

Users Guide and Reference

- **C-Forge Overview**

- **C-Forge Concepts**
 - [Project](#)
 - [Folders](#)
 - [Tools & Tools Options](#)
 - [Hidden Targets](#)
 - [Conversions](#)
 - [Implicit Dependencies](#)
 - [Directory Layout](#)
 - [Importing of External Project](#)
 - [Version Control](#)

- **Projects Manager**
 - [Projects Groups](#)
 - [C-Forge Options](#)
 - [New Project Dialog](#)
 - [Import Project Dialog](#)
 - [Run a Program Dialog](#)
 - [Process Status](#)
 - [Tools Options](#)

- **Working With Projects**
 - [Creating a new project](#)
 - [Importing a Project](#)
 - [Creating a Multiuser Project](#)
 - [Opening an existing project](#)
 - [Adding files to the project](#)
 - [Adding folders](#)
 - [Deleting files](#)
 - [Editing files](#)
 - [Debugging files](#)

- [Making targets](#)
- [Running Targets](#)

• Revision Control System

- [Including Files into RCS](#)
- [Excluding Files from RCS](#)
- [Fetching from the Repository](#)
- [Revision Control Options](#)
- [Checking-Out](#)
- [Checking-in a New Revision](#)
- [RC log-window](#)
- [Version Tool](#)

• Working With External Projects

- [Overview](#)
- [Methods of External Project Using](#)
- [Revision Control](#)
- [Loading Tools options of the Imported Makefile](#)
- [Importing Projects Using Working Directory](#)

• Project Desktop

- [Stable and Working Directories](#)
- [Project Options](#)
- [New Folder Dialog](#)
- [Adding Files to Project Dialogs](#)
- [Templates](#)
- [Deleting Files Dialog](#)
- [Find Node Dialog](#)
- [Node Info](#)
- [Project Log Window](#)
- [Environment](#)
- [Project Conversions](#)
- [Project Macros](#)
- [Project Info](#)
- [Tools Binding](#)
- [Search/Replace Tool](#)
- [Diff/Merge Tool](#)

- [Symbol Navigator](#)

• **SMED (Integrated Smart Editor)**

- [SMED Features](#)
 - [Client/Server Design](#)
 - [Saving Files](#)
 - [Context Highlighting](#)
 - [Automatic Indentation](#)
 - [Bookmark System](#)
 - [Finding text](#)
 - [Replacing text](#)
 - [Working with Symbol Navigator](#)
 - [Drag'n'Scroll](#)
 - [Block Selection](#)
 - [Integration with the IDE](#)
 - [Views](#)
 - [Function Panel](#)
 - [Macro Record/Playback](#)
- [Editor Window Description](#)
- [Search and Replace Dialogs](#)
- [Custom Resources](#)
- [Revert to File Dialog](#)
- [Editor Macros](#)

• **Miscellaneous**

- [Drag and Drop Overview](#)
- [Integrated Help/Man Browser](#)
- [Frequently Used Interface Components](#)
- [File Selection Dialog](#)
- [Path Selection Dialog](#)
- [Mouse Reference](#)
- [GNU Make Automatic Variables](#)
- [GNU Regular Expressions](#)
- [Command-line parameters](#)

Overview



Code Forge is a multi-user integrated development environment that provides full project management and a complete edit/compile/debug cycle support delivering a clear and intuitive user interface.

Features

- [Project Manager](#) provides a visual representation of available projects.
- [A visual representation](#) of the project structure and component status using a dependency tree with virtual folders and a separate desktop area showing work in progress.
- [Stable and Work environments](#): Users work in their private environments. Changes affect other users only when modules are returned into the stable environment.
- A fully configurable [integrated editor](#) with context highlighting, auto indentation, multi-level undo and bookmarks.
- Support for many languages.
- Ability to [import](#) external makefiles.
- Ability to create [multi-user projects](#)
- User ability to [integrate and configure](#) external tools such as compilers, parsers and debuggers.
- Integrated [Symbol Navigator](#) with cross-referencing and parsing.
- [Revision Control System](#): primary revision control functions performed automatically by the environment (Integrated Revision Control Tool allows even finer control).
- Intuitive drag'n'drop functionality enabled throughout the [environment](#).
- A [Diff Tool](#) is provided for visual presentation of the differences between files.
- [Grep Tool](#) searches the project or custom files for lines containing a match to the given pattern.
- Intelligent [Log Window](#) that displays project build output and jumps to corresponding error lines in the editor.
- Integrated hypertext [help facility](#) with MAN interface.
- Full-featured [Path Selection](#) and [File Selection Tool](#) (Explorer-like facility that provides a directory view in *Brief Info* or *Full Info* modes and allows filtering).

C-Forge concepts

Project

A project consists of a collection of targets and their dependencies. A target is the end result of actions performed on its dependencies. It can be anything from a binary executable to a library, to a Perl or shell Script. Each target has a set of actions associated with it. These are shell scripts that perform the activities necessary to Build, Run and Debug the target. Each target is assigned a set of default actions when it is created, based on its type. A user may consequently [modify these actions](#) to better suit their development needs.

Dependencies are components that are used to build a target. Depending on its type, a target may or may not have dependencies. For example, a target of type "C++ Executable" will have C++ source files as its dependencies. Dependencies may also be other targets, such as a library. When building a target, builds are recursively executed for all its dependencies.

Lets say we are designing a project that consists of two executable files each using the same library. We [create three targets](#) - described below in the following format: <target name>:<dependencies>.

```
program1:
```

```
program2:
```

```
library.a:
```

We define their dependencies by [adding source files](#):

```
program1:    prog1.c
```

```
program2:    prog2.c
```

```
library.a:   functionA.c  
            functionB.c
```

Here C-Forge allows you to skip a step, by automatically providing hidden dependency definitions. Behind the scenes the following targets and dependencies are created:

```
program1:    prog1.o
```

```
prog1.o:     prog1.c
```

```

program2:    prog2.o

library.a:   functionA.o
             functionB.o

functionA.o: functionA.c

functionB.o: functionB.c

```

You can in fact force this to happen by creating these intermediate targets yourself, however in most situations such level of detail can be avoided.

Our goal is to have the two executables link to the same library. Thus we [place our library](#) into the dependency list of each executable target:

```

program1:    prog1.c library.a

program2:    prog2.c library.a

library.a:   functionA.c
             functionB.c

```

In essence the rule for target program1 says: build program1 if target1.c needs to be built or library.a needs to be built.

Thus the following checks and actions would be executed to build program1:

1. if file prog1.o doesn't exist or its timestamp is earlier than the timestamp of file prog1.c then a build action is executed for target prog1.o.
2. a similar check is performed for target library.a and recursively doing a build action for its targets functionA.o and functionB.o.
3. if the file program1 doesn't exist or its timestamp is earlier than the timestamps of files prog1.o or library.a, then a build action is executed for target program1.

Each step depends on the successful completion of the step before it. If any of the actions fail - the build is interrupted.

Instead of building program1 and program2 separately it would be more convenient to create one more target to build both program1 and program2. Thus your final project would look like this:

```

All:        program1
           program2

```

```
program1:  prog1.c  
          library.a  
  
program2:  prog2.c  
          library.a  
  
library.a: functionA.c  
          functionB.c
```

As you have probably noticed - this is very much like the structure of a Makefile. C-Forge is, in fact, a Visual Makefile builder that allows you to construct and rearrange make rules through intuitive drag and drop operations.

Folders

[Virtual folders](#) are used for logically grouping targets in a project. As a project gets bigger and the number of targets increases, the use of folders alleviates clutter. In addition to simplifying project navigation, virtual folders can be used to apply similar actions to all targets inside them. For example in order to build all the targets within a folder the user can execute a build action on the folder itself. Virtual folders may be created in the root project folder or inside other virtual folders. Virtual folders are not a physical construct - only references to targets are created when targets are copied into it. The physical layout of the project is not actually changed on disk. A virtual folder may not be deleted until it is empty. Several virtual folders can contain references to the same target.

[Physical folders](#) are used for accessing files and directories directly from the project. Similar actions such as editing or building (whether physical folder includes a Makefile) can be applied to all files included into physical folder. Physical folder directories can be also subjected to version control if necessary.

Physical folders may be created in the root project folder, inside virtual folders or inside other physical folders.

Note: Physical folder can not include virtual folder!

Tools & Tools Options

The IDE assigns a build action to a target during its creation. The [Tools Options](#) mechanism allows convenient configuration of build options for the tools (such as a C compiler and linker) involved in the process. Each tool has its own Macro that is used in writing the build action scripts.

Hidden Targets

As previously mentioned, when adding a source file to the dependency list of a target, an intermediate (hidden) target is created. This target is used to build the object file from the source file. This intermediate target is actually added to the dependency list. When creating an intermediate target the IDE assigns it a build action script. The rules used to assign the build actions are called Conversions and are based on the source file type.

Note: The words hidden and intermediate are used interchangeably in reference to this type of target.

Conversions

[Conversions](#) are a collection of rules used for the creation of hidden targets from source files being added to targets. The selection of a Conversion is based on the file extension of the source file. The Conversion rule contains information on the type of intermediate file that will be produced (for example filename.o from filename.cpp) and the tool used to build (produce) the intermediate target. Many popular conversions have already been defined. In addition to this, the user can define their own Conversions.

Implicit Deps

In addition to [include file dependencies](#) defined explicitly by the user - a project may utilize includes external to the project. Defining these dependencies "by hand" can be tedious and in all probability unnecessary since, most likely, these includes are either system or library files and will probably never change. C-Forge does, however, provide a mechanism for automatically locating and adding such files as dependencies. A list of include files added as Implicit Dependencies can be viewed in a specialized folder in a C-Forge project.

Directory Layout

Targets have one more attribute - directory layout. Targets can be assigned Source, Include and Object directories where their source and include dependencies reside and object files are created.

C-Forge also supports the so-called "flexible" layout, when different object files are created on the base of the same source files while building the target. In order to do that you need to assign different Object directories for object files of different targets while [new target creation](#).

Another way of layout managing is to create physical folder and to link files from it to the Dependency Tree. While building target, object files will be created at the Object catalog assigned for object files.

In addition to this, C-Forge supports two types of project layouts: simple and Stable/Working. Project layout is specified at creation time. Simple layout creates a single (stable) directory tree on disk where the project files reside. When a project is created using a Stable/Working layout, in addition to the stable directory tree, a working directory tree is created in the user's home directory. Only the files that the user is working on are copied into the Work directory. The intermediate targets resulting from builds of the files being worked on are also created in the Work directory. This layout, in affect creates two projects: stable and working. The benefit of this layout is that at any given time the user has access to both the stable project and the modified project. Changes can be applied and tested in the working directory without affecting the stable directory and/or other users. This is especially useful in a [multi-user project](#). In addition to this the user can always revert to the stable version of the project by simply throwing out all the changes in the working area. When a project is created with a simple layout (ie without the working directory) all modifications are applied directly to the stable directory tree.

Importing of External Project

C-Forge supports several methods of [importing external projects](#) so that one can choose which of them suits him better.

To use any of those methods you have to invoke [Import Project dialog](#).

Version Control

An important element of software development is [version control](#). The basic idea is the ability to work with several versions of the same file. This paradigm is useful in keeping track of changes applied to a file or several files, "rolling back" to previous versions and especially as a method of synchronization for [multi-user projects](#). The three most important concepts of version control are:

1. A central repository where the "current" version of a file resides in read-only form, as well as a database of revision changes applied to the file since its creation.
2. Check-Out - a user performs a check-out on the file or catalog to fetch the most current file(s) version into a directory where it can be modified. In certain flavors of version control the repository for the file in question is locked to other users until a check-in is performed.
3. Check-In - a user performs a check-in of the file(s) to apply the changes to the repository and in so doing create a new revision of the file(s).

The current version of C-Forge supports automated revision control through RCS (Walter F. Tichy & Paul Eggert), CVS, SCCS, PRCS and Rerforce. RCS and SCCS work with the separate file on the local disk. CVS, PRCS and Perforce are able to trace changes within the whole project and allow to work with remote repositories.

By default any editable file created/imported into C-Forge will be placed under revision control. (If the catalog where the file in question is created/imported is not subjected to the revision control, the file will not be placed under revision control either.) The user when creating the project should specify revision control type.

Depending on the type of revision control, specified for the file or project, in the directory where the new file is to be created the necessary actions (special records, or subdirectories are created) are performed while the new file is being created/imported.

Check-out is performed automatically by C-Forge when a file is moved into the working directory.

Check-in is performed automatically when a file is moved from the working directory back into the stable project.

Cancellation of changes applied to the files in the working directory will result in locks being removed for all files involved in the cancellation.

Last modified: Wednesday, 04-Jul-2001 10:13:25 EDT

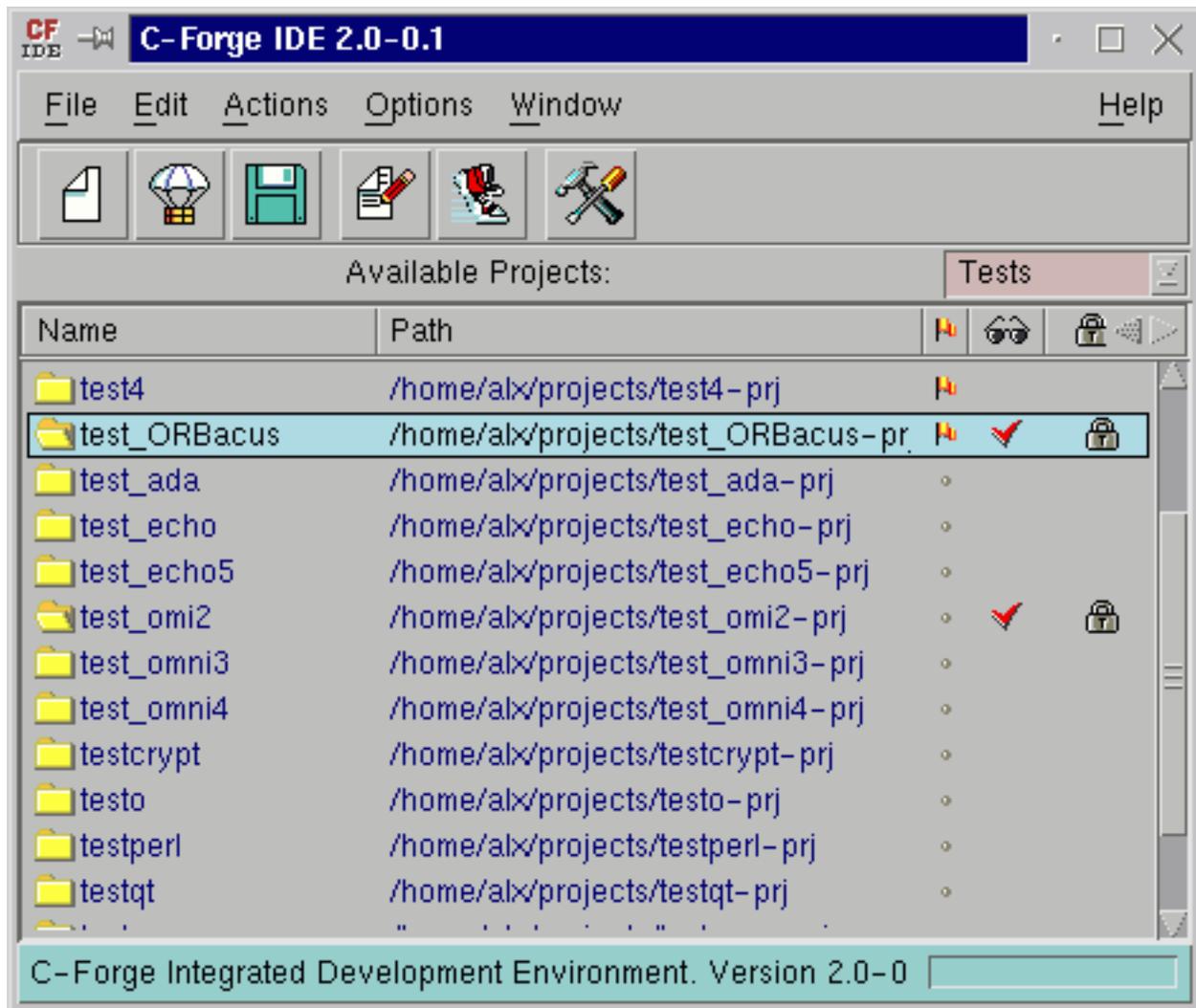
The Project Manager

- [Overview](#)
- [Project List](#)
- [Menu](#)
- [Tool Bar](#)
- [Status Bar](#)

Overview

The **Project Manager** is the main window of C-Forge.

The **Project Manager** provides access to all projects known to C-Forge. It controls the creation of new projects and the importing of external makefiles.



Project List

Project list displays the following information:

- Project Name with an open or closed folder icon depending on the state of the project.
- The project location.
- Read and Write permissions.
- The number of users with permission to work on the project.
- The number of users currently working on the project.
- An icon of a lock is shown when the project is locked.
- A red mark is shown when the project is not hidden.
- A flag icon allows you to mark often used projects.

The title bar of the Project List is the standard element of [interface](#), that allows changing the order and size of columns and sorting the elements.

Double-clicking on a project or selecting it and pressing the **Enter** key opens the project. Right-clicking on a project brings up a menu of project actions.

Menu

File

- **New project** - opens the [New Project](#) dialog that allows creating a new project.
- **Open project** - opens the selected project or raises the window of an opened project.
- **Close project** - closes the selected project.
- **Save project** - writes the changes made in the selected project to a disk.
- **Import project** - opens the [Import Project](#) dialog that allows importing an external project.
- **Delete project** - deletes the selected project, the [Delete Files List](#) dialog will be opened; this command is available only for the closed projects.
- **Project Info** - opens the [Project Information](#) dialog.
- **Hide project** - hides the selected project, i.e., hides all the windows of the project and minimizes the [Project Desktop](#). For the hidden project this command is replaced by **Show project** command that restores all the hidden windows of the project.
- **Exit** - exits C-Forge.

Edit

- **New group** - opens the [New group](#) dialog that allows creating a new projects group.

- **Delete group** - [deletes](#) the current projects group.
- **Move project(s)** - [moves](#) selected projects from current projects group to another group.

Action

- **Run** - opens the [Run a Program](#) dialog to execute an external program.
- **Edit** - opens the [File Selection](#) dialog to select a file to be edited by build-in editor ([SMED](#)).
- **Diff Tool** - launches the [Difference Tool](#).
- **Process Status** - opens the [Process Status](#) window

Options

- **C-Forge Options** - opens [C-Forge Options](#) dialog.
- **Default Tools Options** - specifies the default [tools options](#).
- **Default Project Options** - specifies [project options](#) that will be used by default in all the new projects.
- **Environment** - opens [Environment](#) window to edit the environment variables.

Help menu shows the appropriate help pages.

Tool Bar

The project manager tool bar includes the following buttons:



- **New project** - opens the [New Project](#) dialog that allows creating a new project.
- **Import project** - opens the [Import Project](#) dialog that allows creating a new project.
- **Save project** - writes the changes made in the selected project to a disk.
- **Edit File** - opens [File Selection](#) dialog to select a file to be edited.
- **Run process** - opens [Run a Program](#) dialog to execute an external program.
- **Tools Options** - opens the [Tools Options](#) dialog to specify the options for new projects.

The tool bar can be hidden using the **Show Toolbar** option in the [C-Forge Options](#) dialog.

Status Bar

The status bar shows tips for the selected menu items and Tool Bar buttons. The progress bar on the status bar indicates the progress of lengthy operations.

The status bar can be hidden using the **Show Statusbar** option in the [C-Forge Options](#) dialog.

Last modified: Friday, 15-Jun-2001 06:55:30 EDT

Projects Groups

- [Overview](#)
- [New group](#)
- [Delete group](#)
- [Move projects](#)

Overview

To sort many projects Code Forge has **Projects Groups** tool that allows to create new, delete existing project groups and move projects from one project group to another.

Code Forge always has projects group **All** which contains all projects.

New Group

To create new projects group select "**Edit->New group**" item of the [Project Manager](#) menu, which opens **New group** dialog:



If you previously selected one or more projects and "**Move selected project(s) to new group**" check-button is selected, those selected projects will be moved into new projects group.

