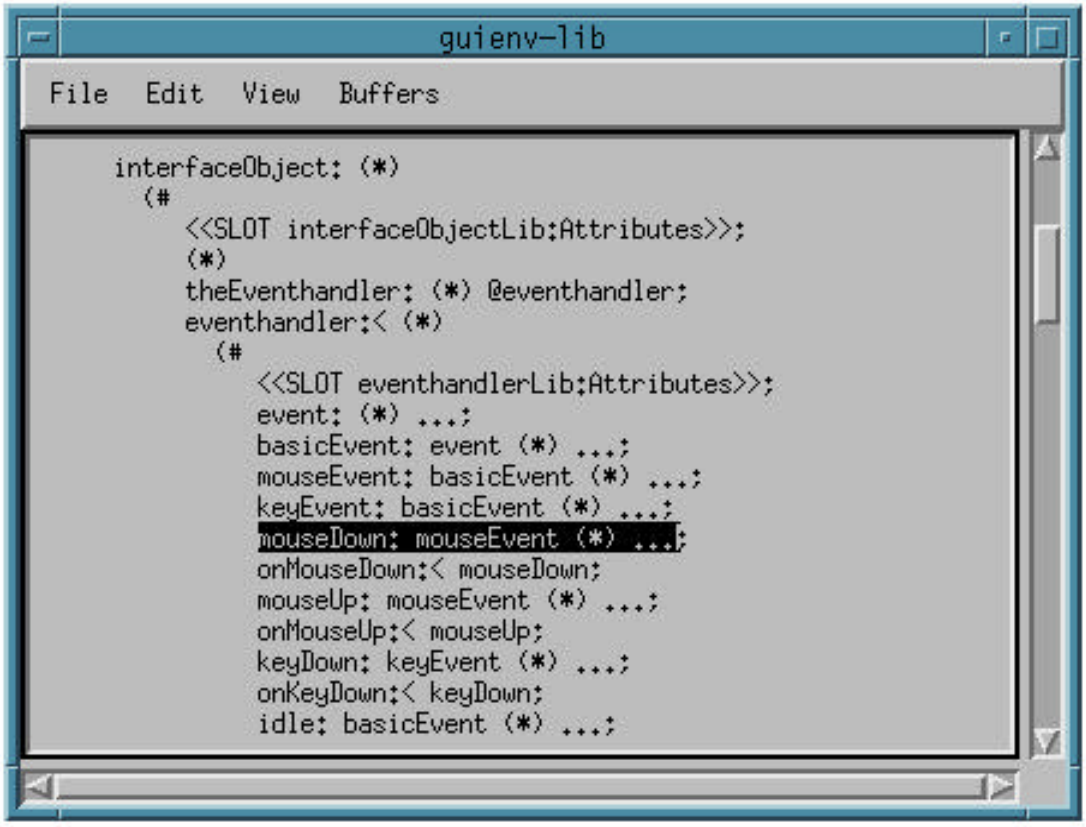
5.4.6. Close Menu



The **Close** menu will contain one item for each fragment form in the workspace. If selected, the chosen fragment form view is removed from the workspace. In the example above these 3 fragment forms have been inserted.

5.5. Separate Code Viewer Window

The separate code viewer window offers facilities for viewing (and possibly editing) a given fragment form, independently of any browser window. The separate code editor window looks like:



the code viewer is functionally identical to the code viewer in browser window.

5.5.1. Menus

The above menus are those supplied by the source browser by default in a separate code editor window. Integrated tools may define additional separate code editor window menus - please refer to those manuals for details.

5.5.2. File Menu

The **File** menu just contains a **Close Ctrl-w** entry to enable closing the separate code editor window.

5.5.3. View Menu

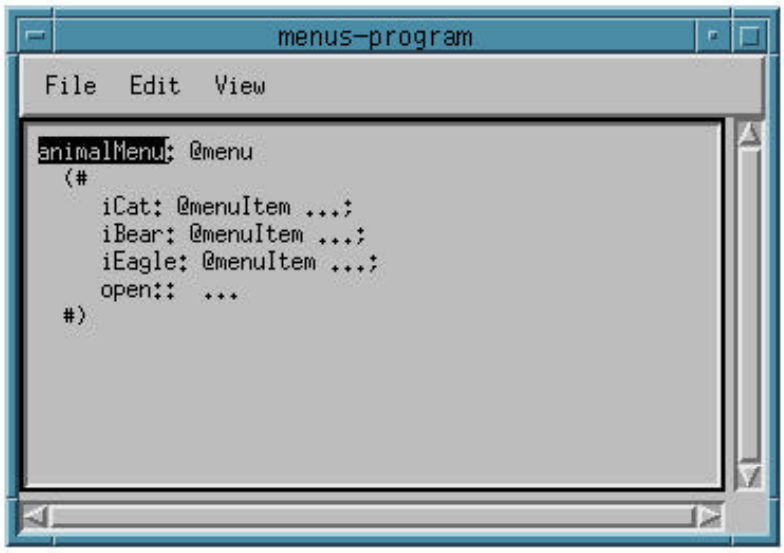
The **View** menu is identical to the **View** menu in the browser window.