Delete Group

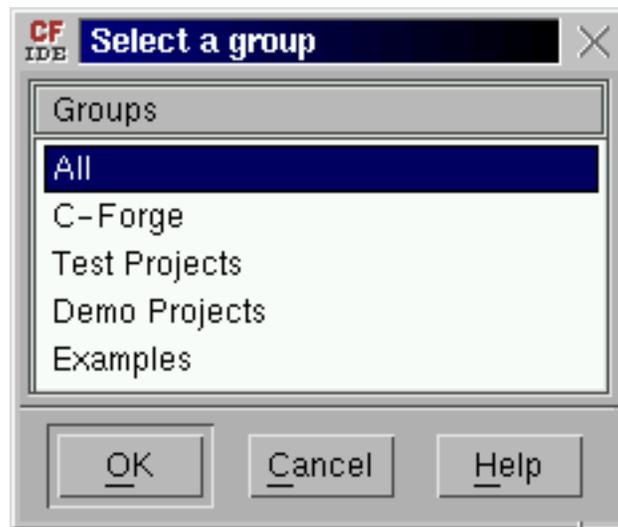
You can delete only currently selected projects group.

To delete current projects group select "**Edit->Delete group**" item of the [Project Manager](#) menu. After selection "**Yes**" button in confirmation dialog current group will be deleted. All projects from this group

will remain in **All** projects group.

Move Projects

To move one or more projects from current projects group to other group select with **Ctrl** or **Shift** key required projects from correct project list and select "**Edit->Move project(s)**" item of the [Project Manager](#) menu, which opens **Select a group** dialog



Select one group and previously selected projects will be moved to it. Projects resided in the **All** group can be only copied to other groups without deleting.

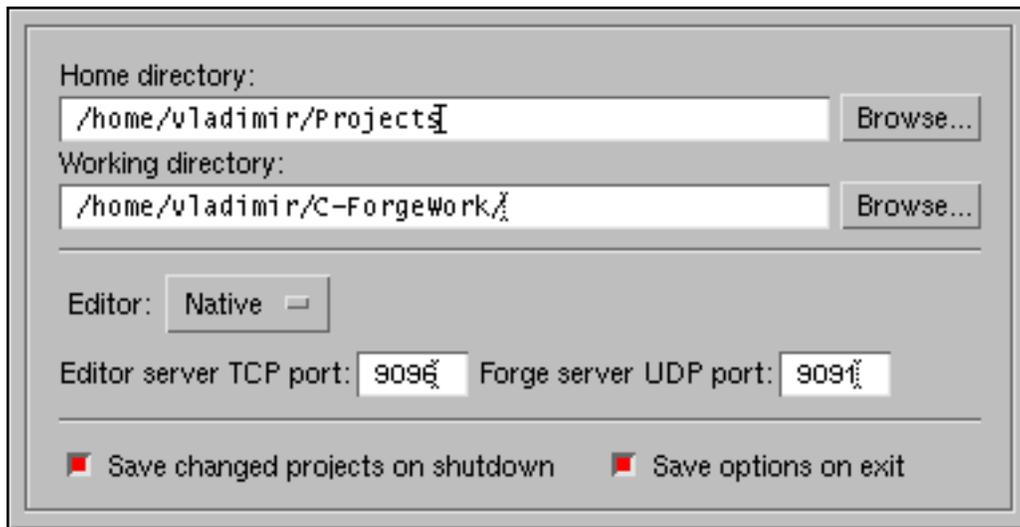
Last modified: Friday, 15-Jun-2001 09:31:27 EDT

Code Forge Options

The **Code Forge Options** dialog is used to modify IDE options. To open it, select **Code Forge Options** from **Options** menu of the [Project Manager](#).

All Code Forge options are divided into the following sections represented by the appropriate tabs:

General Options



Home directory field specifies the path for a home directory of a user where the [Stable Directories](#) of new projects are created by default. The **Browse** button opens [Select Path](#) dialog that allows directory selection.

Working directory field specifies the path for a working directory of a user where the [Working Directories](#) of the projects are created. The **Browse** button opens [Select Path](#) dialog that allows directory selection.

Editor specifies the editor, currently Code Forge supports its own *Native* editor and installed *CRiSP* editor.

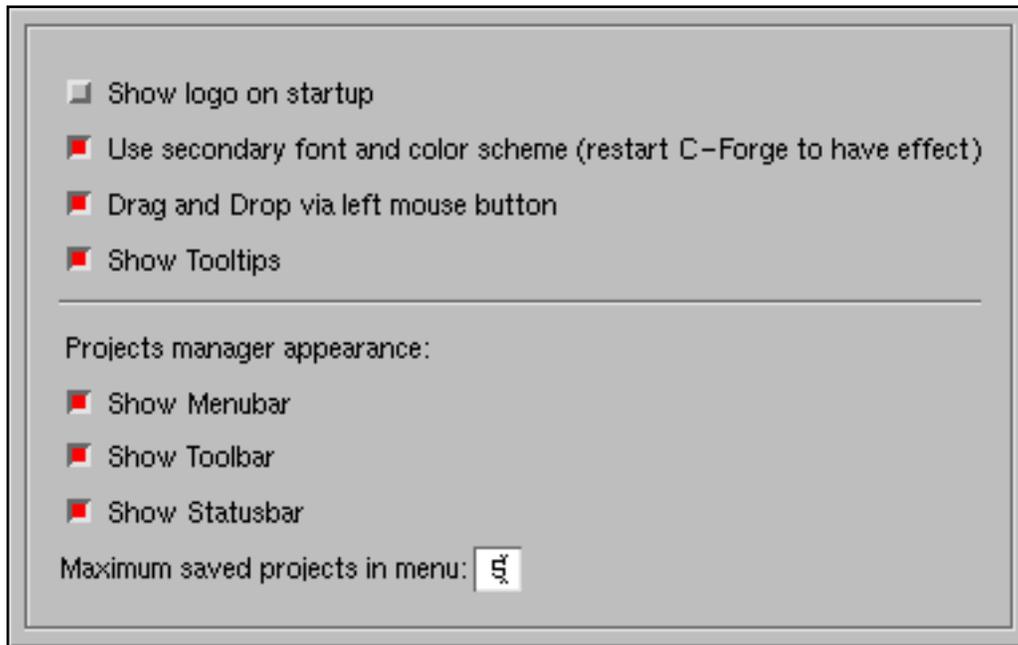
Editor server connection port specifies the TCP port of the editor server.

Forge server connection port specifies the UDP port of *forge_server*.

When the **Save changed projects on shutdown** check box is selected, all the changes will be saved if the process is closed by Termination Signal (SIGTERM).

When **Save options on exit** check box is selected, all modified options will be saved on the exit from the IDE; otherwise the previous set of options will be restored when the IDE is restarted.

Appearance



When **Show logo on startup** check box is selected, Code Forge logo appears when the IDE is launched.

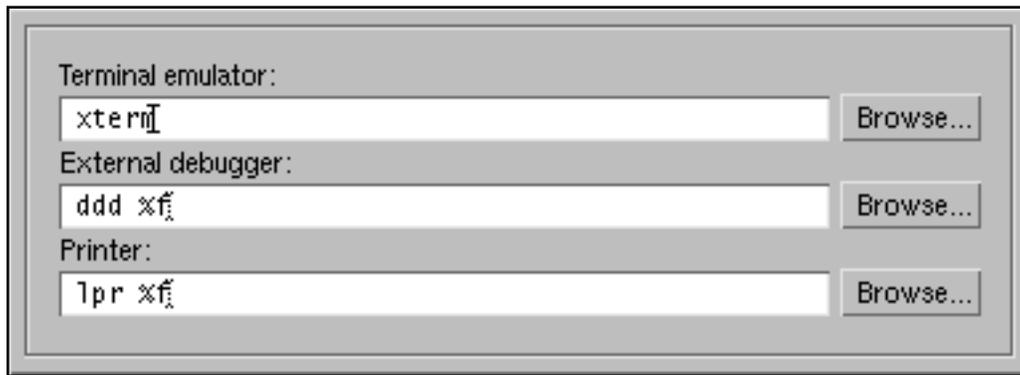
When **Use secondary font and color scheme** check box is selected, the secondary fonts and color scheme is used across the entire environment. You will need to restart Code Forge before new settings will take effect.

Drag and Drop via left mouse button check box enables d'n'd operations using left mouse button in addition to the middle mouse button (Button 2).

Show Tool tips, Show Menu bar, Show Toolbar and **Show Statusbar** check box's show and hide the Tooltips, Menu bar, Tool bar and Status bar.

Maximum saved projects in menu field specifies the number of recently saved projects that appear in the **File** menu in the [Project Manager](#).

Helpers

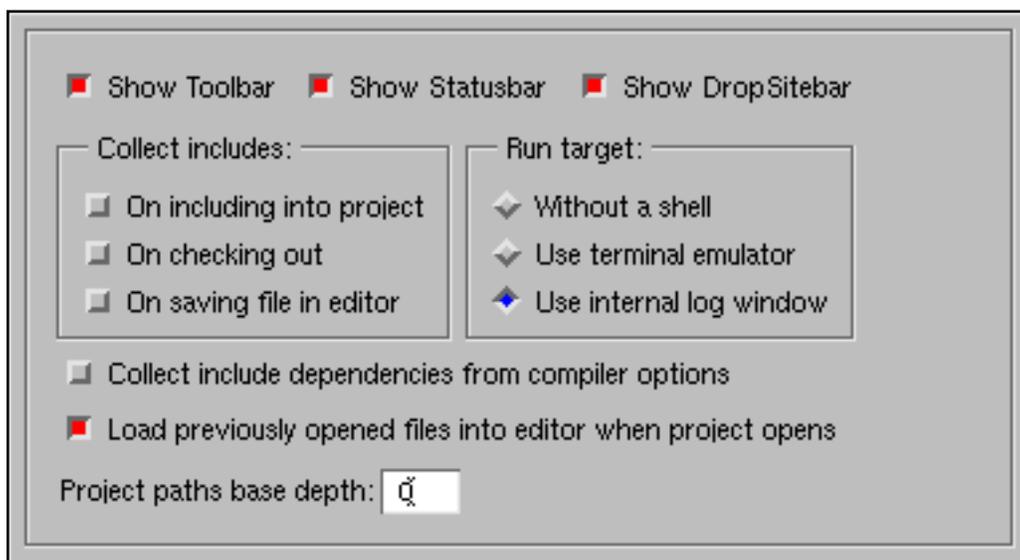


Terminal emulator field specifies a terminal emulator, used instead of [Project Log Window](#), when enabled in [Project Options](#) and as external terminal for text-mode debuggers.

External debugger field is used to specify external debugger.

Printer field is used to specify command to print files. To insert filename to be printed - use **%f**.

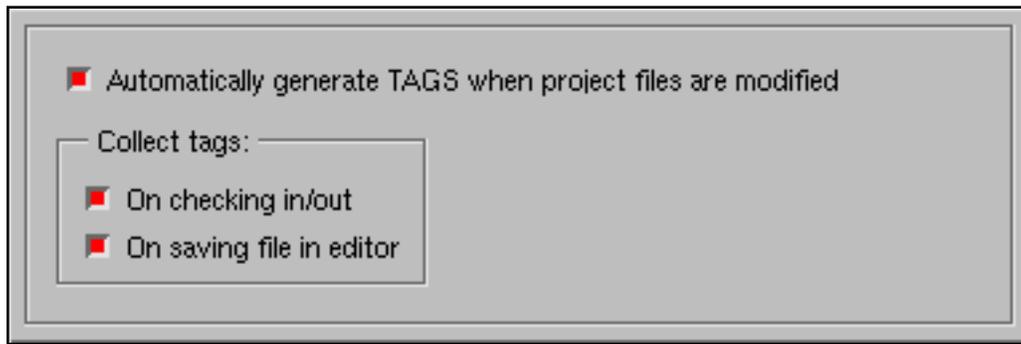
Project



When [Project Options](#) dialog is called from the Project Manager, all applied project settings will be used to initialize options in the new projects, created later.

If the [Project Options](#) dialog is called from the Project Desktop, all settings will be applied only to the current project.

Tags



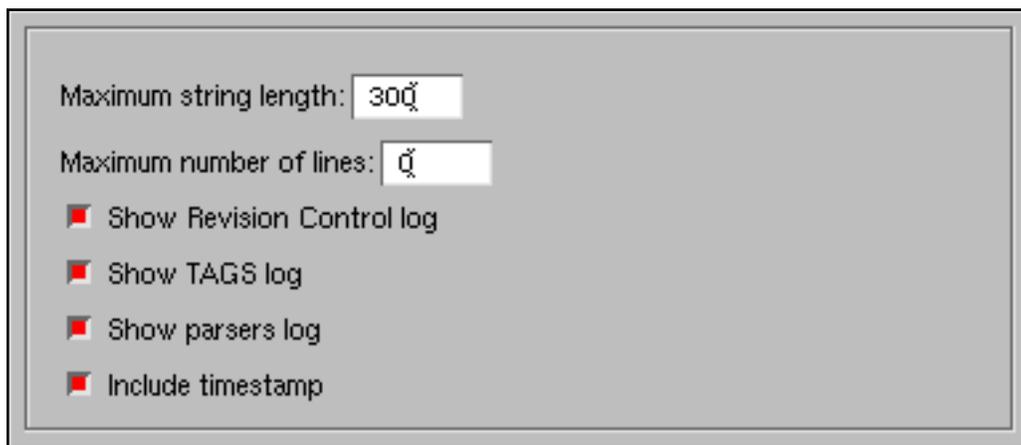
Automatically generate TAGS when project files are modified - generate tags when projects files are added, renamed, removed.

Following options are sensible when above check-button enabled:

On checking in/out - collect tags when file is checked in/out from/to a working area.

On saving file in editor - collect tags when changes are saved in the editor.

Log



Maximum string length determines the maximum length of a string in a [Log-window](#). If string length exceeds the specified value, the string will be wrapped.

Maximum number of lines determines the maximum number of lines in a [Log-window](#). If it is 0 then there is no limit for number of lines.

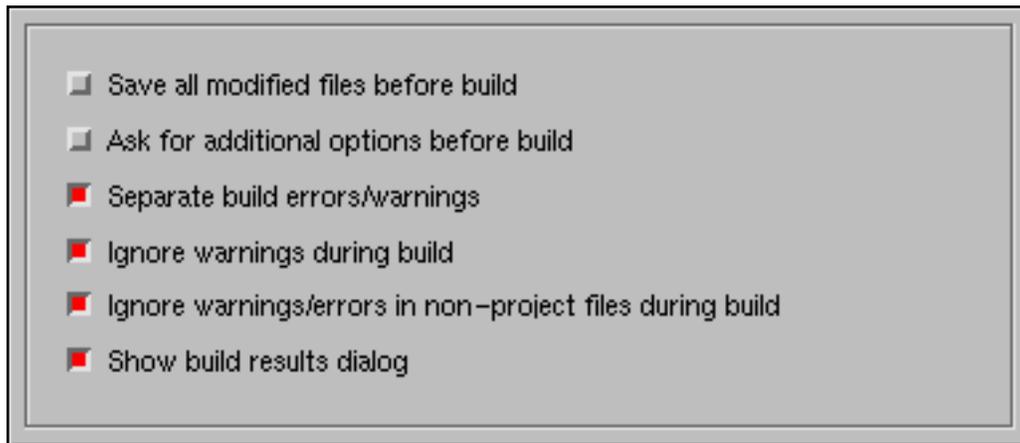
Separate build errors/warnings allows you to view errors and warning in the [Log Window separately](#) from the common log output.

Show Revision Control Log, **Show TAG log** and **Show parsers log** check boxes control whether messages of the Revision Control system, tags collection procedure and parsing files in the **Class Browser and Source Navigator** should be displayed in the log window.

Include timestamp check box turns on/off displaying of the time before message lines in the log

window.

Build



Build options control the following aspects of the build procedure and possible error corrections:

- **Save all modified files before build** - if check box on, C-Forge will mutually save all modified in editor files needed to the requested build. If off, C-Forge will ask confirmation for saving modified files.
- **Ask for additional options before build** - if on, C-Forge will ask for additional options for *make* command. This may be necessary for the imported project or for the build in the physical folders.
- **Separate build errors/warnings** - if on, C-Forge will put errors and warnings messages from the build process into separate folders in the [Log-window](#).
- **Ignore warnings during build** - if on, C-Forge will not load warnings messages into editor during [correct errors](#) action.
- **Ignore warnings/errors in a non-project files during build** - don't load to the editor files with errors and warnings which not belong to the project. Project specific files are the files which are in the dependency tree or lie under the project stable or working directory or one of the physical folders attached to the project.
- **Show build result dialog** - show the dialog which indicates the short summary of the last build.

Dialog Buttons

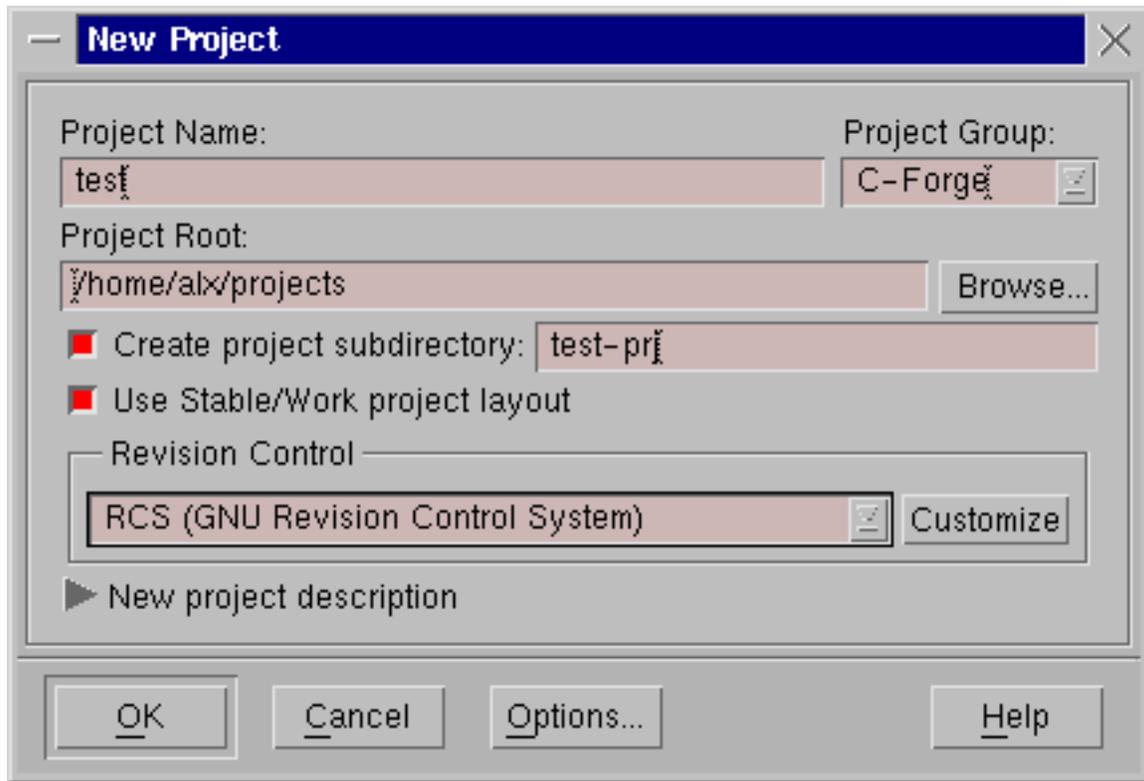
OK - saves the changes in IDE options and closes the dialog.

Cancel - closes the dialog without saving the changes.

Help - shows this Help window.

New Project Dialog

The **New Project** dialog is called by selecting **New project** command from the **File** menu of the [Project Manager](#) window or by clicking the **New Project** button and is used to [create new projects](#).



Project Name - specifies the name of the project.

Project group - dropdown-combobox that specifies the [projects group](#) where new project will be placed. If that group doesn't exist it will be created.

Project Root - specifies the root directory of the new project. The directory can be selected by clicking the **Browse** button (which opens the [Path Selection](#) dialog).

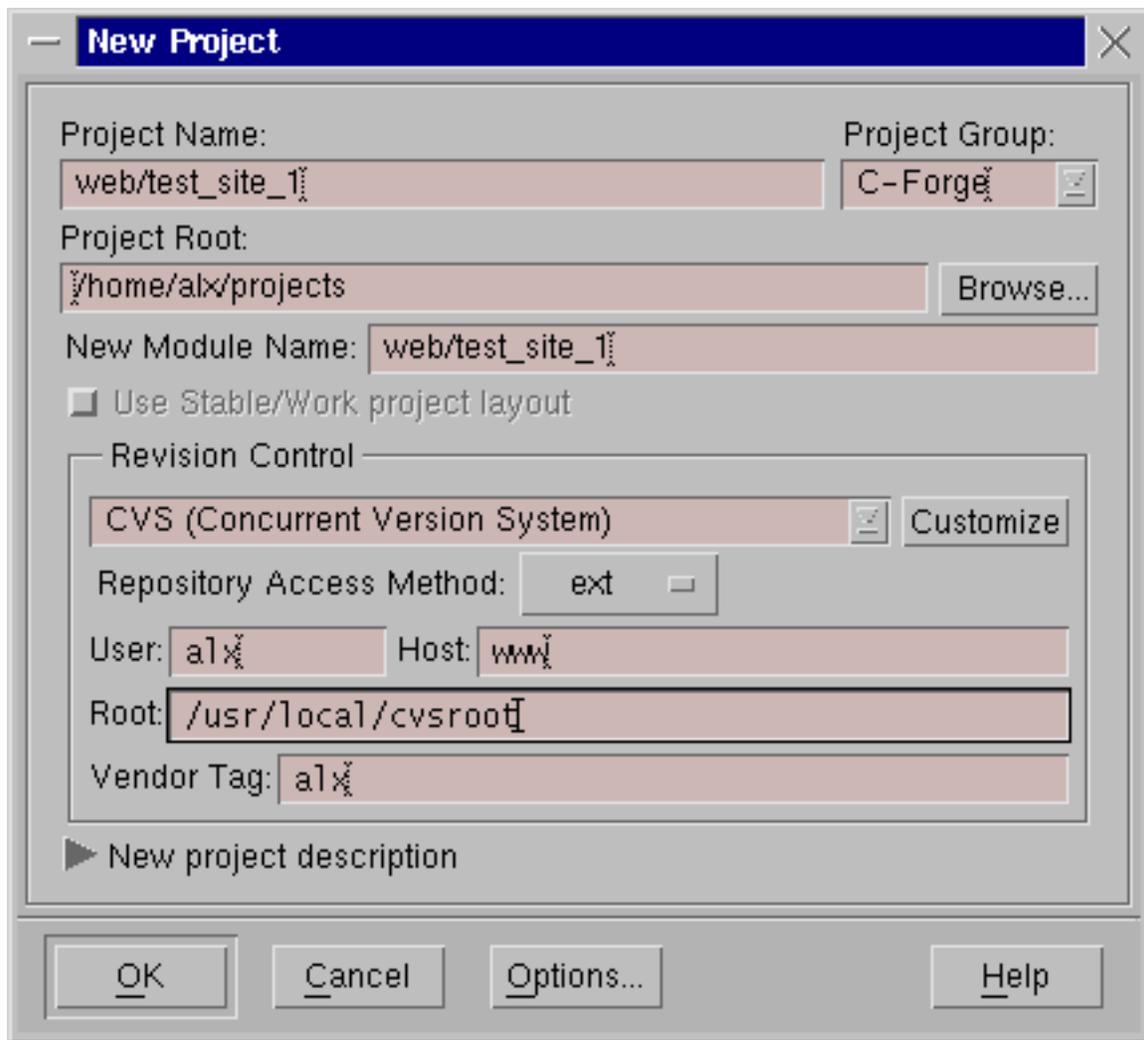
Create project subdirectory - specifies if project subdirectory should be created and the name of subdirectory.

Use Stable/Work project layout - specifies if project should have Working directory or not (This is useful for some types of projects).

New project description - clicking the arrow button will open an additional area with the following fields:

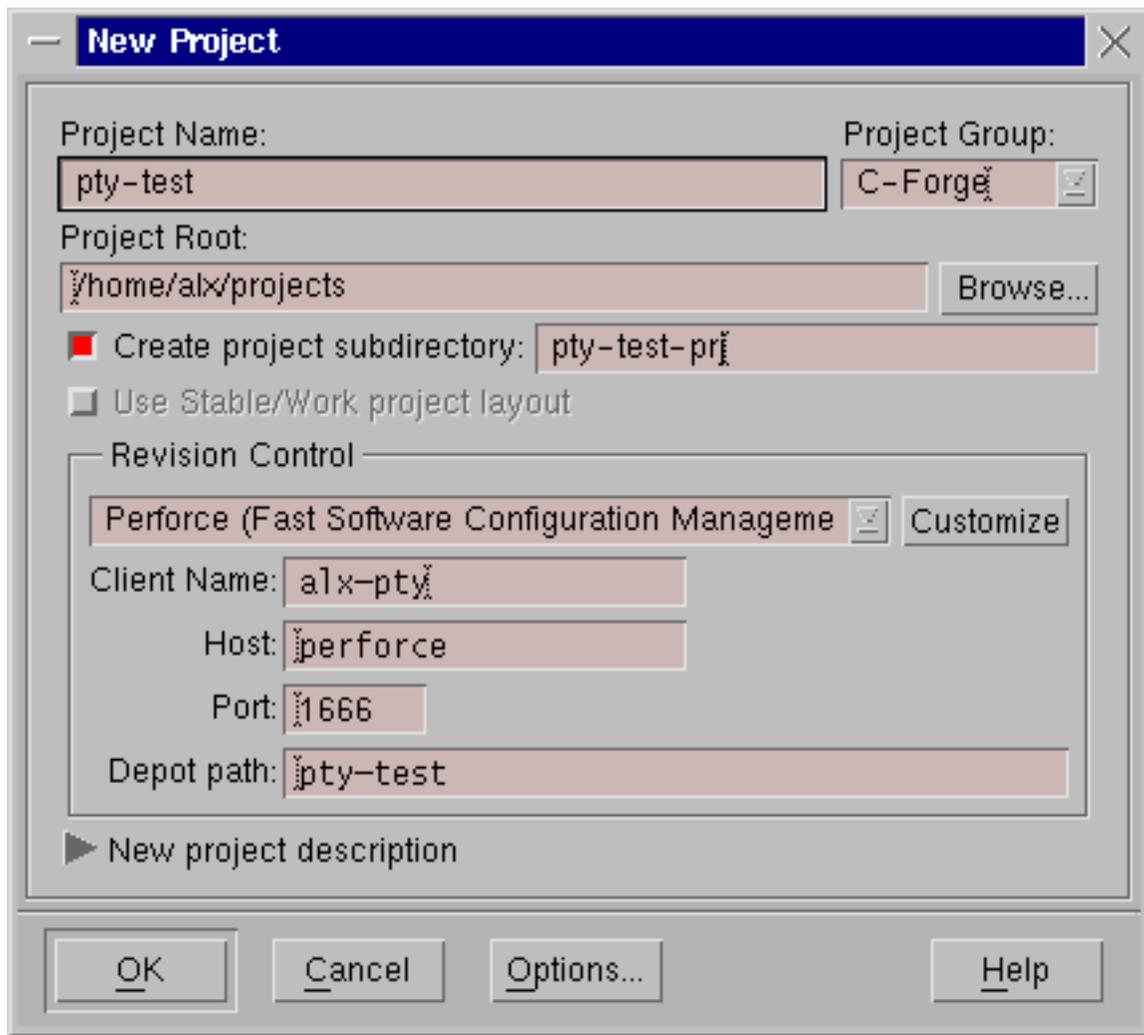
- **Author** - specifies the author of the project; the name of the user is inserted by default.
- **Description** - project description can be specified if necessary.
- **Comments** - comments can be entered if necessary.

Revision Control - drop-down combo-box specifying type of [revision control](#) to which the project will be subjected. Pressing **Customize** button right to the combo-box will call the [Revision Control Options](#) dialog, which helps to specify or change some of revision control options. Depending on the type of the revision control chosen the dialog can be slightly changed: if CVS revision control type has been chosen the dialog will include the following additional fields:



- **Repository Access Method** check-box allows to choose method of access to the repository
- **User** - if access to the remote repository is necessary, user name can be specified in this field
- **Host** - if access to the remote repository is necessary, host name can be specified in this field
- **Root** - if the local repository is used, repository root is specified in this field
- **Vendor Tag** - owner tag is specified by default and can be changed.

if Perforce revision control type has been chosen the dialog will include the following additional fields:



- **Client Name** - new client name can be specified.
- **Host** - host name can be specified.
- **Port** - port number can be specified.
- **Depot path** - relative path in the Perforce Depot directory tree can be specified for mapping to the newly created project.

Dialog Buttons

OK - accepts the entered fields, closes the New Project dialog and opens a [Create New Target\(s\)](#) dialog to create the first targets in the project.

Cancel - cancels the creation of the new project and closes the dialog.

Options - opens the [Project Options](#) dialog to specify the options of the new project.

Help - shows this Help window.

Last modified: Wednesday, 04-Jul-2001 10:13:18 EDT

Import Project Dialog

To be imported or attached an external project should include a root directory, including Makefile which consists of a collection of targets and their dependencies. The Project may also include source files and version control repositories.

Importing External Project

There are several [methods](#) of importing of an external project

To use any of them you have to invoke **Import project** dialog. In order to do that:

1. Select **Import project** from the **File** menu in the [Project Manager](#) window or click the **Import Project** button. Code Forge will display the **Import Project** dialog:

Project Name: C-Forge Project Group: All

Project Root: /home/s0l Browse...

Import from

Makefile Directory **Repository**

Import from existing Revision Control Repository

Revision Control

CVS (Concurrent Version System) Customize

Show repository import log

Repository Access Method: pserver

User: Host:

Root: /usr/cvsroot

Module Name: C-Forge Browse...

Repository import done action

Attach to received Directory Import from received Makefile

Execute:

New project description

OK Cancel Options... Help

- Turn on **Makefile** toggle button for the **Import method**, **Directory** toggle button for **Attach method**, if the Makefile and all other files already exist, or **Repository** for importing from the repository.

Project Name - specifies the name of the project.

Project group - dropdown-combobox that specifies the [projects group](#) where new project will be placed. If that group doesn't exist it will be created.

Project Root - specifies the location of the project. The directory can also be selected by clicking the

Browse button (which opens the [Select Path](#) dialog).

New project description - clicking the arrow button will open an additional area with the following fields:

- **Author** - specifies the author of the project; the name of the user is inserted by default.
- **Description** - project description can be specified if necessary.
- **Comments** - comments can be entered if necessary.

Dialog Buttons

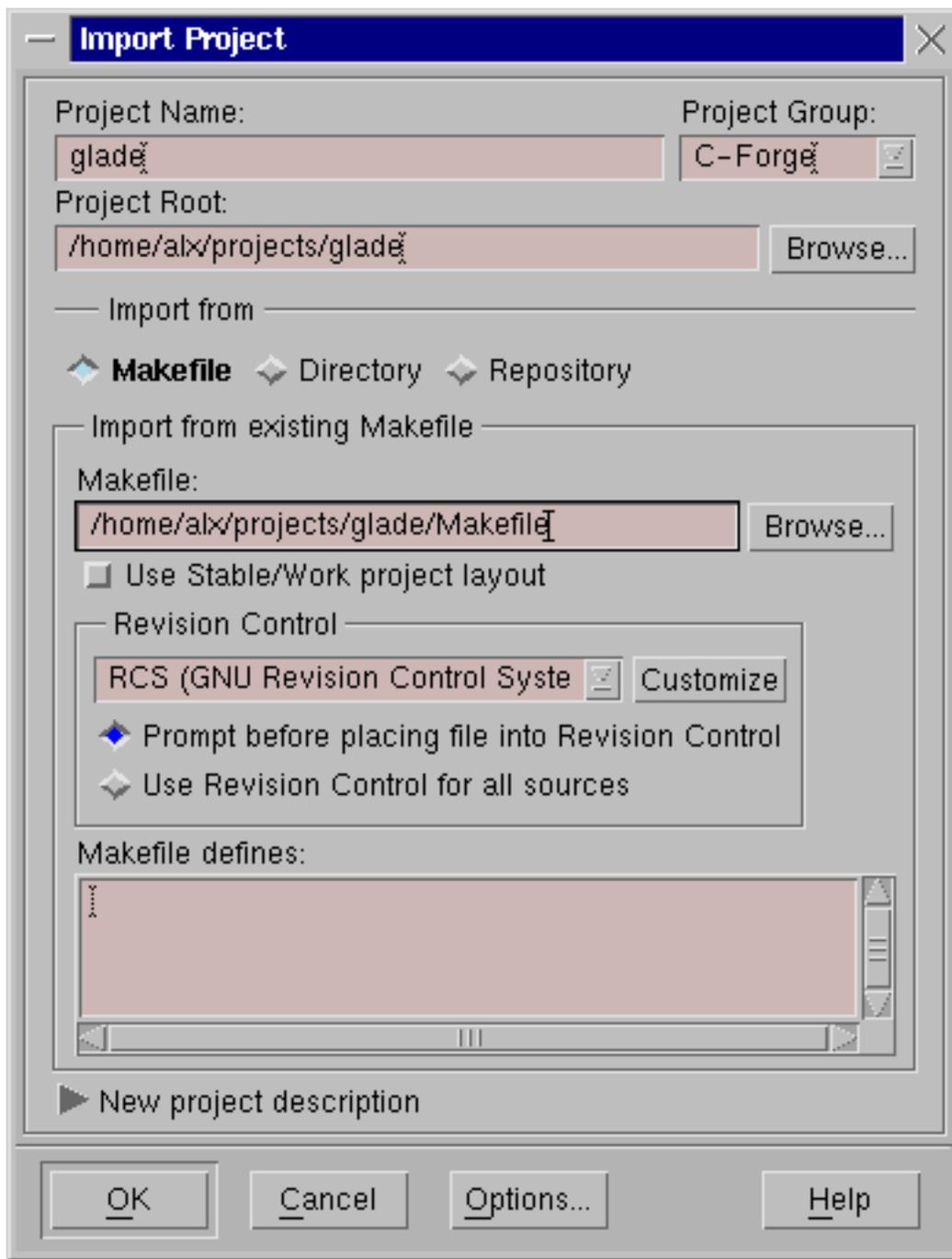
OK - imports the project.

Cancel - cancels the operation and closes the dialog.

Options - opens [Options](#) dialog window.

Help - shows this Help window.

Import method



Makefile toggle button is turned on.

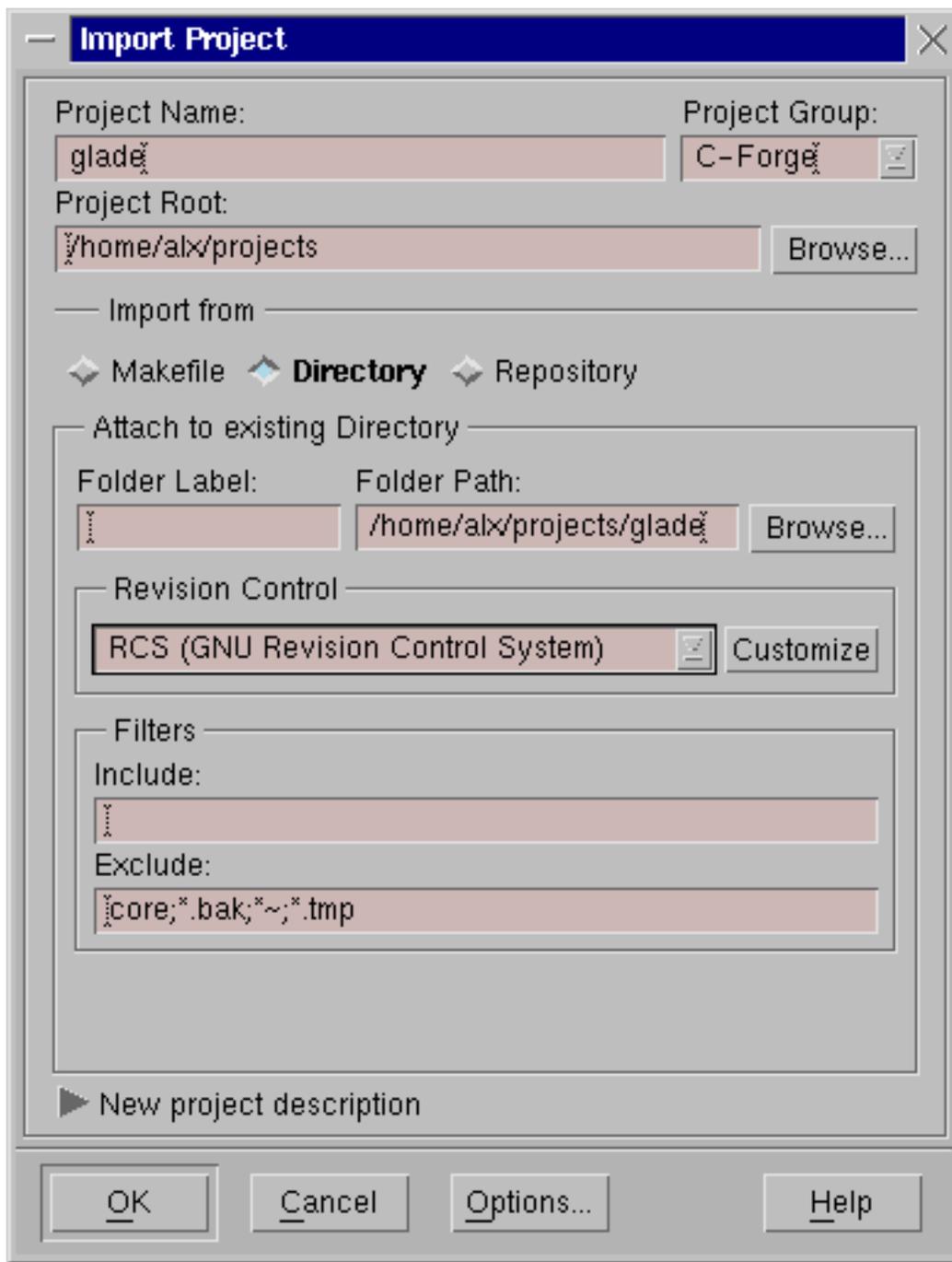
1. Enter the name of the project in the **Project name** field.
2. Enter the project root directory in the **Project Root** field. As usual it should be the path to the catalog from which the Makefile is being imported. (or press **Browse** button and the **Select Path** window dialog will appear on the screen).
3. Enter the path to the existing makefile being imported in the **Makefile** field (or press **Browse** button and the [Select a File](#) window dialog will appear on the screen)
4. Code Forge will try to figure out what kind of revision control should be used. By default any editable file imported into Code Forge will be placed under revision control. (The Version Control option may be turned off when a file is added to the project).

5. If Code Forge failed - choose revision control type in the **Revision control** combo-box. The **Customize** button right to the **Revision Control** combo-box will open the [Revision Control Options](#) dialog.
6. Choose one of the following toggle buttons, specifying how the new files of the project will be included into [Revision Control System](#).
 - **Prompt before placing file into Revision Control System** - Code Forge will prompt you before including any of the new files into RCS.
 - **Use Revision Control System for all sources** - all the new source files will be included into RCS.
7. **Makefile defines** - the values for compiling the conditional Makefile while importing can be specified.
8. Press **OK** button of the **Import project** window. The project window containing dependency tree of the Makefile will appear.

Code Forge uses the original directory of the project as a [Stable Directory](#). By default the [Working Directory](#) is not created. To create the Working Directory select the **Use Working Project Version** option. This option will enable the indirect editing of project files (see [Editing Files](#)).

If the MakeFile of the imported project contains macros with tools options, Code Forge includes these options into [Tools Options](#) dialog.