5.5.4. Buffers Menu

The **Buffers** menu is similar to the variable part of the **History** menu in the browser window. It contains one entry for each fragment form, that has been shown in this separate code editor window.

5.6. Subviewer Window

The subviewer window offers facilities for viewing (or possibly editing) a given fragment form, independently of any other code editor window. The subviewer window looks like:



the code viewer is functionally identical to the code viewer in browser window.

There is no Buffer menu in a subviewer. If double-click in a subviewer implies that another fragment form must be shown, it is shown in another subviewer.

5.6.1. Menus

The above menus are those supplied by the source browser by default in a subviewer window. Integrated tools may define additional subviewer window menus - please refer to those manuals for details.

5.6.2. File Menu

The **File** menu just contains a **Close Ctrl-w** entry to enable closing the subviewer window.

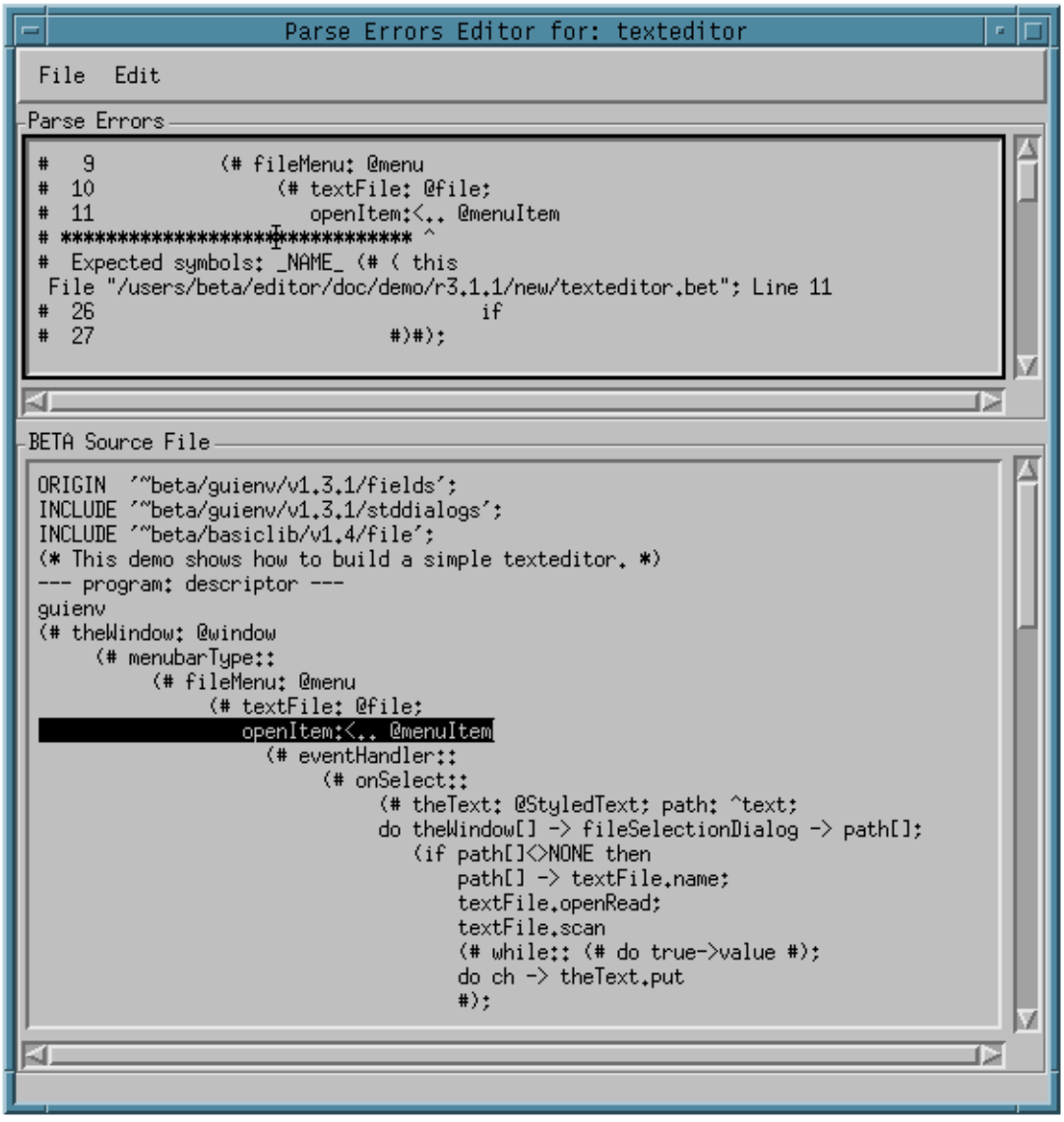
5.6.3. View Menu

The **View** menu is identical to the **View** menu in the browser window.

5.7. Parse Editor Window

If you select a fragment group that haven't been parsed (i.e. no .ast[L] file exist, or it's outdated), a prompt will ask you if this should be done. If you say yes, the source file will be parsed.

If the source file may contain parse errors, a parse errors editor window is automatically displayed. It looks the following:



The upper pane will display the list of parse errors, with the first one selected.

The lower pane will contain the source file, with the line with the first parse error selected.

If you press <cr> in the upper pane, the next parse error is selected and the corresponding line displayed in the lower pane.

As in the browser window, you can place the cursor between the two panes, and thereby make the one smaller and the other bigger.

5.7.1. Menus

The above menus are those supplied by the source browser by default in a parse editor window. Integrated tools may define additional parse editor window menus - please refer to those manuals for details.