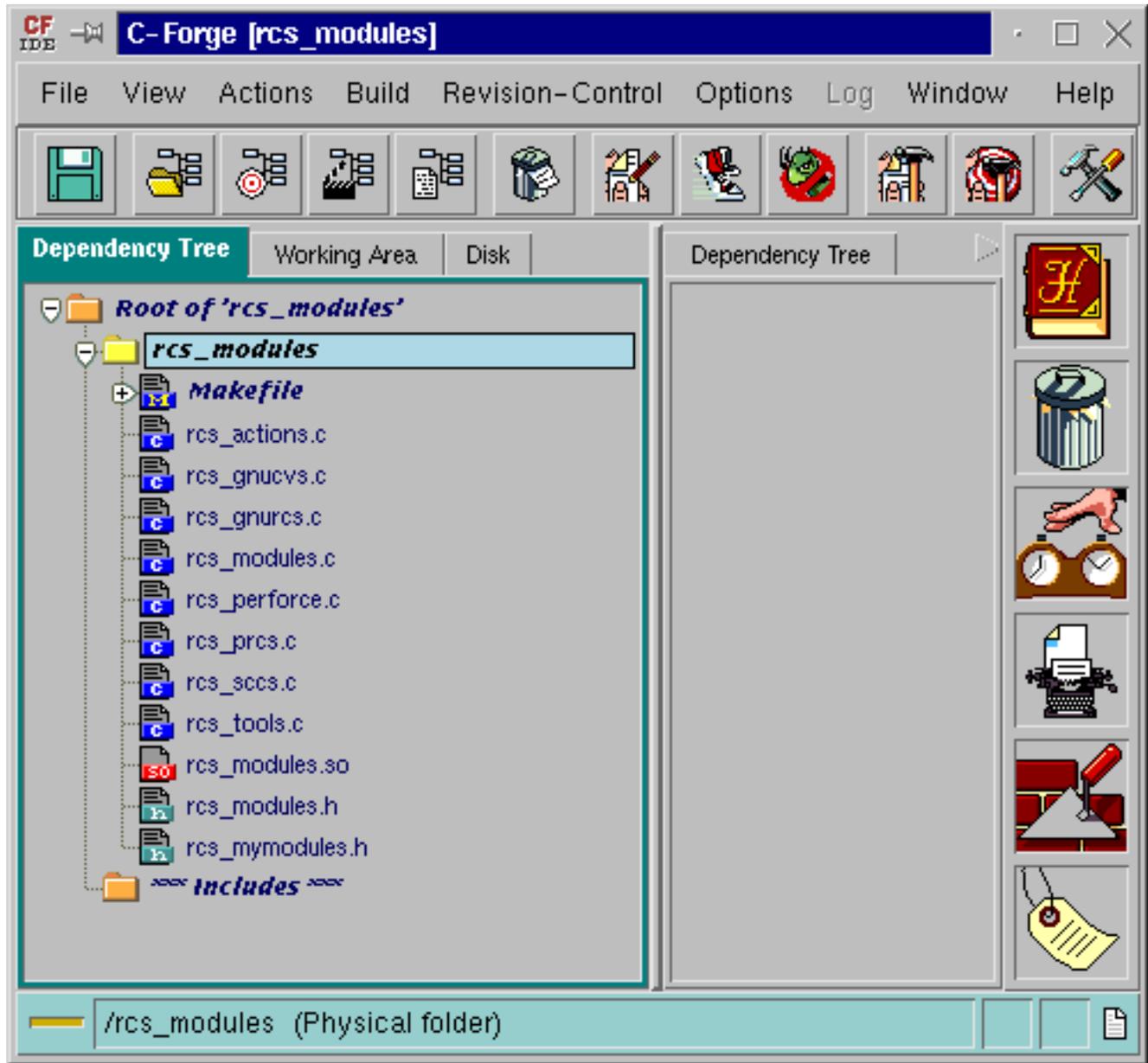
Attach method



Directory toggle button is turned on.

1. Enter the name of the project in the **Project name** field.
2. Enter the project root directory in the **Project Root** field (or press **Browse** button and the [Select Path](#) window dialog will appear on the screen).
3. Fill in **Folder Name** field with the name of attached directory
4. Fill in the field **Folder Path** with the attached directory path (or press **Browse** button and the [Select Path](#) window dialog will appear on the screen). Code Forge will try to figure out the type of revision control used.
5. Press **OK** button of the **Import project** dialog window. The small empty project will be created and the physical folder with specified name will be attached to it.

This Physical folder contains files and directories just like a file browser:



You can work with them like with targets in the tree. "Makefiles" in physical folders can be opened as folders and the list of targets of this Makefile appears. You can build targets in the Makefile or files in physical folders.

Import from revision control repository

Project Name: C-Forge Project Group: All

Project Root: /home/s0l Browse...

Import from

Makefile Directory **Repository**

Import from existing Revision Control Repository

Revision Control

CVS (Concurrent Version System) Customize

Show repository import log

Repository Access Method: pserver

User: Host:

Root: /usr/cvsroot

Module Name: C-Forge Browse...

Repository import done action

Attach to received Directory Import from received Makefile

Execute:

New project description

OK Cancel Options... Help

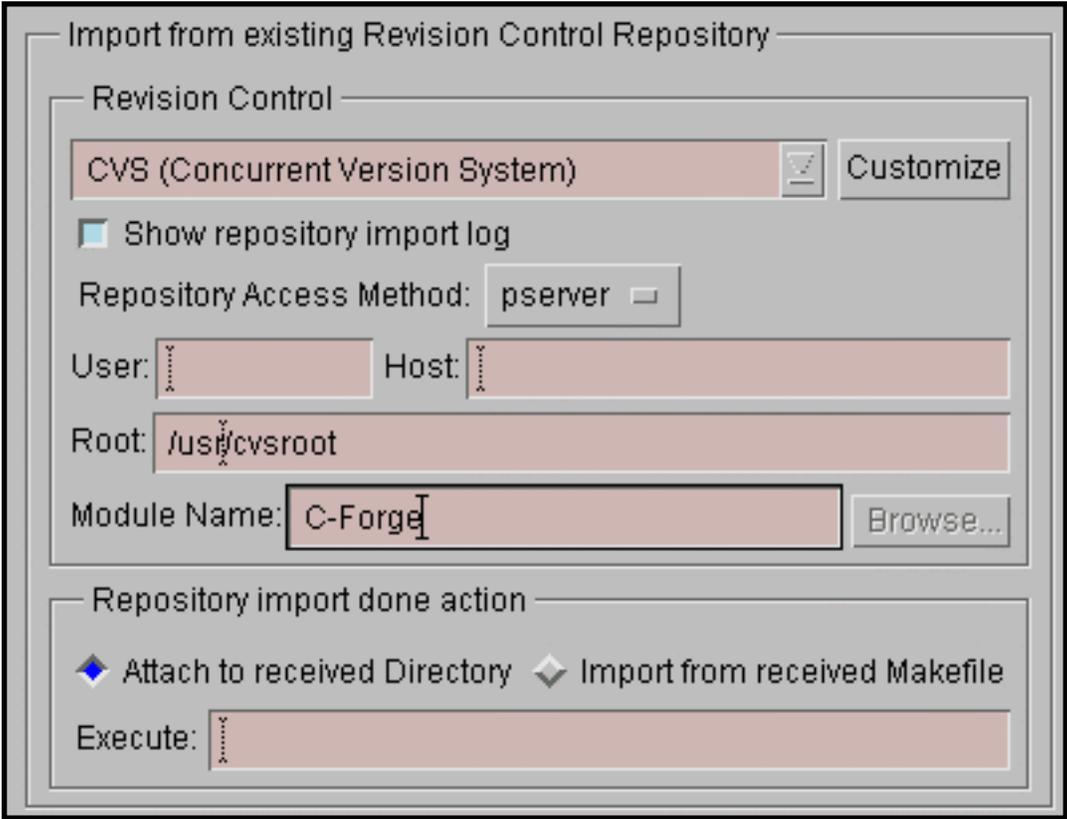
Repository toggle button is turned on.

1. Enter the name of the project in the **Project name** field.
2. Enter the project root directory in the **Project Root** field (or press **Browse** button and the [Select Path](#) window dialog will appear on the screen).
3. Select the revision control type in the **Revision Control** combo-box. After the revision control type has been chosen, new fields depending on this type will be added to the dialog window.
4. Select **Import** or **Attach** method for the retrieved project by turning on **Import from received Makefile** or **Attach to received directory** toggle buttons. When the **Import from received Makefile** method is selected, the [Select a File](#) window dialog will appear on the screen. The

selected with the help of this dialog Makefile will be imported. When the **Attach to received directory** method is selected, the new project will be created and the project root directory specified in the **Project Root** field will be attached to it.

 Show the repository import log - if this toggle button is pressed, the import log window will be opened on the screen while importing procedure.

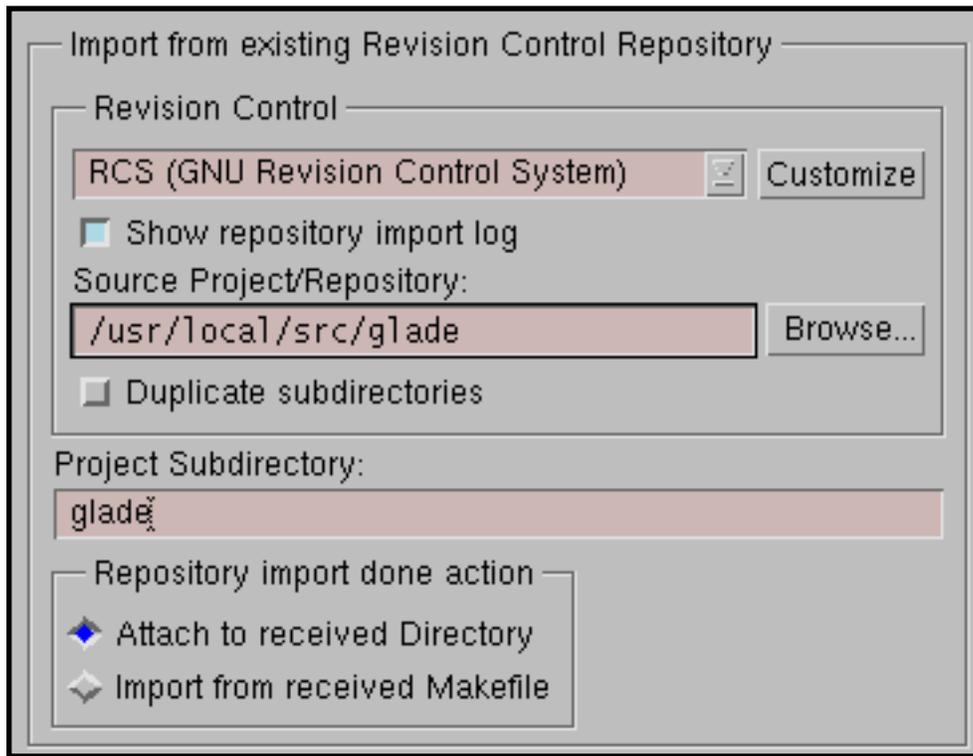
• If CVS type of revision control has been chosen, the following fields will be added to the dialog:



- **Repository Access Method** - method of access to the repository can be specified
- **User** - user name can be specified if the remote repository is worked on
- **Host** - host name can be specified if the remote repository is worked on
- **Root** - repository root can be specified if the local repository is worked on
- **Module name** - the name of the module being saved into CVS can be specified.

 'Browse' button will work only if 'local' method was specified. Module names are only listed and are selectable in local repositories.

• If RCS or SCCS type of revision control has been chosen, the following fields will be added to the dialog:

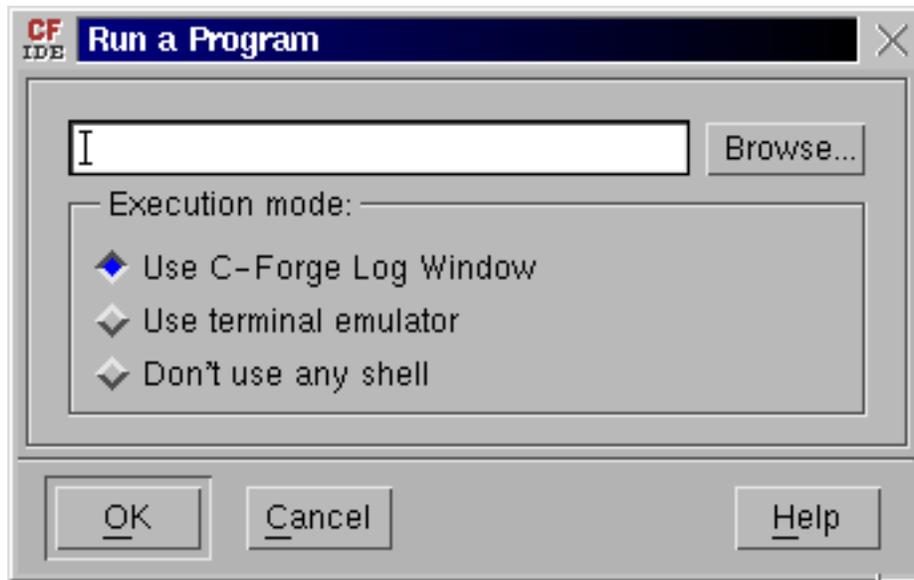


- **Source Project/Repository** - - name of the repository or source project should be specified in this field. The path to the folder can also be selected by clicking the Browse button (which opens the [Select Path](#) dialog).
- **Duplicate subdirectories** toggle button should be pressed if you wish to duplicate subdirectories, included into the repository, in your project.
- **Project Subdirectory** - specify the name of the catalog, where the files from the repository will be copied.

Last modified: Monday, 01-Apr-2002 09:05:26 EST

Run a Program

The **Run A Program** dialog is used to execute an external program. It can be accessed by selecting **Run** from the **Actions** menu in the [Project Manager](#).



Commands can be entered, as they would be on the actual command prompt. The **Browse** button opens the [File Selection](#) dialog that allows the selection of a file to be executed.

Execution mode selects the shell that will be used to run the command:

- **Use Code Forge Log Window** - [Log-window](#) is used as a shell.
- **Use Terminal emulator** - program, defined in [Code Forge Options](#) as **Terminal Emulator** is used, *xterm* is used by default.
- **Don't use any shell** - no shell is used.

Dialog Buttons

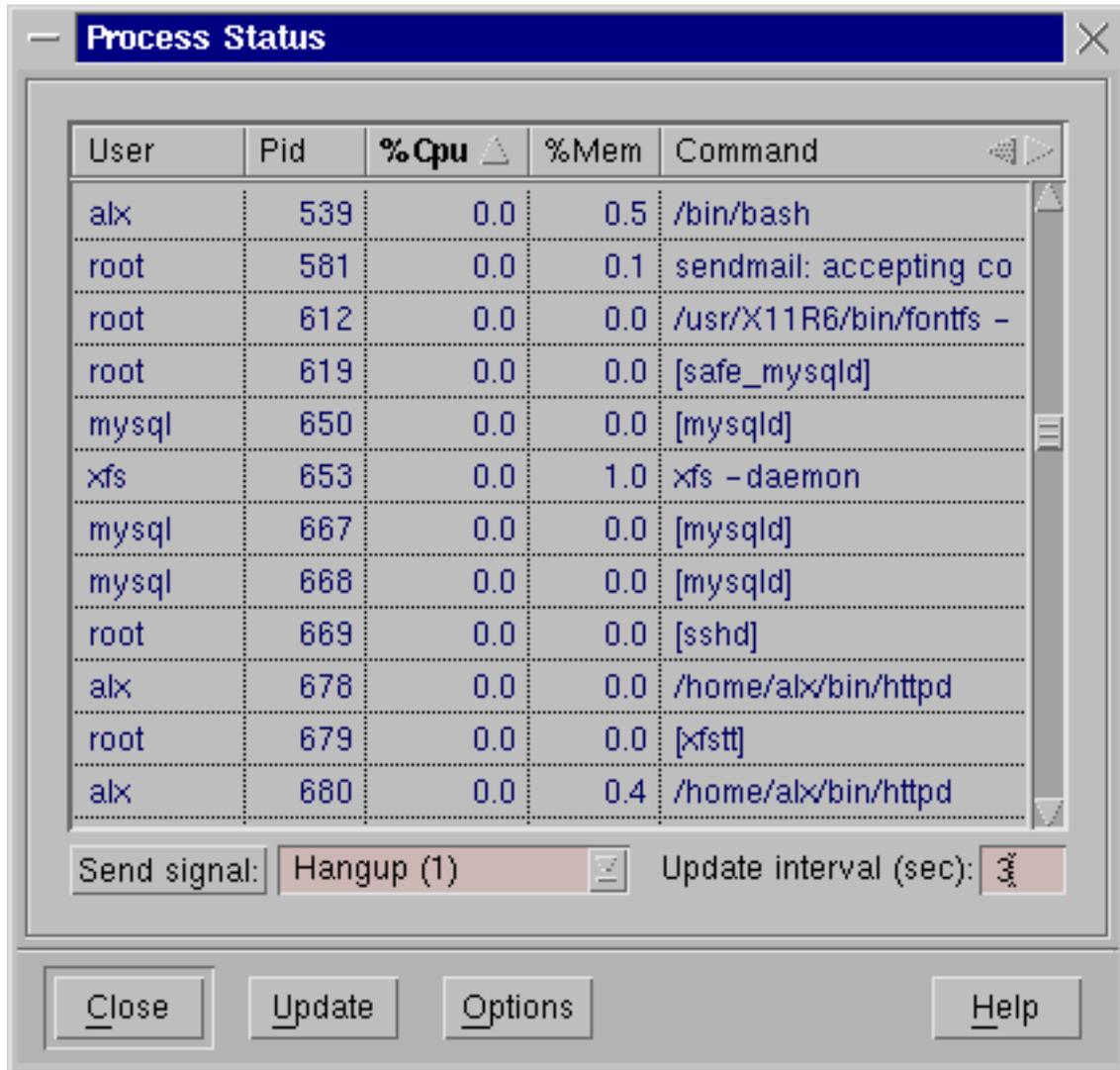
OK - executes the command.

Cancel - cancels the execution.

Help - shows this Help window.

Process Status

The **Process Status** window displays the state of the active processes in the system. It also allows the sending of signals to selected processes. Selecting **Process Status** in the **Actions** menu of the [Project Manager](#) window accesses this screen.



The main part of the window displays a table of process information. The title bar is a standard [interface](#) element that allows changing the column characteristics and element sorting.

The **Send signal** button sends the signal, displayed in the combo-box, to the selected process. If the user is not permitted to send a signal to a process, an error message is displayed.

The **Update interval** specifies the window refresh frequency. The new interval takes effect only after the **Update** button is pressed.

Dialog Buttons

Close - cancels all the changes and closes the dialog.

Update - updates the screen.

Options - allows you to change a process state command.

Help - shows this Help window.

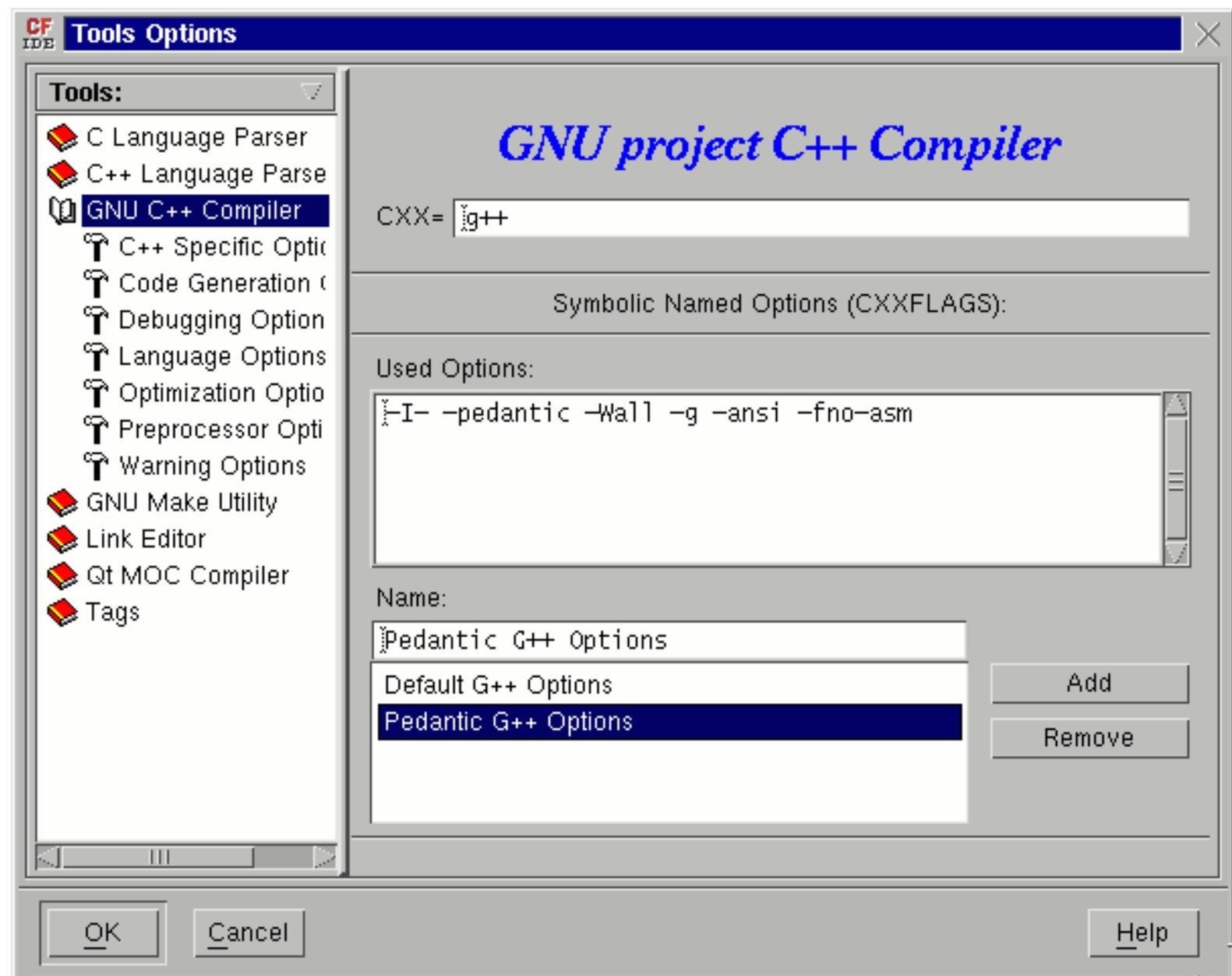
Last modified: Monday, 14-May-2001 07:03:19 EDT

Tools Options

The **Tools Options** dialog is used to modify tools options.

This dialog is activated when **Default Tools Options** is selected from the **Options** menu of the [Project Manager](#) window or when **Tools Options** is selected from the **Options** menu of the [Project Desktop](#) window.

When the dialog is opened from the [Project Manager](#), it modifies the all default tools options for new projects, otherwise options for tools used in the current project are modified.



The left side of the dialog displays the list of available tools with groups of related options. Click on the option group to modify.

Name list-box displays the names of previously saved options. It always includes the default set of options.

New sets of options can be created and saved for future use.

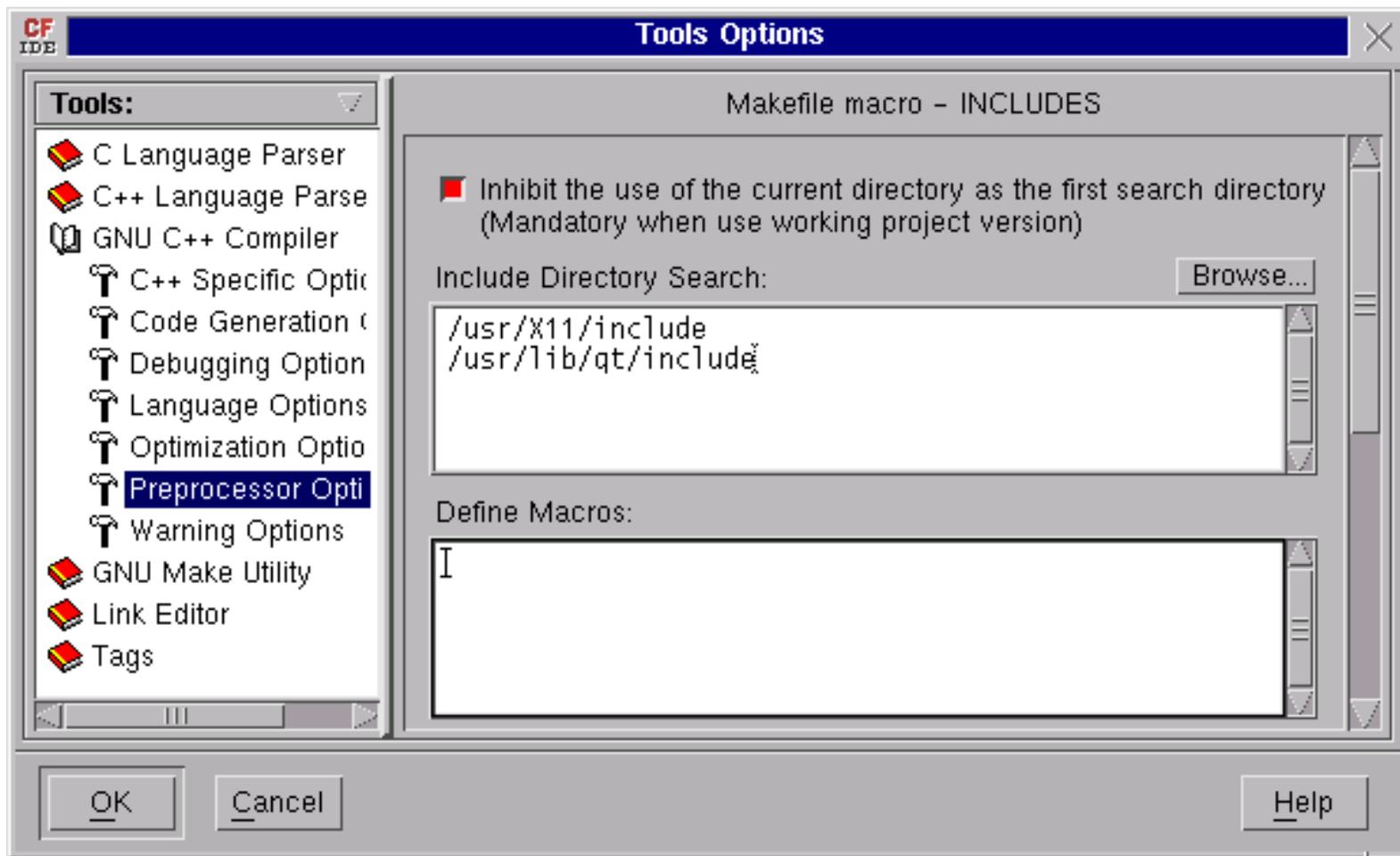
To create a new set of options for a tool:

1. Select the tool in the left part of the dialog.
2. Edit the list of options in a **Used Options** text-box.
3. Enter the name of a new set into the **Name** text-box.
4. Press the **Save** button to save the changes.

To load an existing set of options, select its name in the **Name** text-box and press return.

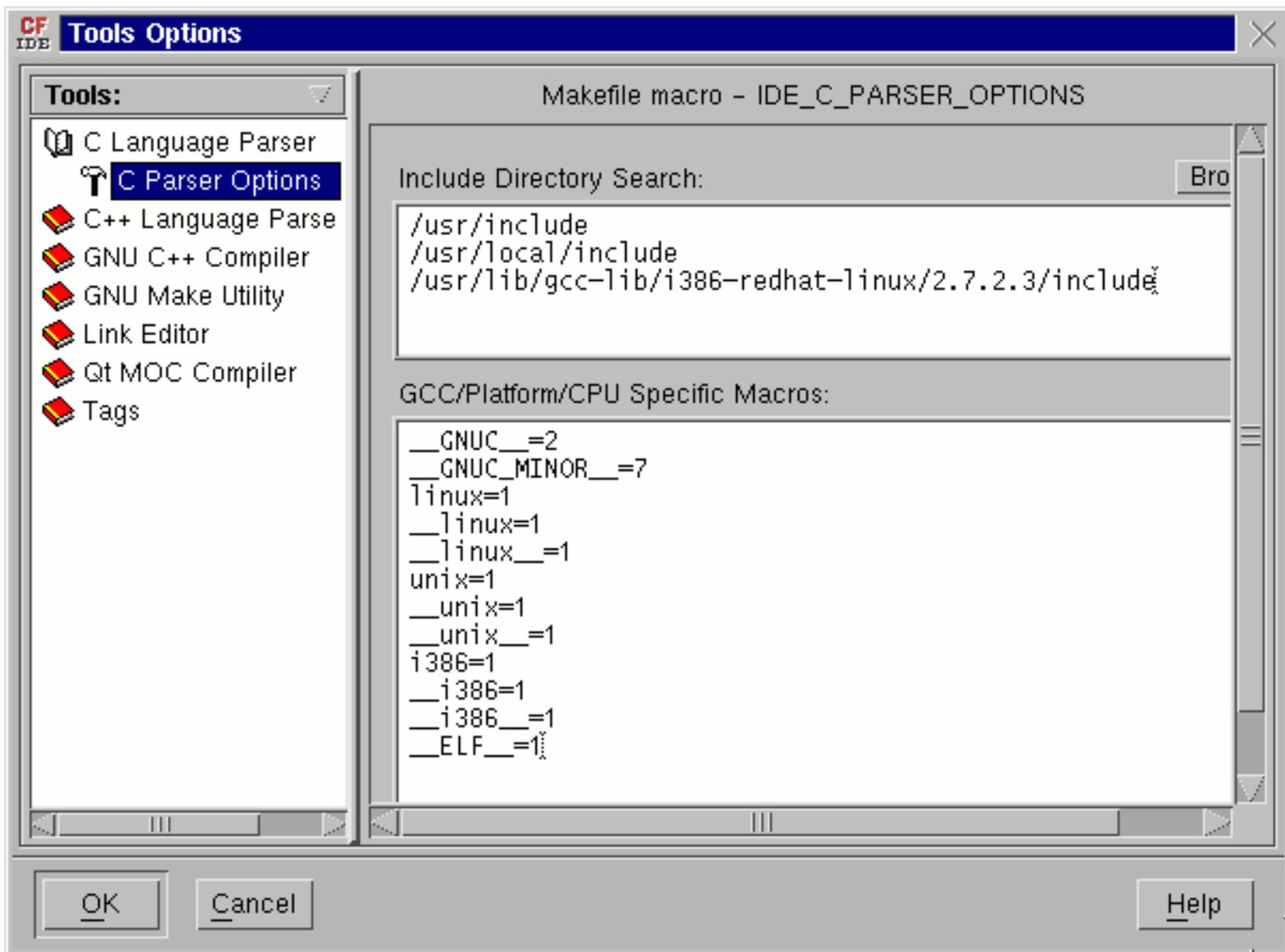
To delete an existing set of options, select its name in the **Name** text-box and then press **Remove** button.

If a group of options is selected in the left side of the dialog, the right side displays the description of the selected group options using text-boxes, combo-boxes, check-boxes and other option groups. Some fields can be filled using the **Browse** button located next to them.



In order for compiler flag pop-up tips to appear, select the **Show tips** check-button in [C-Forge Options](#) dialog.

For normal using [Word Completion](#) and [Symbol Navigator](#) you must specify the paths and default options for each **Parser Tool**:



Dialog Buttons

OK - saves the changes in tools options and closes the dialog.

Cancel - closes the dialog without saving the changes.

Help - shows this Help window.

Last modified: Friday, 15-Mar-2002 11:58:38 EST

Working with Projects

- [Creating a new project](#)
- [Importing a project](#)
- [Creating a Multi-user Project](#)
- [Opening a project](#)
- [Adding files and macros to project](#)
 - [Adding targets](#)
 - [Adding source files](#)
 - [Adding a new source file](#)
 - [Adding an existing source file](#)
 - [Adding include files](#)
 - [Collecting includes](#)
 - [Adding macros](#)
 - [linked macros](#)
- [Adding Folders](#)
- [Deleting files](#)
- [Editing files](#)
- [Debugging files](#)
- [Making targets](#)
- [Running targets](#)

Creating a New Project

To create a new project:

1. Select **New project** from the **File** menu in the [Project Manager](#) window or click the **New Project** button.

C-Forge displays the [New Project](#) dialog.

2. Complete the fields of the dialog and click the **OK** button.

C-Forge creates a [Stable Directory](#) for the new project at the location specified by the New Project dialog. The name of the Stable Directory is composed from the name of the project plus the **-prj** suffix.

C-Forge also creates a [Working Directory](#) at the location specified by [C-Forge Options](#). The Working

Directory has the same name as the project.

Example: If you create **TestPrj** project in the `/home/user_name` directory and `/home/user_name/C-ForgeWork` is specified in [C-Forge Options](#) dialog as the root directory for the Working Directories then C-Forge creates the Stable Directory `/home/user_name/TestPrj-prj` and the Working Directory `/home/user_name/C-ForgeWork/TestPrj`.

Importing a Project

To import a project,

1. Select **Import project** from the **File** menu in the [Project Manager](#) window or click the **Import Project** button.

C-Forge displays the [Import Project](#) dialog.

2. Complete the fields of the dialog and click the **OK** button.

C-Forge uses the original directory of the project as a [Stable Directory](#).

By default the [Working Directory](#) is not created. To create the Working Directory press **Use Stable/Work project layout** toggle button of the [Import Project](#) dialog. This option will enable the indirect editing of project files (see [Editing Files](#)).

If the MakeFile of the imported project contains macros with tools options, C-Forge includes these options into [Tools Options](#) dialog. These can be edited later through this dialog.

For more information about importing projects see [Working With External Projects](#) and [Import Project Dialog](#).

Creating a Multi-user Project

C-Forge supports work with multi-user projects. This ability is useful when several users are working on the same project, even if some of them are not using C-Forge.

The user creates a new project under revision control and exports the project into Makefile by selecting **Export** command of the **File** menu of the [Project Desktop](#). Thus, the Makefile is created. It is automatically placed under revision control. Now this Makefile can be used by those who do not work

under C-Forge as a usual Makefile, and by any user working under C-Forge for making their changes accessible to others.

In order to get access to the project, other users should [import or attach the existing Makefile](#) by selecting **Import project** command from the **File** menu of the [Project Manager](#).

After other users have made their own changes in the project and wish those changes to be accessible, they should choose **Export** command from the **File** menu of the [Project Desktop](#). Thus, the new revision of the Makefile is placed under revision control. Those who need to get the latest version should select **Merge** command from the **File** menu of the [Project Desktop](#). When merging the changes made by other users will be implemented into the Project and changes made by this user will be also saved.

That is why while performing **Merge** operation a conflict between revisions of the file may arise. In such a case the **Question** dialog window will appear, warning the user that the conflict has arised and asking if the user would like to merge versions anyway. If the user answers in affirmative, the Makefile in question will be opened in the [Editor window](#) where the conflict points will be marked with >>> and <<< symbol sets. The user should edit the Makefile, and when all the conflicts are solved, save it. Now merging will be completed successfully. If the user rejects to merge versions, he should repeat merging command next time, when he needs to get the latest project revision.

Opening Projects

To open an existing project, double click on its icon in the [Project Manager](#) window

-Or-

1. Select **Open project** from the **File** menu in the [Project Manager](#) window.

C-Forge displays the [File Selection](#) dialog.

2. Select the file in the dialog and click the **OK** button.

C-Forge displays the [Project Desktop](#) with the selected project.

To open one of the recently saved projects, select its name from the bottom of the **File** menu in the [Project Manager](#).

If the project has no [Working Directory](#), C-Forge creates it at the location specified by [C-Forge Options](#) (except for the projects imported without the creation of [Working Directory](#)).

Example: If you open the project named **TestPrj** that has no working directory and **/home/your_name/C-ForgeWork** is specified in [C-Forge Options](#) dialog as the root directory for Working Directories then C-Forge creates the Working Directory **/home/your_name /C-ForgeWork/TestPrj**.

Adding Files and Macros to Project

You can add targets, source files, include-files and macros to a project.

Adding New Targets

To add a target to a project:

1. Select **New Target** from the **Actions** menu in the [Project Desktop](#) window or click the **New Target** button.

C-Forge displays the [Create New Target\(s\)](#) dialog.

2. Specify the type of a new target.
3. Enter the name of a new target.
4. If you wish to use three different directories for source files, include files and object files, enter the name of these directories or click the **Browse** buttons. Otherwise leave the corresponding text-boxes blank or clear **Directory Layout** check-box.
5. Press the **Enter** key and specify another targets, or click the **OK** button to add targets into the project.

The new target is inserted into the current folder of the [Dependency tree](#). A target can later be added to another target as a dependency by dragging it while depressing the **Ctrl+Shift** keys and dropping it on the icon of the main target.

You can add several targets with the same name to a project.

Adding Source Files

Adding a New Source File

To add a new source file to a project,

1. Select a node in the [Dependency tree](#) where the source is to be added.
2. Select **New Source** from the **Actions** menu or click on the **New Source** button.

C-Forge displays the [Create New Node\(s\)](#) dialog.

3. Specify the type of a new source.
4. Enter the name of a new source.
5. Press the **Enter** key to add another sources, or click the **OK** button to add targets into the project.

Adding an Existing Source File

For adding existing files to project - select files on [Disk view](#) and Drag'n'Drop them to the target icon in the [Dependency Tree](#). The [Add Nodes\(s\)](#) dialog will be popped up.

1. Select an operation type (**Move, Copy, Link**) for this source.
2. Correct the **Destination Path** if you wish to place this source to different location.
3. Make sure that all source types are right.
4. Check-out the **Revision Control** check-boxes if you don't wish to use [RCS](#) for some source.
5. Click the **OK** button.

Adding Include Files

To add an include file to a project, follow the instructions for adding [new](#) or [existing](#) source file, But for include file make sure that **Add as Include** mark is enabled.

Only in this case include files would be placed in special folder, not into dependency list of given target!

Collecting the includes

When [make a target](#) is activated, C-Forge searches all the files pointed in include directives and add them to the project. The search is made in Include folders of targets, Stable Directory of the project and the directories specified in directory list of the [Tools Options](#).

The include collection can be activated manually by selecting **Collect all include** in the **Actions menu** or by pressing **Ctrl+A** shortcut. To collect the include files for one target, single-click on the target in the [Dependency tree](#), and then choose **Determine Implicit Dependency** in the pop-up menu, or press **Ctrl + I (shortcut)**.

To make C-Forge collect the includes automatically, select **Implicit Dependencies** check-boxes in the [Project Options](#) dialog.

The files in **Includes** folder are sorted by name.

Adding Macros

C-Forge separates **Macros** into two distinct groups: (1) those that contain references to project files or are themselves included in the dependency list of a target and (2) all other macros. The macros of the first type are displayed in the [Dependency Tree Macros](#) folder; all other macros are listed in the [Project Macros](#) dialog.

You can add a macro of the first type by clicking on the Add Macro button of the Project Desktop toolbar or by choosing **New Macros** from **Actions** menu. This action displays the [Single Value Input](#) dialog. Enter the name of a macro and click **OK** button.

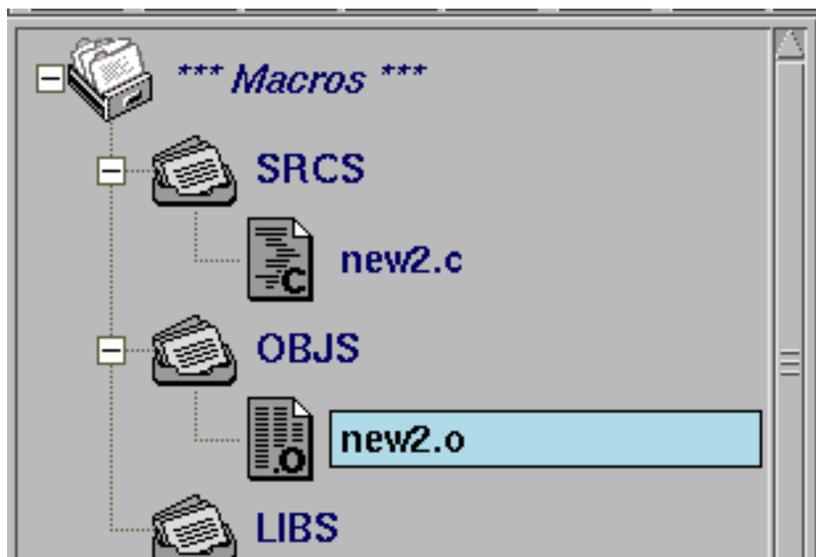
To add the macro of the second type, open the [Project Macros](#) dialog by choosing **Macros** from the **Options** menu.

Macros displayed in the [Dependency Tree](#) are similar to targets. You can add new files and targets to a macro the same way you add them to a target. You can also use [drag'n'drop](#) operation to add a file or a target to the list of dependencies of a target or to another macro.

Linked Macros

The list of dependencies for a library or an executable file may contain macros that include object files. If this list does not contain macros with corresponding source files, then C-Forge links them in the following way:

1. C-Forge searches for the substring OBJS or SRCS in the macros names;
2. If the substring is found, the substitution OBJS <- SRCS is performed and the macro with the new name is searched;
3. If the macro with a new name is found both of the macros are added to the [Dependency Tree](#) of a project.



If a project contains two macros named `*OBJS*` and `*SRCS*`, then when you add a new source file to `SRCS` macro, C-Forge adds a corresponding object file to `OBJS` macro.

Adding Folders

To add a folder to a project:

1. Select a folder where you want to create a new one.
2. Press **Add Folder** button of the Project Desktop toolbar or select **Folder** command from the **New** menu (**Actions** menu of the [Project Desktop](#) window). Depending on the type of a node or folder which is selected when the new folder is created, C-Forge displays the [New Directory](#) dialog (the new folder is being created in the physical folder), then
 - Enter the name of the new directory in the appropriate field of the dialog
 - Press **OK** button of the dialog window

or [New Folder](#) dialog, then

- Enter the name of a new folder.
- Specify the type of a new folder in the **Folder type** combo-box.

If [virtual folder](#) is being created, check if necessary the **Move selected nodes to new folder** check-box and press **OK** button of the dialog window. The virtual folder has been created.

If [physical folder](#) is being created:

1. Specify the path to the folder being created in the **Folder Path** field. The path to the folder can

also be selected by clicking the **Browse** button (which opens the [Select Path](#) dialog). When **Browse** button is used for path selection, the **Revision Control** field will be automatically filled in with the revision control type of the selected folder.