5.7.2. File Menu

Save

Saves the edited file.

Revert

Reads in the original file again, thereby removing the changes that might have been made to the file.

Close Ctrl-w

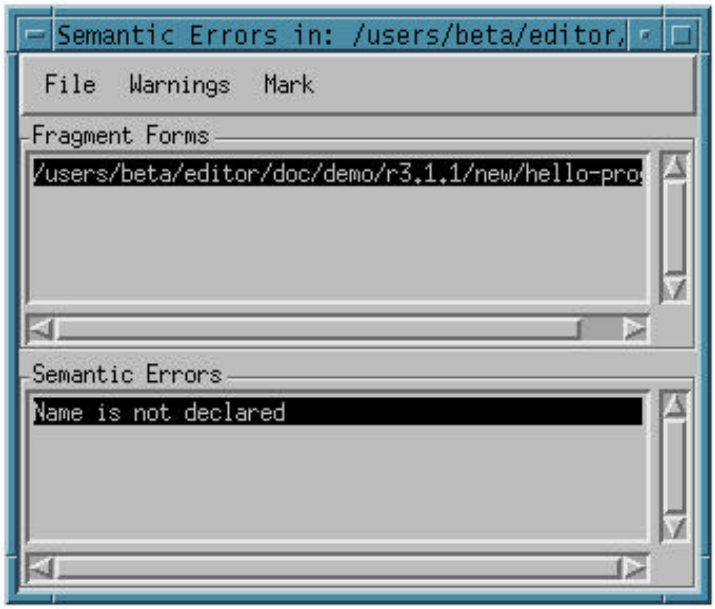
Closes the parse editor window. If changes have been made to the file, and not saved, a prompt will be displayed asking whether or not to save the changes.

5.7.3. Edit Menu

Edit contains the **Cut Ctrl-x**, **Copy Ctrl-c** and **Paste Ctrl-v** items with the obvious meanings (working on the contents of the lower pane).

5.8. Semantic Errors Viewer Window

The semantic errors viewer window is opened through the Windows menu in the browser window, or in those tools integrated with the BETA compiler, when a file is compiled and the compiler reports semantic errors. It appears like the following:



The upper pane contains a list of fragment forms containing semantic errors.

If you press <cr> in the fragment form list, the next fragment form is selected. If you press <cr> in the lower pane: the semantic error list, the next semantic error is selected.

When a fragment form is selected in the fragment form list, all semantic errors in that fragment form is listed in the semantic error list.

When a semantic error is selected in the semantic error list, the corresponding place in the source code is selected in the code viewer in the browser window:

As in the browser window, you can place the cursor between the two panes, and thereby make the one smaller and the other bigger.

5.8.1. Menus

The above menus are those supplied by the source browser by default in a semantic errors viewer window. Integrated tools may define additional semantic errors viewer window menus - please refer to those manuals for details.

5.8.2. File Menu

Reload Ctrl-r

Enables reloading the fragment group after a possible rechecking have been done.

Close Ctrl-w

Entry enables closing the semantic errors viewer window.