2. Specify the type of the revision control to which the folder will be subjected in the **Revision Control** combo-box. Pressing **Customize** button right to the combo-box will call the [Revision Control Options](#) dialog, which helps to specify or change some of revision control options.
3. If you need to fetch files from the repository, press **Fetch from the repository** toggle button.
4. Specify the name of the catalog, where the files from the repository will be copied to in the **Subdirectory** or **Module name** field.
5. Specify filters in the **Filters** box fields, if necessary.
6. Press **OK** button of the dialog window. The new folder has been added into the Dependency Tree.

Deleting Files

To delete a file or several files,

1. Select a node or several nodes in the [Dependency tree](#).
2. Select **Remove** from the **Actions** menu

-Or-

Click the **Delete** button in the toolbar.

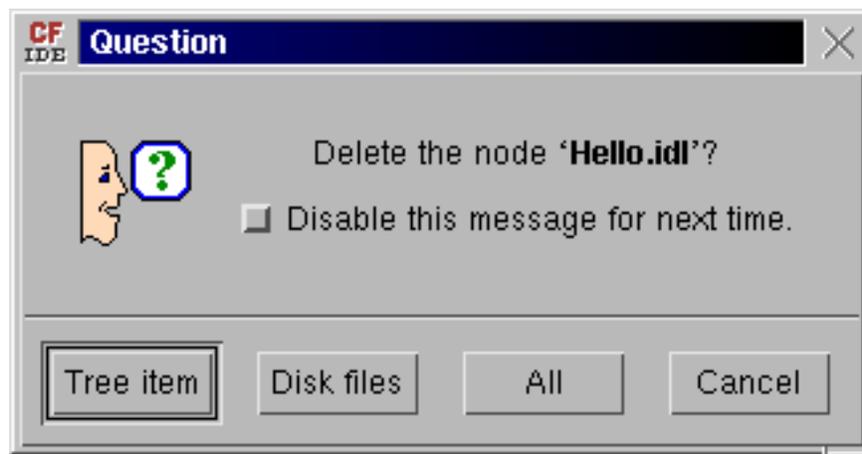
-Or-

Press **Delete** button.

-Or-

Drag a node or a group of nodes and drop it on the Remove node Drop Site:  .

C-Forge prompts you to select one of the four deleting modes:



3. Press one of the four buttons of the prompt dialog:

Tree Item - delete the tree node but keep the file on a disk

Disk Files - delete files on a disk but keep the tree node

All - delete both the tree node and the file on a disk

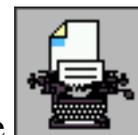
Cancel - cancel the delete operation.

If more than one file is to be deleted C-Forge displays the [Delete Files Dialog](#).

Editing Files

To start editing a file, drag it from the [Dependency Tree](#) and drop it onto the [Working Files Desktop](#).

-Or-



Drag the file from the [Dependency Tree](#) and drop it on the Edit node Drop Site

-Or-

Select **Edit Text** from the **Actions** menu.

-Or-

Click on the **Edit Node** Tool bar button.

-Or-

Double-click on the file in the [Working Files Desktop](#).

If the file was included in the [RCS](#), the latest revision is "fetched" into the working directory and becomes the working version; otherwise the stable version of a file is placed into the working directory.

C-Forge adds the icon of the file to the [Working Files Desktop](#) and opens an [Editor Window](#) with the working version of the file.

If the project was imported without the creation of [Working Directory](#) then no working versions are created and the stable versions of the files will be edited directly.

Double-clicking on the node in the [Dependency Tree](#) causes C-Forge to open the file for viewing (read-only mode). If modifications are attempted to a file in view mode, C-Forge will suggest that you switch to the edit mode.

For the description of editing see the [Editing Files](#) and the [Editor Window](#) help sections.

After editing a file, drag its icon from the [Working Files Desktop](#) back to the [Dependency Tree](#) to update the stable version. If the file was included into [RCS](#) then the [Check-In new revision](#) dialog will appear.

Debugging Files

To debug files click on the **Debug Target** button of the [Project Desktop](#) toolbar. This launches the debugger that was specified in [C-Forge Options](#).

Making Targets

To make a target, drag its icon from the [Dependency Tree](#) and drop it on the Make node Drop Site



-Or-

Select **Make Target** from the **Build** menu.

-Or-

Click on **Make Target** Tool bar button.

To make a node, select **Make Node** from the **Build** menu or click on the **Make Node** Tool bar button.

You can open [Properties](#) dialog (**Actions** Tab) by selecting **Properties -->Build Actions** from the **Actions** menu to edit target actions.

Running Targets

To run a target, select it in the [Dependency Tree](#) and choose **Run** from the **Build** menu

-Or-

Click on the **Run Target** button.

-Or-

Double-click on its icon in the [Dependency Tree](#).

If the target is not up-to-date, C-Forge will ask if the target should be built first.

C-Forge runs the target in the window specified by **Run Target** part of the [Project options](#) dialog.

Last modified: Wednesday, 27-Jun-2001 07:07:30 EDT

Revision Control System

- [Overview](#)
- [Including Files into RCS](#)
- [Excluding Files from RCS](#)
- [Fetching from the Repository](#)
- [Revision Control Options](#)
- [Checking-Out](#)
- [Checking-in a New Revision](#)
- [Fetching files](#)
- [RC log-window](#)
- [Version Tool](#)

Overview

Code Forge comes with a built-in **Revision Control System** that transparently keeps track of file modifications.

The current version of Code Forge performs automated revision control through RCS, CVS, SCCS, PRCs and Rerforce. RCS and SCCS work with a separate file on the local disk. CVS, PRCs and Perforce are able to trace changes within the whole project and allow to work with remote repositories.

Including Files into Revision Control System

When the project is created, the type of revision control can be chosen in the [New Project](#) dialog with the help of **Revision Control** drop-down combo-box.

When a file is added to a project, it is placed under revision control by checking **Revision Control** check-field in [Add File](#) dialog. For imported project, files are included into RCS as it is specified by **Revision Control** option in [Import Project](#) dialog.

To include an existing file into the repository select it in the [Dependency Tree](#) and then apply the **Create New** command of the **Revision-Control** menu. The file can be also added to the repository by checking the **Revision Control** check-box of the **Node Tab** (see [Node Properties dialog](#)).

Excluding Files from Revision Control System

The file can be excluded from the repository by releasing the **Revision Control** check-box of the **Node Tab** (see [Node Properties](#) dialog).

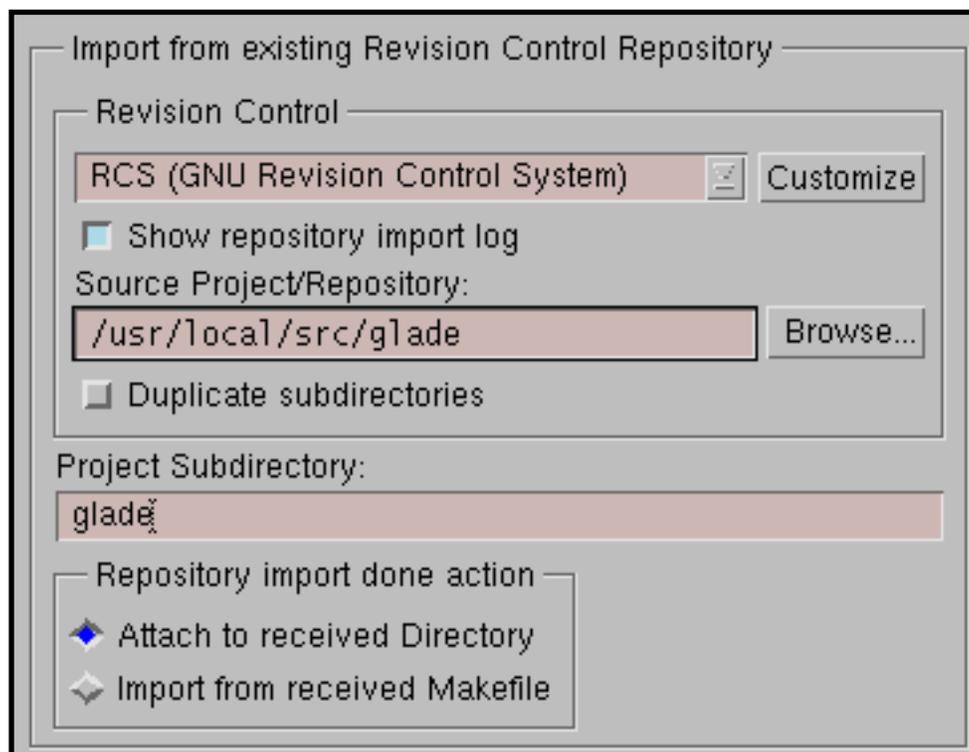
The file can be also excluded from the repository while deletion. To exclude the file choose **Disk entries** button of the **Question** dialog (appears when file is being deleted). After that the [Delete Files](#) dialog will appear. Choose **Repository entry for the node '..'** for the node you wish to exclude from the repository and press **OK** button of the dialog.

Fetching from the repository

An existing physical folder can be added to the project by fetching it from the repository. When a [new physical folder](#) is added to the project, the [Create New Folder](#) dialog appears on the screen. The revision control type of the folder being added should be chosen with the help of **Revision Control** combo-box of the [Create New Folder](#) dialog.

Import RCS Project dialog

If the RCS or SCCS revision control type has been chosen in the **Revision Control** field, the dialog will be self-modified and following fields will appear:



- **Source Project/Repository** - name of the repository or source project should be specified in this field. The path to the folder can also be selected by clicking the **Browse** button (which opens the

[Select Path](#) dialog).

- **Duplicate subdirectories** toggle button should be pressed if you wish to duplicate subdirectories, included into the repository, in your project.

Import CVS Project dialog

If the CVS revision control type has been chosen in the **Revision Control** field, following fields will appear:

Import from existing Revision Control Repository

Revision Control

CVS (Concurrent Version System) Customize

Show repository import log

Repository Access Method: pserver

User: Host:

Root: /usr/cvsroot

Module Name: C-Forge Browse...

Repository import done action

Attach to received Directory Import from received Makefile

Execute:

- **Repository Access Method** check-box allows to choose method of access to the repository
- **User** - if access to the remote repository is necessary, user name can be specified in this field
- **Host** - if access to the remote repository is necessary, host name can be specified in this field
- **Root** - if the local repository is used, repository root is specified in this field

Revision Control Options

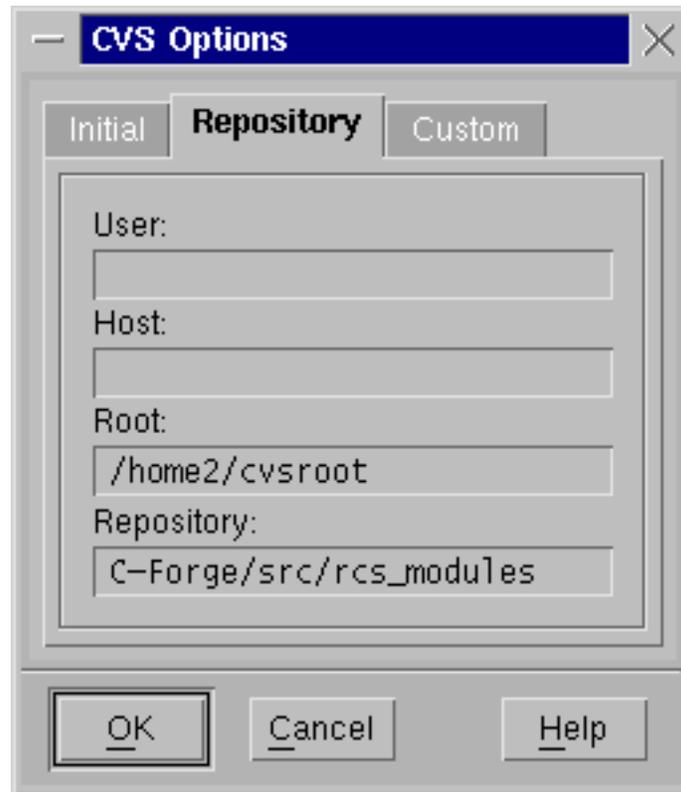
To view or edit revision control options, select **Options** from **Revision-Control** menu of the [Project Desktop](#). Those options are set while creating folder or importing file to the project. Depending on the type of revision control, CVS, RCS, SCCS or Perforce options window will be opened on the screen.

CVS Options

CVS Options window includes three tabs **Initial**, **Repository** and **Custom**.



Initial tab includes **New revision settings** option, where in the **Message** field the user can specify the comment with which new files will be added to CVS repository.



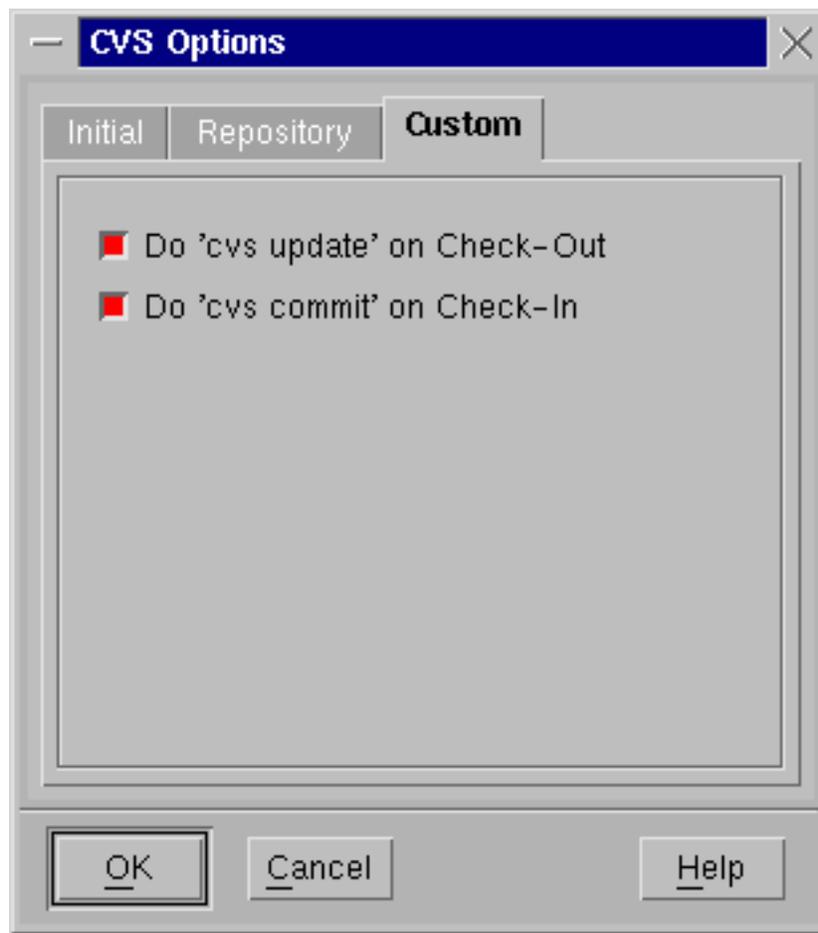
Repository tab is used only for viewing information.

User - displays user name if the remote repository is worked on

Host - displays host name if the remote repository is worked on

Root - displays repository root if the local repository is worked on

Repository - displays repository name.



Do 'cvs update' on Check-Out - if the check-box is marked, 'cvs update' command will be executed when **Check-Out** command is selected.

Do 'cvs commit' on Check-In - if the check-box is marked, 'cvs commit' command will be executed when **Check-In** command is selected.

Both options are switched on by default when the local repository is in question. When remote repository is worked on, both options are by default switched off, as sometimes it is not convenient that check-out and check-in are performed automatically when dragging the file from the **Dependency tree** area to the **Working area** or back.

Dialog buttons

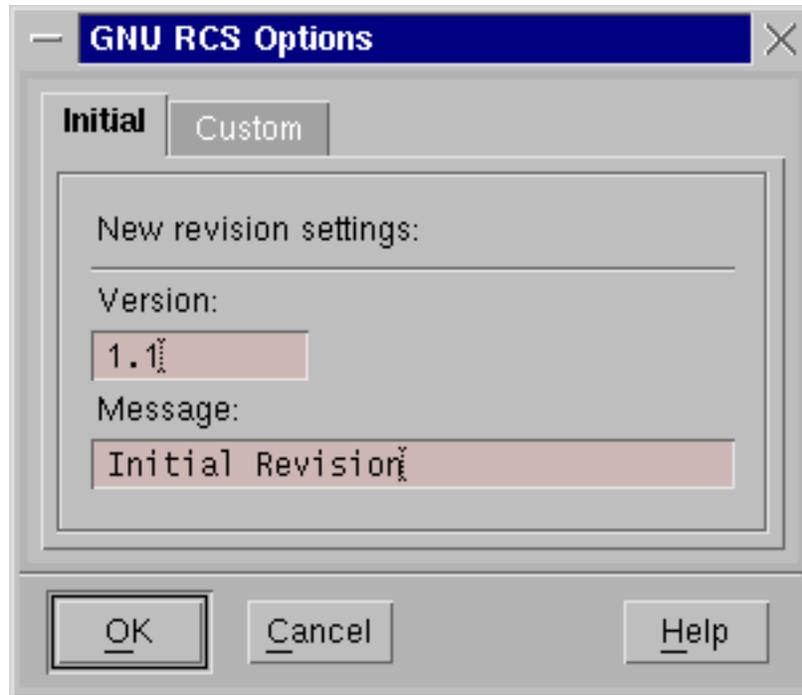
OK - closes **CVS Options** window and saves changes made.

Cancel - closes **CVS Options** window and cancels the changes made.

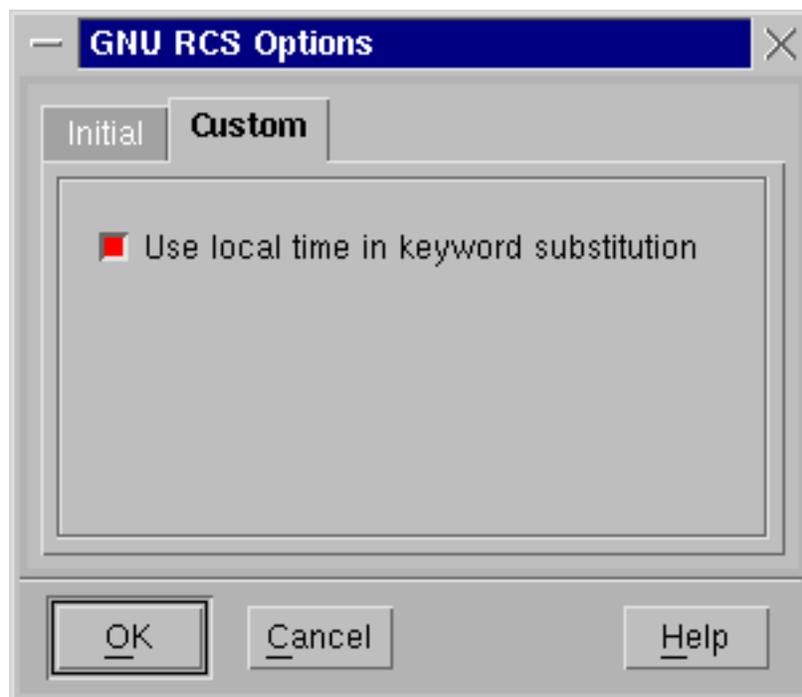
Help - shows this Help window.

GNU RCS Options

RCS Options window includes two tabs **Initial** and **Custom**.



Initial tab includes **New revision settings** option, where in the **Version** field file version number can be specified, and in the **Message** field the user can specify the comment with which new files will be added to repository.



Use local time in keyword substitution - if the check-box is marked, local time will be used by RCS while recording the revisions history. If the check-box is not marked, the Greenwich time will be used.

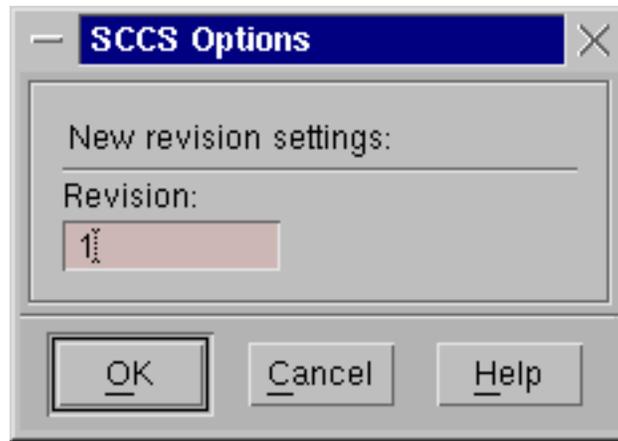
Dialog buttons

OK - closes **GNU RCS Options** window and saves changes made.