5.8.3. Warnings Menu

Include is a toggle. If toggle on, then warnings will be included in the list in the semantic error list. Otherwise, warnings are not shown.

5.8.4. Mark Menu

Mark as Corrected

Marks the currently selected element in the semantic error list as corrected (indicated by an asterisks "*" in front of the element). This is merely a help to the user during correction of the semantic errors, enabling him to mark those errors, that he thinks he has corrected.

Unmark

Removes such mark on the currently selected element in the semantic error list.

5.9. Semantic Errors Editor Window

The semantic errors editor window is opened through the Windows menu in the browser window. It offers the facility to view and edit semantic errors independently of any browser window. It is a semantic error viewer with a code editor adds.

When a semantic error is selected in the semantic error list, the corresponding place in the source code is selected in the code viewer in the lower part of the semantic errors editor window.

As in the browser window, you can place the cursor between the panes, and thereby make the one smaller and the other bigger.

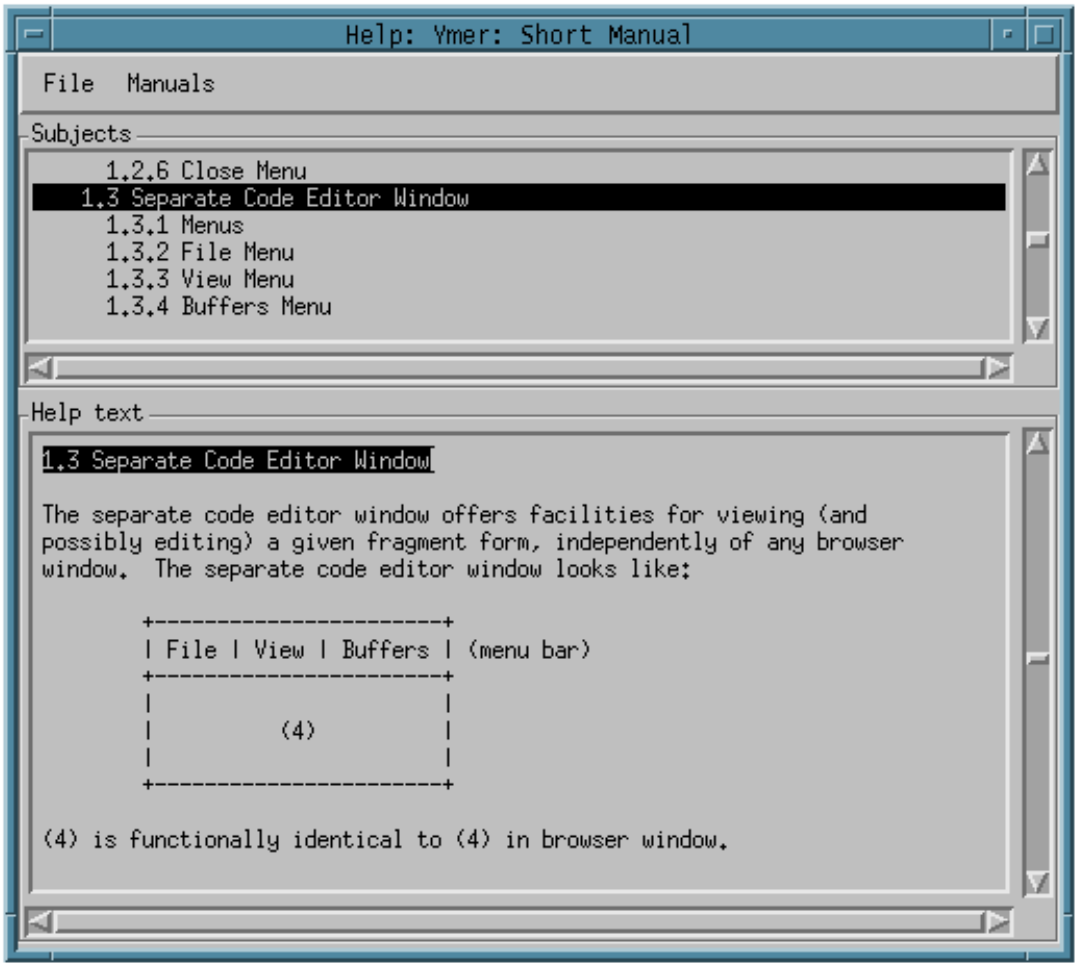
5.9.1. Menus

The menus of the semantic errors editor window include those one defined for semantic errors viewer. **View** and **Edit** menu may also be available (depending on the tool integrated with the source browser).

5.10. Help Window

The help window offers access to various help documents. The help documents are those supplied by the various tools integrated with the source browser.

The help window looks like:



The subjects pane contains a table of contents of the selected manual, and the Help text pane shows the text of the selected entry.

As in the browser window, you can place the cursor between the two panes, and thereby make the one smaller and the other bigger (and zoom the panes, as described in section 5.3.11).

5.10.1. File Menu

Load Ctrl-l entry enables you to load any HTML file into the help window.

Close Ctrl-w entry enables closing the help window.

5.10.2. Manuals Menu

The entries in the **Manuals** menu depend on the exact configuration, the source browser is used in. Here the entries supplied by Ymer is shown . Additional entries may be available, depending on the tools integrated with the source browser.

Selecting one of the entries, will load the corresponding manual into the help window panes.

5.10.3. HTML support

The help window offers additional support for reading HTML files in general. This is done by dynamically including the following 'manuals' in the **Manuals** menu:

1) If there is a file with the same name as the current group, but with extension `.html`
2) If the current project is:
   - a directory, and there are files with extension `.html` in that directory
   - a root/domain/extent project, and there exists a file with the same name as the project, but with extension `.html`
   - a project file, and there exists a file with the same name as the project file, but with extension `.html`.

All these .html files will become available in the **Manuals** menu of the help window.

5.11. Project definition

A project is either:

A. Defined in a 'project-file' (more later)

B. A directory

C. A fragment group

A project may contain subprojects. These are defined differently for the three project types:

A. A project may have defined subprojects in the project-file.

B. The sub-directories are considered subprojects.

C. Two subprojects are defined: [domain] and [extent]. These subprojects contain the fragment forms in the dependency-graph of the fragment group. [domain] contains all those in the domain (that is excluding BODY fragments), and [extent] contains all those in the extent (that is including BODY fragments). These two subprojects have in turn defined subprojects, namely the fragment groups in the domain (resp. the extent), sorted out into the libraries they are part of (e.g. basiclib, process, etc.).

5.11.1. Standard Projects

Assuming that you have not used source browser applications before, the project-list (1) will display three projects at start-up: "Std. Libraries", "~/" and "./".

"Std. Libraries" is a predefined project (defined in a project-file located in ~beta/configuration/r4.0/ymer.pjt), containing all the libraries in the standard MBS system (basiclib, containers, etc.). An example of this file is:

```
project basiclib '~beta/basiclib/v1.5 '
project containers '~beta/containers/v1.5'
project guienv '~beta/guienv/v1.4'
project bifrost '~beta/bifrost/v2.1'
project process '~beta/process/v1.5'
project persistentstore '~beta/persistentstore/v1.5'
project oodb '~beta/oodb/v2.1'
project distribution '~beta/distribution/v1.1'
project sysutils '~beta/sysutils/v1.5'
project unixlib '~beta/unixlib/v1.5'
project Xt '~beta/Xt/v1.9'
project mps '~beta/mps/v5.1'
project objectserver '~beta/objectserver/v2.4'
project contrib '~beta/contrib'
project demo '~beta/demo/r4.0'
```

"~/" is a directory project, referring to your home directory, and "./" is a directory project, referring to the current directory (i.e. the directory, where you invoked the source browser application from).

If you have invoked a source browser application with command line arguments, these will be shown in the projects list as well.

When a project/directory is selected in the project list pane, the group list pane will display all fragment files in that project. The source browser remembers the last selection in the group list pane and the group viewer, such that it will reestablish these selections next time you enters the same project or group.

When a group with one fragment form, it will automatically be selected.

5.11.2. Fold/unfold Projects

To see the subprojects/subdirectories of a project, double-click on the project (called unfold the project), then you will see the subprojects, properly indented, e.g.:

Std. Libraries

- basiclib
- containers
- bifrost
- (etc.)

Double-clicking again on the "Std. Libraries" will 'fold' the subprojects, only showing the "Std. Libraries" project, and not the subprojects.

5.11.3. Making a Fragment Group into a Project

You can make a fragment group into a project by double-clicking on it in the group list pane. It will then appear in the project list pane as a fragment group project. Double-clicking on it in the project list pane will calculate the dependency graph, and show the [domain] and [extent] subprojects.