Cancel - closes **GNU RCS Options** window and cancels the changes made.

Help - shows this Help window.

SCCS Options



The **SCCS Options** window includes the **New revision settings** option, where in the **Revision** field file version number can be specified.

Dialog buttons

OK - closes **SCCS Options** window and saves changes made.

Cancel - closes **SCCS Options** window and cancels the changes made.

Help - shows this Help window.

Perforce Options

Perforce Options window includes two tabs **Initial** and **Custom**.



Initial tab includes the following fields:

New Client Name - new client name can be specified.

Host - host name can be specified.

Port - port number can be specified.



Do 'p4 sync' on Check-Out - if the check-box is marked, 'p4 sync' command will be executed when **Check-Out** command is selected.

Do 'p4 submit' on Check-In - if the check-box is marked, 'p4 submit' command will be executed when **Check-In** command is selected.

Both options are switched off by default.

Dialog buttons

OK - closes **Perforce Options** window and saves changes made.

Cancel - closes **Perforce Options** window and cancels the changes made.

Help - shows this Help window.

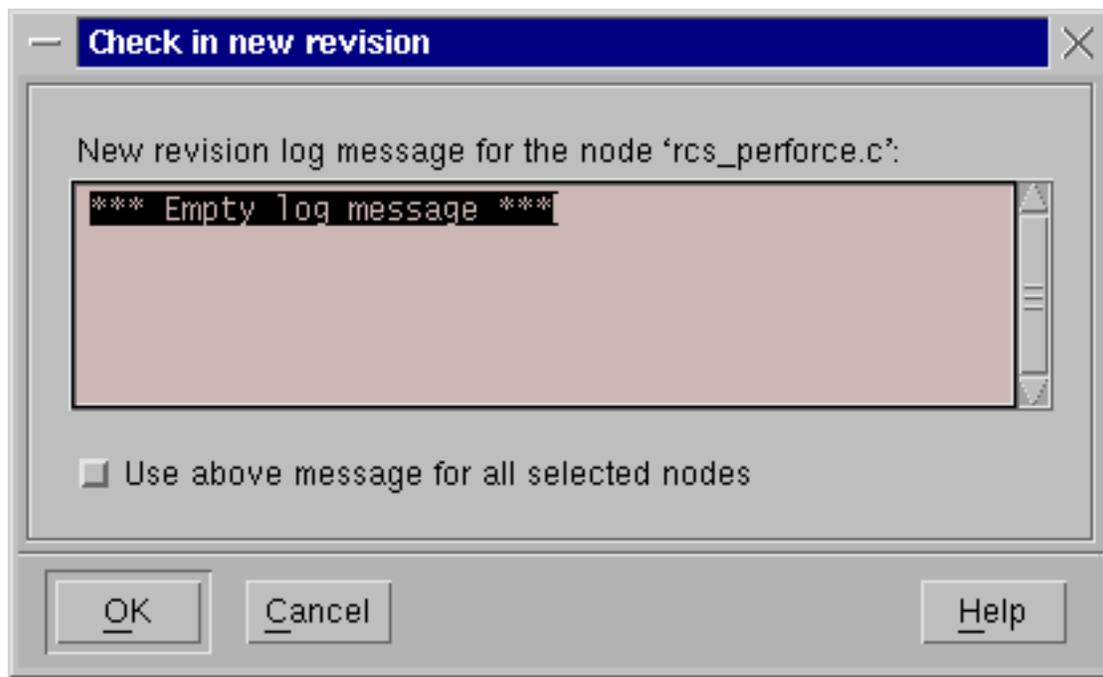
Checking-out

Checking-out will be done automatically when dragging a file from [Dependency Tree](#) to the [Working Files Desktop](#). "Check-out" can also be performed from the **Revision-Control** menu in the [Project Desktop](#).

Checking-in a New Revision

A new revision can be checked-in by dragging a file from the [Working Files Desktop](#) to the [Dependency Tree](#). "Check-in" can also be performed by selecting **Create New** command from the **Revision-Control** menu in the [Project Desktop](#). The checked-out file will be placed to the **Working area**.

The **Check-In New Revision** dialog is as follows:



The **Use above message for all selected nodes** check-box useful for multiple files "check-in".

Dialog buttons

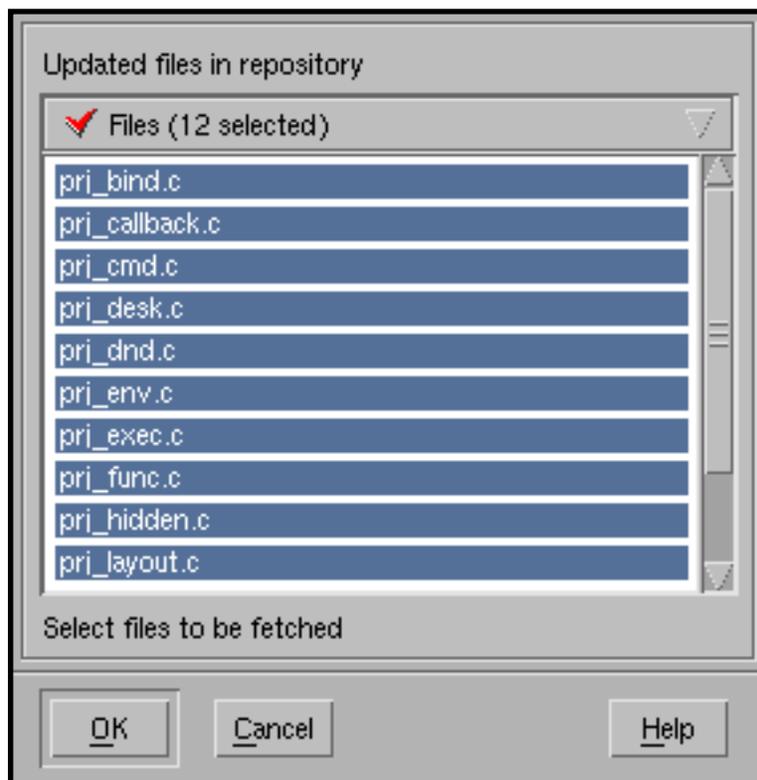
OK - checks-in a new revision.

Cancel - cancels the operation.

Help - shows this Help window.

Fetching files

When you are opening project or Physical Folder and the number of files in the Revision Control repository do not match the number of files in the user directory (sandbox) you will be prompted for fetching nonexistent files.



The same dialog is appear when you are manually pressing "Fetch sources" button in the [Revision Control](#) label of the [Node Properties](#) dialog.

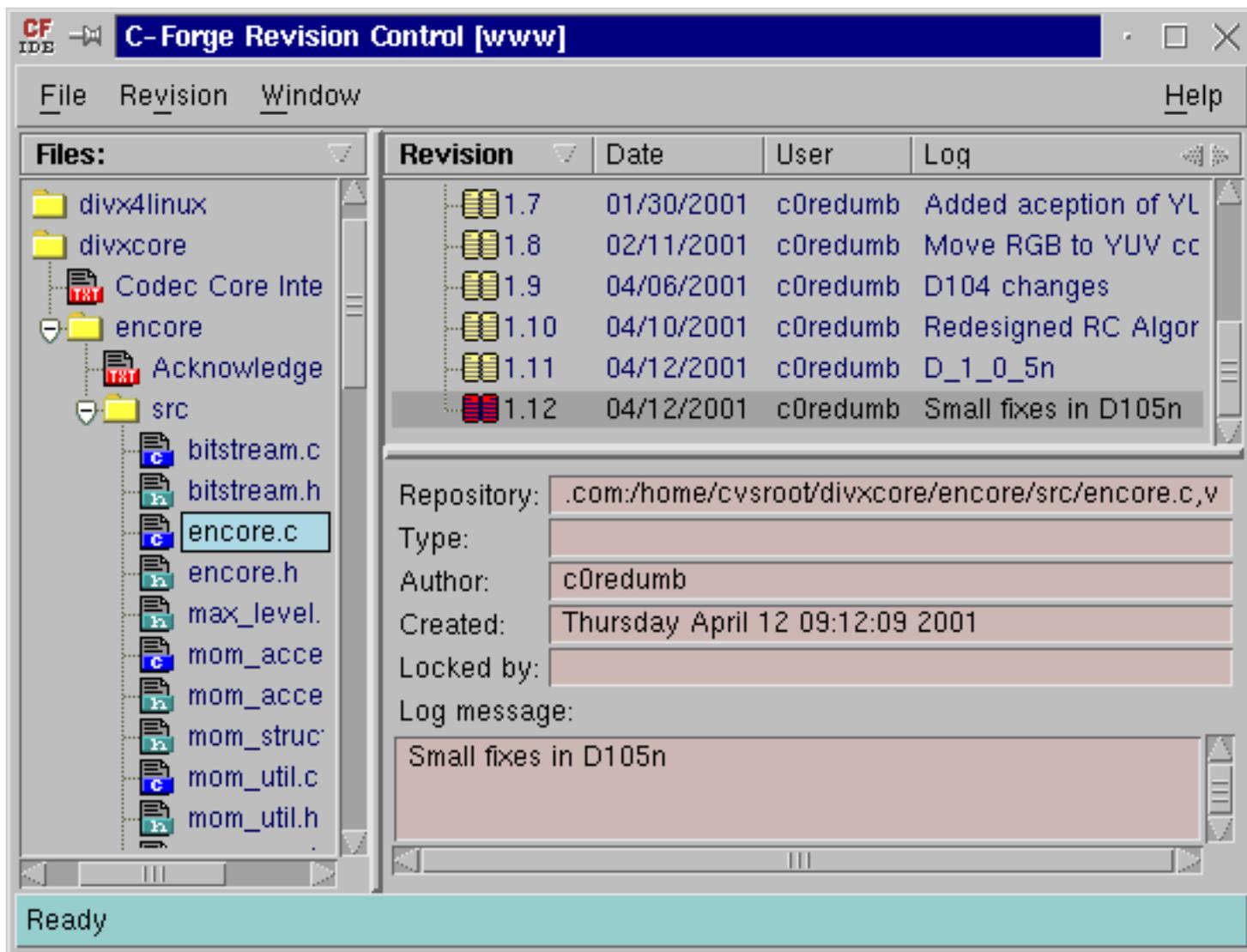
RC log-window

To display the revision control log-window press toggle button of the **Show Log** command from **Revision-Control** menu of the [Project Desktop](#). The opened log-window will include the history of revision control commands.

Version Tool

To display the Version Tree, drag and drop a file from either the Dependency Tree or the Working Area onto the Version Control System Drop Sites .

Code Forge displays the Revision Control window.



The left part of the Version Tool window displays the list of files included into repository for the current project represented as a dependency tree.

The right part of the window displays the **Version Tree** for the file selected in the left part of the window. The **Version Tree** is a standard element of [interface](#).

The default revision of the selected file is marked with red. This revision corresponds to the [stable version](#) of the file within the current project. Locked revisions are displayed with an icon of lock.

To launch the [Difference Tool](#) from the Version Tool window, drag the icon of one revision and drop it onto the icon of another revision. The Difference Tool will be launched for these two revisions. To load any other revision to the Difference Tool, drag its icon from the Version Tool window to the Name Drop site of the Difference Tool.

Detailed information about the currently selected revision is displayed below the **Version Tree**:

- **Repository** - specifies name and path to the repository.
- **Type** - specifies the type of revision control.
- **Author** - the author of the selected revision.
- **Created** - date and time when the selected revision was created.
- **Locked by** - the name of the user holding the latest lock on the revision.
- **Log Message** - log message for the selected revision.

Menu

File

- **Reload** - refreshes the Version Tree.
- **Close** - closes the Version Tree.

Revision

- **Set as default** - sets the selected revision as default.
- **Message log** - allows changing message log.
- **Lock** - locks the selected revision.
- **Unlock** - unlocks the selected revision.
- **View text** - shows contents of the selected revision in the editor.

Last modified: Friday, 15-Mar-2002 11:58:38 EST

Working With External Projects

- [Overview](#)
- [Methods of Using the External Project](#)
- [Revision Control](#)
- [Loading Tools options of the Imported Makefile](#)
- [Importing Projects Using Working Directory](#)

Overview

There are several methods when importing an existing project. An External Project can be imported from the repository, imported as a Makefile or attached as a physical folder. If a user wishes to join a project that is being worked on by several other users, the user should [import the project from the repository](#). Another method, if the user already has the project copied on his local disk, is to [import the project as a Makefile](#). Thus, nobody else except this user will be able to modify source files. If the project includes many subcatalogs then the best choice is the [Attach method](#).

To be imported or attached an external project should include a root directory. This directory should include a Makefile which consists of a collection of targets and their dependencies. The Project may also include source files and version control repositories.

Methods of Using the External Project

When it is necessary, that several users work with remote repository files within the same project, the best way is to import project from the repository. This will result in copying repository files to the local disk. (To find out how to apply this method, see [Import from revision control repository](#) section).

If there is already an existing Makefile, there are two methods of adding it to the project:

First is to convert Makefile into internal Code Forge project. The conversion will result in creation of the full-functional project as if it has been created under Code Forge. Let's call this method - **Import** (To find out how to apply it, see [Import method](#) section). But sometimes Makefiles are written (or generated) so awkward that such import may appear to be not very useful, as the user may have some troubles when searching source files. In such a case you would better use another method.

Second is to use Makefile "as it is". Thus, a small empty project is created under Code Forge and external root directory is attached to it as a [physical folder](#). When using this method the user will not be

able to change the project with the help of GUI Makefile editor, the division to Stable and Working versions will be lost, and it will be impossible to use **Tools Options** (**Options** menu). Let's call this method - **Attach** (To find out how to apply it, see [Attach method](#) section).

For both methods you have to invoke [Import Project dialog](#).

Revision Control

By default any editable file imported into Code Forge will be placed under revision control. (If the catalog where the file in question is imported to is not subjected to the revision control, the file will not be placed under revision control either). While importing process Code Forge will try to recognize the type of revision control to which imported project (files) are subjected, and will offer using the same type of revision control (by default). If Code Forge fails while trying to verify revision control type, the user should specify it.

Loading Tools options of the Imported Makefile

If the MakeFile of the imported project contains macros with tools options, Code Forge includes these options into [Tools Options](#) dialog. This dialog displays the macros that can be recognized by Code Forge in **Symbolic Named Options** field of every tool, for example in *CFLAGS* for C compiler.

If the imported makefile includes Macros that are not recognized by Code Forge, then [Tools Options](#) dialog can miss some tools options. To avoid this situation, we recommend editing the Makefile before importing. It is needed to assign the macros of Makefile to the macros that are recognized by Code Forge, and then to replace macros in the target actions with macros that are recognized with Code Forge.

For example, the following Makefile:

```
GCCFLAGS = -g -ansi
```

```
LDFLAGS = -static
```

```
LD = gcc
```

```
CC = gcc
```

```
.c.o:
```

```
$(CC) $(GCCFLAGS) -o $@ $<
```

```
prog: main.o file.o
```

```
$(LD) $(LDFLAGS) -o prog main.o file.o
```

should be edited as follows:

```
GCCFLAGS = -g -ansi
```

```
LDFLAGS = -static
```

```
LD = gcc
```

```
CC = gcc
```

```
# CFLAGS is recognized by Code Forge as C-compiler options macro
```

```
CFLAGS = $(GCCFLAGS)
```

```
# LDOPTIONS is recognized by Code Forge as linker options macro
```

```
LDOPTIONS = $(LDFLAGS)
```

```
.c.o:
```

```
$(CC) $(CFLAGS) -o $@ $<
```

```
#change file names to automatic make variables
```

```
prog: main.o file.o
```

```
$(LD) $(LDOPTIONS) -o $@ $^
```

Importing Projects Using Working Directory

The project imported with [Use Stable/Work layout](#) option includes the [Stable and Working directories](#). The Working Directory of the current project becomes the current directory for the system where the working Makefile is generated. This working Makefile includes the absolute names of targets and dependency files.

The generation of working Makefile in the Working directory of a user allows to isolate working areas of different users. But it contradicts the general usage of Makefile in the same directory where the project files are located, when relational paths of files can be used.

So, to avoid the situation when Make can not find a file with a relational path while importing, it is recommended to edit target actions and replace file names with [Make Auto Variables](#) before importing of a project.

Project Desktop

- [Overview](#)
- [Project Views](#)
 - [Dependency Tree](#)
 - [Drag'n'Drop Operations](#)
 - [Working Area](#)
 - [Disk view](#)
- [Menu Bar](#)
- [Tool Bar](#)
- [Drop Site Bar Frames](#)
- [Status Bar](#)

Overview

The Project Desktop provides a visual representation of a project. It includes **Menu Bar**, **Tool Bar**, one or more **Project Views**, **Drop Site Bar** and the **Status Bar**.



Project View

The **Project View** area contains three folders with different types of subview. It is now easy to reorganize the look of your project using a few mouse clicks !

Dependency Tree

The **Dependency Tree** folder contains a hierarchical list of project components. It is the standard element of [interface](#). The complete project structure and status of each item can be seen from this list.

Dependency Tree always includes at least two folders:

- **Root Folder** - for all [folders](#), [macros](#) and [targets](#) of this project.
- **Includes Folder** - obligatory folder for the alphabetically list of the all included files.

An any file in this tree can be found by selecting **Find** in the **Actions** menu or by pressing **Ctrl+F** shortcut keys. This will bring up the [Find Node](#) dialog.

In a huge projects, where is a lot of different targets and macros it is very useful to use **Virtual Folders**.

Drag'n'Drop Operations

Note: All Drag'n'Drop operations in **C-Forge** are performed by [Mouse Button 2](#) (Middle mouse button).

Dependency Tree management operations:

- **Rearrange** nodes:
To change any node position at one tree level, Drag'n'Drop (**Btn2**) any node to parent node or to corresponding position at the same level.
- **Move** nodes:
To change the target or folder node parent, **Shift+Btn2** Drag'n'Drop it to new folder.
- **Link** nodes:

To link (copy) any node to other, **Ctrl+Shift+Btn2** Drag'n'Drop it to new target node.

Note: The deletion of linked nodes (except folders) is not removes the source node of link.

Working Area (Working Files Desktop)

The **Working Area** (also known as **Working Files Desktop**) contains a list of files, sorted by name, that are presently being modified. Files are added or removed from this list using Drag'n'Drop operations to [Drop Sites](#) or corresponding elements of popup menus. When a file is placed in the **Working Area** it is locked and no other users can modify it.

Pressing **Enter** on a selected file node or double-clicking on it will open the file in the [Editor Window](#).

Disk

The **Disk** panel contains a view on the file system.

Using Drag'n'Drop it is possible to simply add sources and includes to **Dependency Tree** and manipulate files on disk.

Note: dragging a file(s) from this view to the target icon in the **Dependency Tree** opens the [Add Node\(s\)](#) dialog.

Popup menu:

- **Create** - creates an object in currently selected directory:
 - **Project Source** - creates new file and adds it as source to current target.
 - **Directory** - creates new directory.
 - **File** - creates new file.
- **Update** - updates this **Disk** view.
- **Brief/Full info** - sets the Brief/Full type of files list viewing.
- **Filter** - sets the filter for files.
- **Orientation** - changes the placement (vertical/horizontal) of directory tree.
- **Select All** - sets selected disk entries as selected.
- **Edit** - edits selected disk entries.
- **View** - views selected disk entries.
- **Print** - prints selected disk entries.
- **Touch** - touches selected disk entries.
- **Rename** - renames selected disk entry.
- **Remove** - removes selected disk entries.
- **Terminal** - launches the terminal emulator in currently selected directory.
- **Properties** - opens [Node Properties](#) dialog.

It is possible to apply a command to multiple files by selecting them via box-select and/or click/shift-click/ctrl-click.

Menu Bar

File

- **Save** - writes the changes made in the project to a disk.
- **Export** - allows exporting project to makefile.
- **Merge** - merges changes of previously exported project makefile.

- **Info** - calls up [Project Information](#) dialog.
- **Hide Project** - all windows of current project minimizes into one icon.
- **Close** - closes the project.

View

- **Sort** - sorts all items of **Dependency Tree**.
- **Small/Large Icons** - changes size of icons in **Dependency Tree**. Small icons are very useful for low-resolution display modes.
- **Add new view** - adding new **Project View**.
- **Remove current view** - removes current **Project View**.
- **Change orientation** - changes orientation (horizontal/vertical) of all **Project Views**.
- **Split Dependency Tree** - splits the current **Dependency Tree**.
- **Unsplit Dependency Tree** - unsplit previous split **Dependency Tree**.
- **Brief/Full info** - changes the view mode of **Disk Tree**.
- **Disk Filter** - sets a new filter for **Disk Tree**.
- **Disk Panels Orientation** - changes the placement (vertical/horizontal) of directory tree for **Disk Tree**.
- **Select All** - sets selected nodes or disk entries as selected.