5.11.4. Domain and Extent Projects

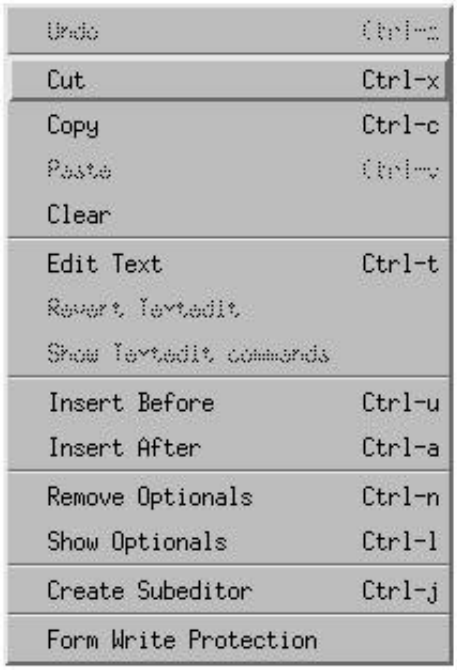
When selecting the [domain] or [extent] subprojects, the source browser will try to find possible conflicts in versions of libraries used, and report such conflicts in a pop-up window. The conflict analysis is based on the assumption, that versions are identified by subdirectory names of the form "vn.m", where 'n' is the major version number, and 'm' is the minor version number.

6. Editor Reference Manual

6.1. Code editor

The code editor provides structure editing on each fragment form.

6.1.1. Edit Menu



Undo

Initiating this command will undo the latest performed editing operation in the code editor. If the latest operation was **Undo**, the fragment will return to the state before the **Undo**.

Cut

The current selection is removed from the fragment form. If the current selection may be deleted (e.g. a statement or a declaration) it will disappear. If the current selection may not be deleted (e.g. the expression part of an if-statement) a nonterminal of appropriate syntactic category will be inserted instead of the current selection. The deleted fragment part will be put onto a clipboard and can later be pasted into the same or another fragment form.

Copy

The current selection is copied onto the clipboard, and can later be pasted into the same or another fragment.

Paste

Whenever the current selection and the structure on the clipboard are of exchangeable syntactic categories, **Paste** is enabled. **Paste** will replace the current selection with the structure currently on the clipboard. The content of the clipboard will not be changed.

Clear

Same as **Cut** but nothing is put on the clipboard.

Textedit

This command enters text editing mode on the current selection. See also *Text editing* in *Editor tutorial.* Alternatively the <space> key can be used.

Please note that text editing is only fully available if a parser for the supported language has been generated. If a parser is not available only lexems can be edited textually, i.e. name declarations, name applications, strings and constants. In that case, there is no check that the lexems are legal.

External Textedit

Like **Textedit** but here an external texteditor is used. This facility is only available in Unix versions. The environment variable EDITOR is recognized by Sif. When this command is invoked the current selection is inserted in the external editor. When the external editing has finished, by saving the results, the resulting text is inserted instead of the current selection. Parse errors are detected in the same way as in the internal texteditor.

E.g. emacs can be used by setting EDITOR to `/usr/local/bin/emacsclient` and starting emacs as a server in the following way: `emacs -f server-start`.

Parse Text

Exits text editing mode and parses the edited text according to the syntactic category of the current selection before entering text editing mode. See also *Text editing* in *Editor tutorial*.

Revert Text Editing

Exits text editing mode and reestablishes the situation before text editing was entered. Alternatively the <esc> key can be used.

Show Text Commands

When text editing, the text editing commands can be shown in a separate read-only window. Currently not enabled.

Insert Before/Insert After

Text editing commands

Whenever the current selection is one or more list elements, these entries will enable you to insert a nonterminal of same syntactic category as the list elements either before or after the current selection, respectively. If the list elements are on separate lines before/after means above/below, respectively. If the list elements are on the same line before/after means left/right respectively. An alternative to **Insert After** is the <cr> key.

Remove Optionals

Removes all nonterminals representing optional productions from the current selection.

Show Optionals

Inserts all nonterminals representing optional productions in the current selection.

Create Subeditor

A subeditor on the current selection is created, i.e. a code editor where the current selection is the root. Shift-double-click can also be used.

Form Write Protection

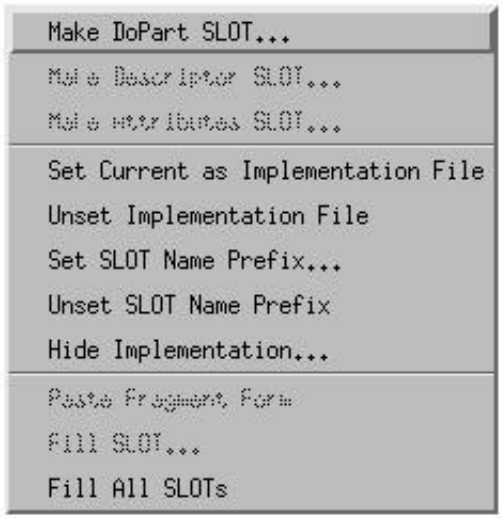
If checked it is not possible to modify the fragment form. Is automatically set if group write protection or global write protection is set.

6.1.2. Expand Menu

This menu is a pop-up menu, that is activated by the *Pop-up Menu Button* on the mouse (See basic user interface principles).

The content of the **Expand** menu is dependent on the current selection. If the current selection is a nonterminal, the **Expand** menu will contain an entry for each language construct that can replace this nonterminal. If the current selection is an optional or list nonterminal, the **Expand** menu will contain an **empty** entry as well. This entry enables you to remove that nonterminal.

6.1.3. SLOTs Menu



This menu is only present in the menu bar if the current fragment group is a BETA fragment group. The menu makes use of the *fragment clipboard* that is used as an intermediate store for fragment forms that are "moved around" between fragment groups. The fragment clipboard is also used in the **Fragment** menu.

Make DoPart SLOT...

Substitutes the current selection with a SLOT definition and the removed part of the document is inserted in an `DoPart` fragment form, that is stored in the fragment clipboard. The name of the SLOT is prompted for. The default name is the name of the enclosing pattern.

Make Descriptor SLOT...

Substitutes the current selection with a SLOT definition and the removed part of the document is inserted in an `Descriptor` fragment form, that is stored in the fragment clipboard. The name of the SLOT is prompted for. The default name is the name of the enclosing pattern.

Make Attributes SLOT...

Substitutes the current selection with a SLOT definition and the removed part of the document is inserted in an `Attributes` fragment form, that is stored in the fragment clipboard. The name of the SLOT is prompted for. The default name is the name of the enclosing pattern.

Set Current as Implementation File

The current fragment group selected in the browser is set to be the implementation file. If this file is set the **Make SLOT** commands and the **Hide Implementation...** command will automatically paste the created fragment forms that correspond to the SLOTs into the implementation file.

Unset Implementation File

The file set in **Set Current as Implementation File** is unset.

Set SLOT Name Prefix...

When one of the **Make SLOT** commands or the **Hide Implementation...** command is used the default name in the dialog for the SLOT name will be the text set by this command, followed by the name of the enclosing pattern (if any).

Unset SLOT Name Prefix

Clears the text set in **Set SLOT Name Prefix...**

Hide Implementation...

For each DoPart in the current selection in the code editor the user is asked whether it should be hidden into the implementation file by making it into a SLOT and moving the DoPart into a fragment form in the implementation file.

Paste Fragment Form

Currently disabled.

If the current selection is a SLOT definition with the same name and syntactic category as the fragment form on the clipboard this command substitutes the SLOT definition with the fragment form on the fragment clipboard.

Fill SLOT...

Currently disabled.

If the current selection is a SLOT definition, this command searches for a binding of the SLOT in the BODY and MDBODY hierarchy of the fragment group that this fragment form belongs to. If found, you are asked whether you want to substitute the SLOT definition with a copy of the found fragment form.

Fill All SLOTs...

Currently disabled.

Like **Fill SLOT...** but tries to fill all SLOTs in the fragment form. There is no undo on this operation.

6.2. Group Editor

The group editor is only relevant if the language in question is BETA. It is used for browsing or editing BETA programs. It is used to present and modify the structure of a group, i.e. the properties (ORIGIN, BODY, MDBODY, INCLUDE etc.) and fragments (Descriptor forms or Attributes forms).

6.2.1. Group Menu

New...
Makes it possible to create a new fragment group with one fragment form. You can choose between the currently available grammars. A dialog pops up.

New BETA Program...
Creates a minimal BETA program. You are prompted for the name and location of the new fragment group.

New BETA Library...
Creates a minimal BETA library. You are prompted for the name and location of the new fragment group.

Close
Closes the current fragment group. If changes have been made to the fragment group, you are asked whether the changes should be saved.

Save
If changes have been made to the fragment group, it is saved. A backup of the old text file (if existent) is taken on e.g. `foo.bet~` and a new text file is generated e.g. foo.bet.

Save As...
Saves the current fragment group under a another name. You are prompted for the name and location of the new fragment group. The editor continues with the new fragment group.

Save All...
Calls **Save** on all open group editors.

Save Abstract...
Saves the current fragment group on textual form at the current abstraction level. You are prompted for the name and location of the text file.

Recover
Recovers to the latest auto save file (.ast# or .astL# on PC). This command is enabled if the .ast# exists and is newer than the .ast file.

Revert
Reverts any changes (since the latest save) made to the current fragment group.

Group Write Protection
If checked, it is not possible to modify the fragment group. Group write protection is set automatically if there is no write permission on the fragment group file or if the global write protection is set.

Global Write Protection
If checked the whole editor is in read-only mode.

6.2.2. Fragment Menu

The cut, copy and paste operations are using a so-called *fragment clipboard*.

Most commands in this menu are performed according to the currently selected item in the group editor. The selected item can be a property, like ORIGIN, INCLUDE and BODY or a fragment form,.

Undo
Undo the last executed command in the group editor. Not implemented yet.

Cut
The selected fragment form is deleted. A copy of the fragment form is put on the fragment clipboard.

Copy
A copy of the selected fragment form is put on the fragment clipboard.

Paste Before
The fragment form on the fragment clipboard is inserted in the fragment group before the selected item.

Paste At End

The fragment form on the fragment clipboard is appended to the list of fragment forms in the fragment group. It can also be pasted into a SLOT definition in a code editor (See the **Paste Fragment Form** command in the **SLOTs** menu).

New Descriptor Fragment Form...

A new Descriptor fragment form is created and inserted before the selected item. You are prompted for the name of the fragment form.

New Attributes Fragment Form...

A new Attributes fragment form is created and inserted before the selected item. You are prompted for the name of the fragment form.

New DoPart Fragment Form...

A new Dopart fragment form is created and inserted before the selected item. You are prompted for the name of the fragment form.

Edit Properties

The properties of this fragment group can be edited in a separate code editor that supports the “property specification language”. See *Property Editor.*

Edit Name...

The name of the selected fragment form item can be modified.

6.3. Property Editor

The property editor is able to edit the properties that can be attached to a fragment group, i.e. properties like ORIGIN, INCLUDE; BODY, MDBODY, OBJFILE, LIBFILE, BETARUN etc. In order to edit the properties of a fragment group, a structure editor supporting a grammar for the “property language” is used. The property editor is activated in the **Fragment** menu by means of the **Edit Properties** command.

6.4. Tools Menu

This menu is only present in the menu bar, if the language supported in the active code editor is BETA.

Check Current

The current fragment group in the browser is checked. Output from the checking is written in the Info pane. Possible semantic errors are shown in semantic error viewer and the structure corresponding to the firs semantic error in the list is selected.

Recheck

The fragment group that was previously checked or compiled is checked.

Compile Current

The current fragment group in the browser is compiled, i.e. checking, code generation, assembling (if binary code is not directly generated on the actual platform) and linking is performed. Output is written in the Info pane. Possible semantic errors are shown in semantic error viewer and the structure corresponding to the firs semantic error in the list is selected.

Recompile

The fragment group that was previously checked or compiled is compiled.

Execute Current

If an executable exists that has the same name as the current fragment group it is executed.

Reexecute

The application that was previously executed is started again.

Debug

Activates the debugger Valhalla on the executable of the current fragment group if it exists..

Semantic Errors...

Opens the semantic error viewer.

Show Diagram

The diagram that corresponds to the current selection in the code editor of the browser window is shown. in Freja, the CASE tool. Is only enabled if the actual version of Sif is integrated with Freja.

6.5. Backup

Backup group file: .ast~ (or .astL~ on PC)

When a fragment group is opened in Sif a backup of, say, `foo.ast` is taken on `foo.ast~`.

Auto-save group file: .ast# (or .astL# on PC)

When a certain number of structure editing operations has been performed, a copy of the fragment group, say, `foo.ast` is stored on `foo.ast#` Text editing including parsing is considered as 1 structure editing operation. When a fragment group is being opened in the editor and an auto-save group file exists that is newer than the .ast file, you are asked whether that file should be used instead.

Backup of text file: .bet~

When a fragment group is saved either by the user or by the compiler, a copy of the corresponding text file, say, `foo.bet` (if present) is taken on `foo.bet~`

6.6. Opening Grammar Files

Sif can also be used to edit or inspect the available grammars and corresponding prettyprint specifications. The grammar files of the Mjølner BETA System are normally located in `~beta/grammars`.

File naming convention

There is a naming convention for the grammar files and the prettyprint specification files. For BETA the grammar file is called `beta-meta.gram` and the corresponding group file is called `beta-meta.ast`. The prettyprint specification file for BETA is called `beta-pretty.pgram` and the corresponding group file is called `beta-pretty.ast`. To open the grammar file for BETA the entry `beta-meta` must be selected. To open the prettyprint specification file for BETA the entry `beta-pretty` must be selected. The grammar directory contains also a group file called `beta.ast` but this is for internal use only.

For more information on grammars and prettyprint specifications etc. see [MIA91-14].

7. Bibliography

[MIA91-14] The Metaprogramming System - Reference Manual

[MIA90-2] BETA Compiler - Reference Manual

[MIA90-8] Basic Libraries

Index