Actions

- **New**
 - **Folder** - opens the [New Folder](#) dialog to add a new [folder](#).
 - **Target** - opens the [Create Target\(s\)](#) dialog to add a new targets.
 - **Macro** - opens [Single Value Input Dialog](#) to add a new macro.
 - **Source** - opens the [Add File to Project](#) dialog to add a new [source file](#).
- **Remove**
 - **Remove Node** - [deletes](#) the node and its files; if more then one file will be deleted, C-Forge displays the [Delete Files](#) dialog.
 - **Remove Disk Dependency(s)** - deletes the objects and target files (if exists) for selected node; if one or more files will be deleted, C-Forge displays the [Delete Files](#) dialog.
- **Touch** - changes the modification time of the node file to the current time.
- **Find** - opens the [Find](#) dialog that allows searching nodes in the Dependency tree.
- **Edit Text** - opens the file in an [Editor Window](#) in read/write mode.
- **View Text** - opens the file in an editor window in read-only mode.
- **Print Text** - prints the selected file to printer.
- **Diff/Merge Tool** - launches the [Diff/Merge](#) tool.
- **Search/Replace** - opens the [Search/Replace](#) dialog.
- **Symbol Navigator** - opens the [Symbol Navigator](#) dialog.
- **Terminal** - launches the terminal emulator, the type of the being specified in [Project Options](#)

(**Helpers** Tab, **Terminal emulator** field). If the type of the terminal is not specified, the type of the terminal will correspond to the value of **TERM** environment variable.

- Stable Root - launches the terminal emulator in the Stable Root.
- Working Root - launches the terminal emulator in the Working Root.
- Stable Node - launches the terminal emulator in the Stable Node.
- Working Node - launches the terminal emulator in the Working Node.
- **Properties** - opens [Node Properties](#) dialog with specified folder:
 - **Node**
 - opens [Node Tab](#) or [Folder Tab](#) of the [Node Properties](#) dialog depending on the type of the node selected in the **Project View** area.
 - **Build Actions** - opens [Actions Tab](#) of the [Node Properties](#) dialog. Helps to specify actions used to build current node or target (activated by **Make Node** and **Make Target** actions).
 - **Exec Actions** - opens [Actions Tab](#) of the [Node Properties](#) dialog. Helps to specify actions, activated by double-click on current node.
 - **Debug Actions** - opens [Actions Tab](#) of the [Node Properties](#) dialog. Helps to specify actions used to debug current node (activated by **Debug** action).
 - **Disk Entry**
 - opens [Disk Entry Tab](#) of the [Node Properties](#) dialog.
 - **Revision Control**
 - opens [Revision Control Tab](#) of the [Node Properties](#) dialog.
- **Collect All Includes** - collects all includes and determines the implicit dependencies.
- **Collect Includes** - collects includes and determines the implicit dependencies for the current node.
- **Collect All Tags** - collects all tags for all project files.
- **Rescan Folder for New** - rescans folder searching for new nodes.
- **Rescan Folder for All** - rescans folder searching for any changes made.

Build

- **Build Options** - opens the [Options](#) dialog.
- **Make Node** - makes current node (actions are depends from type of node):
 - Current node is target - equal to **Make Target** action.
 - Current node is source (have hidden dependencies) - execute default conversion for current node.
- **Make Target** - makes current target (actions are depends from placemen of node).
 - Current node is target - makes all hidden conversions and executes **Build Actions** of this node,
 - Current node is placed in the dependency list of target - makes target of current node.
- **Make clean** - executes 'make clean' command for selected nodes.
- **Rebuild** - executes 'clean' and 'make' actions for selected nodes.
- **Run** - Executes the default actions for current node (specified by **Exec Actions**).
- **Debug** - executes the **Debug Actions** for current target or node.

Revision-Control

Revision-Control menu items are differs depending on the type of the revision control to which the project is subjected. If the project is not subjected to revision control, this menu will not be displayed at all. For the **RCS** and **SCCS** types **Update All, Commit All, Login, Logout** commands will not be displayed, as those types of revision control don't allow to work with projects and remote repositories.

- Options - depending on the type of revision control to which the project is subjected, opens **Options** dialog. For CVS revision control type it opens [CVS Options](#) dialog window. For RCS type it opens [GNU RCS Options](#) dialog window.
- Show Log - shows or hides additional log-area, including the history of revision control commands given.
- Create New - adds selected file to the repository.
- Check-Out - performs check out of the selected node.
- Check-In - performs check in of the selected node.
- Fetch Source - deletes file and fetches its default revision from the repository. All the changes made in the file are cancelled while fetching.
- Version Tool - opens [C-Forge Revision Control](#) window.
- Properties - opens [Revision Control Tab](#) of the [Node Properties](#) dialog.

The next part of the menu is dynamic and appears or not depending on the type of the revision control, which the selected node is subjected to.

- Update All - updates all changes made in the project.
- Commit All - commits all changes made in the project.
- Login - opens the **Login** dialog where the password for logging in to the remote repository can be entered.
- Logout - terminates session with remote repository.

Options

- **Project options** - opens the [Project Options](#) dialog.
- **Tools options** - opens the [Tools Options](#) dialog.
- **Tools Binding** - opens the [Tools Binding](#) dialog.
- **Environment** - opens the [Environment](#) dialog. All environment variables specified in this dialog, are visible only in current project.
- **Macros** - opens the [Project Macros](#) dialog.
- **Conversions** - opens the [Project Conversions](#) dialog.

Log

This is the menu of [Log-window](#).

Tool Bar

The project desktop tool bar contains the following buttons:



- Save Project - writes the changes made in the project to a disk.
- Add Folder - opens the [New Folder](#) dialog to add a new [folder](#).
- Add Targets - opens the [Create New Target\(s\)](#) dialog to add a new targets.
- Add Macro - opens the [Single Value Input](#) dialog to add a new [macro](#).
- Add Sources - opens the [Add File to Project](#) dialog to add a new source and include files.
- Remove Node - [deletes](#) the node and its files; if more then one file will be deleted, C-Forge displays the [Delete Files](#) dialog.
- Edit Node - opens an [Editor Window](#) to edit the node file.
- Run Target - runs current target.
- Debug Target - executes the command, specified in the [Debug Actions](#) of current target.
- Make Node - makes current node.
- Make Target - makes current target.
- Tools Options - opens the [Tools Options](#) window.

The Toolbar can be toggled on and off using the [Project Options](#) dialog.

Drop Site Bar Frame

The Drop Site Frame is a group of icons, on the right side of the Project Desktop, used to perform C-Forge commands. To use a Drop Site drag a node and drop it into the icon matching the desired operation. The same functionality is duplicated in menu structure. The following is a list of the Drop Sites:



- Version Tool Drop Site - launches the [Revision Control Tool](#) and displays dropped node Revision control information. If dropped node is not under the RC, but it can be places there - works like Revision-Control/Create New.



- **Remove Drop Site** - deletes the nodes (for nodes, dragged from Dependency Tree or Disk) or discards changes (for nodes from Working Area).



- **Touch Drop Site** - sets the time of last modification of the dragged nodes to current time.



- **Edit Drop Site** - opens an [Editor Window](#) for all editable nodes.



- **Make Drop Site** - performs the Make Node action on each dropped node. If node is folder - makes all targets in this folder.



- **Information Drop Site** - opens the [Node Information](#) dialog first dropped node.

Status Bar

The Status Bar is used to display the type and status of the current node and project.



The busy lamp indicates the busy state of the current project during lengthy operations.

The first icon on the right hand side of the status bar indicates the editing state of selected node in the Dependency Tree.

- [empty] - not loaded into the editor.

-  - loaded and unmodified.
-  - loaded and modified.

Clicking on this icon closes selected file in the editor.

The next site -  is displayed when [RCS](#) deposits are available for the currently selected node.

And at the last site placed the editor launch policy indicator.

-  - open new files in current window.
-  - open new files in separate (new) windows.

Clicking on the icon toggles the policy state.

As with the Tool Bar, the Status bar can be turned on/off using the Show Status Bar option found in the [Project Options](#) dialog.

Last modified: Friday, 15-Mar-2002 11:58:38 EST

New Folder/Directory Dialog

The **New Folder dialog** is used to add a new node into [Dependency Tree](#) of a project. This dialog is called by pressing the **Add folder** button found in the [Actions](#) menu of the **Project Desktop**.

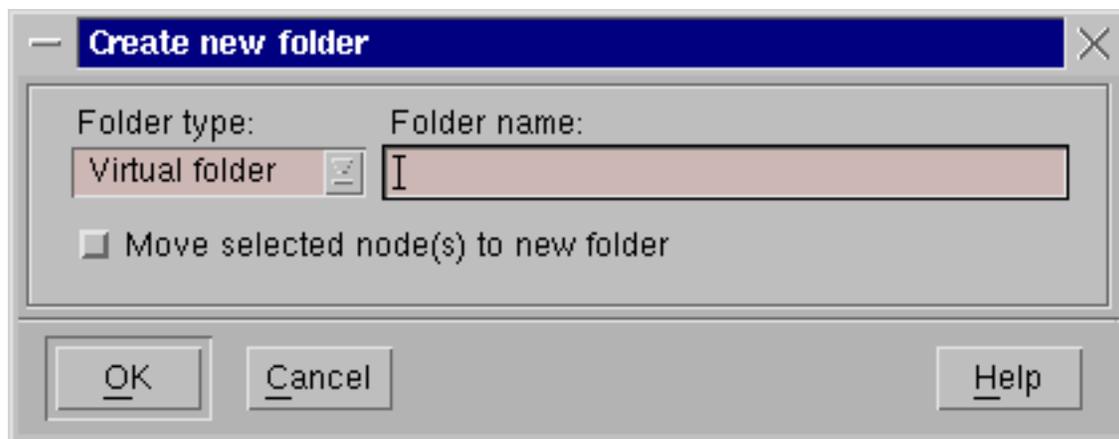
New Directory

When a [physical folder](#) is selected in the [Dependency Tree](#) of a project, the **New folder dialog** will be the following:



After filling in the **Enter the name of new directory** with the name of the new folder and pressing the **OK** button of the **Create new directory** window, the new physical folder will be created as a subdirectory of the selected folder.

New Folder



- With the help of **Folder Type** combo-box the type of a new folder can be specified ([Virtual Folder](#) or [Physical Folder](#)). **Virtual Folder** value is set by default.

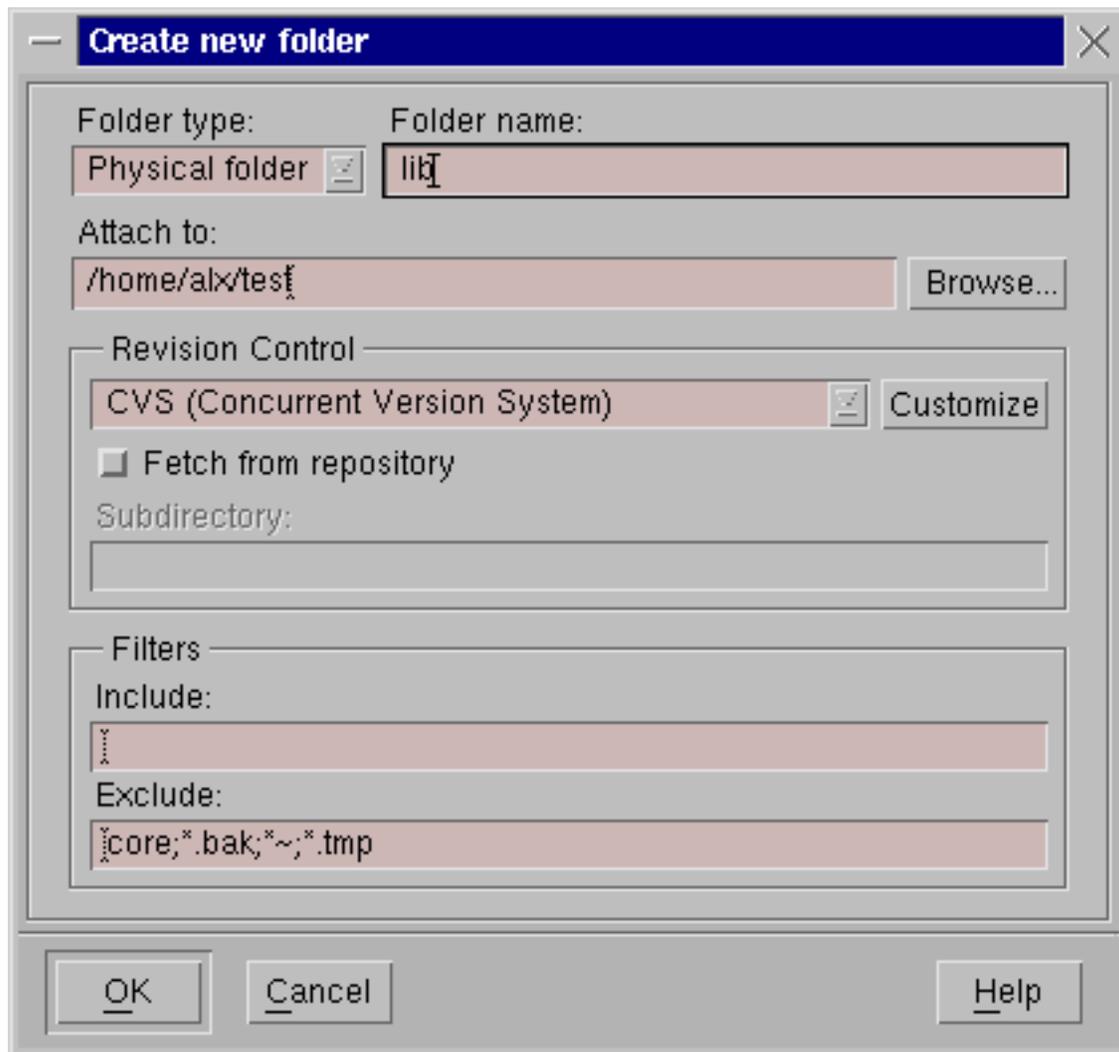
- **Folder Label** field specifies the name of new folder.

Virtual Folder

When **Virtual Folder** is being created, press **Move selected node(s) to new folder** toggle button, if you wish to move all currently selected nodes into this new virtual folder. After specifying the name of the folder and pressing **OK** button creation of virtual folder is completed.

Physical Folder

As there is still no opportunity to create physical folder by choosing an appropriate menu item, creation of the physical folder should start with the [New Folder](#) dialog. Where the **Folder type** should be specified as **Physical Folder**. When **Physical Folder** is being created, the following dialog window will appear on the screen:



- **Folder Path** - specifies the path to the folder. The path to the folder can also be selected by

clicking the **Browse** button (which opens the [Select Path](#) dialog). When **Browse** button is used for path selection, the **Revision Control** field will be automatically filled in with the revision control type of the selected folder. The user can change it at any time. If files are fetched from the repository, physical folder (its name is specified in the **Folder Label** field) will be created in the selected catalog. Fetched files will be copied to this folder.

- With the help of **Revision Control** combo-box [Revision Control](#) type can be specified. Pressing **Customize** button right to the combo-box will call the [Revision Control Options](#) dialog, which helps to specify or change some of revision control options.
- **Filters** - fields **Include** and **Exclude** specify filter, using *sh* wild card pattern and thus indicating files that should be included or excluded while viewing the [Dependency Tree](#).
- **Fetch from the repository** - press this toggle button if you need to fetch files from the repository. When the button is pressed, some additional fields depending on the type of the chosen revision control will be added to the dialog.
- If CVS revision control type is chosen, the following fields will be added to the dialog:

— Create new folder

Folder Type: Physical folder

Folder Label: smed

Folder Path: /home/alx/projects/test Browse...

Revision Control

CVS (Concurrent Version System) Customize

Attach to Revision Control Repository

Repository Access Method: pserver

User: Host:

Root: /home2/cvsroot

Module Name: C-Forge/smed-mbu

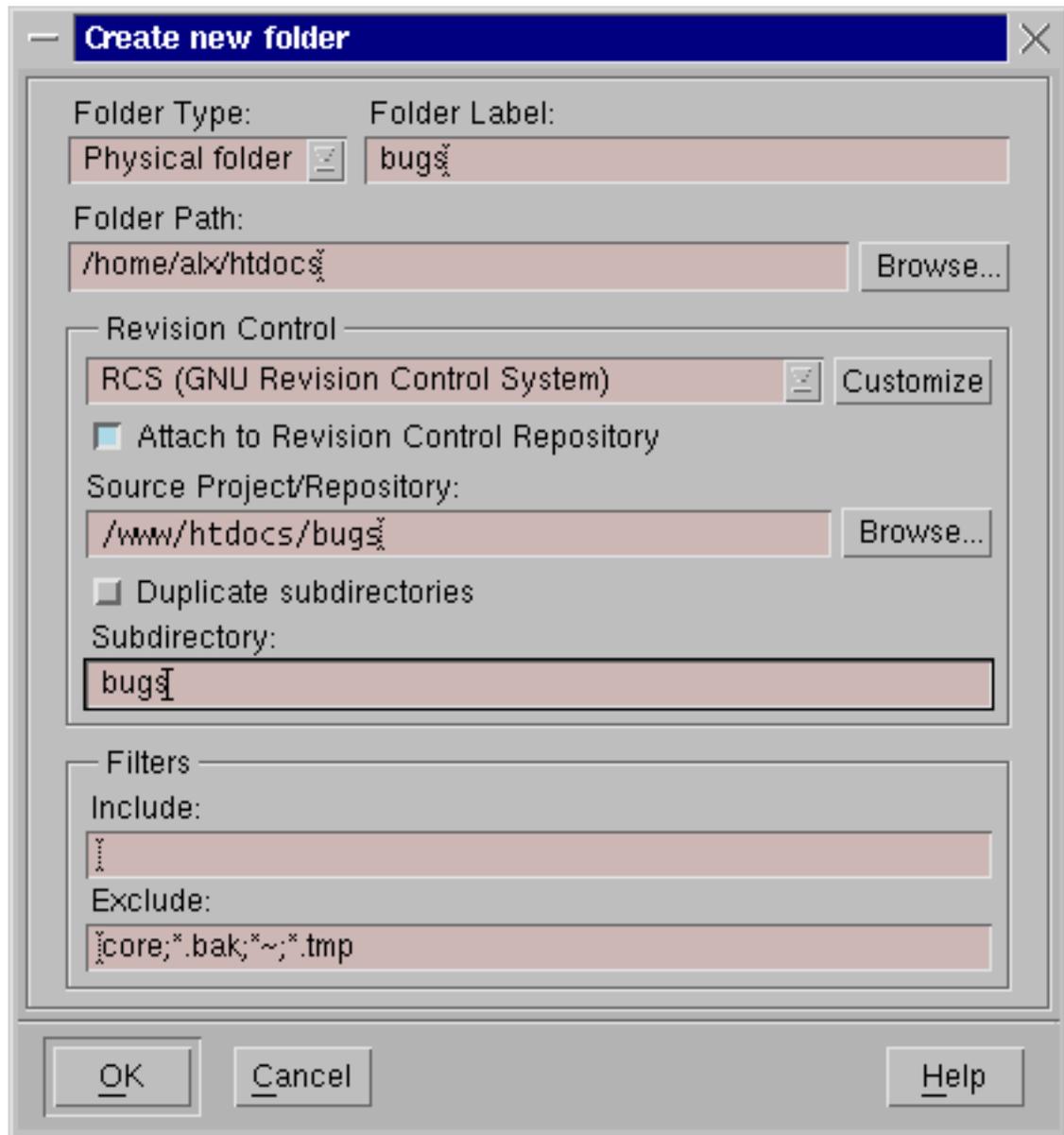
Filters

Include:

Exclude: core;*.bak;*~;*.tmp

OK Cancel Help

- **Repository Access Method** - method of access to the repository can be specified
- **User** - user name can be specified if the remote repository is worked on
- **Host** - host name can be specified if the remote repository is worked on
- **Root** - repository root can be specified if the local repository is worked on
- **Module name** - the name of the module being saved into CVS can be specified.
- If RCS or SCCS type of revision control has been chosen, the following fields will be added to the dialog:



- **Source Project/Repository** - name of the repository or source project should be specified in this field. The path to the folder can also be selected by clicking the Browse button (which opens the [Select Path](#) dialog).
- **Duplicate subdirectories** toggle button should be pressed if you wish to duplicate subdirectories, included into the repository, in your project.
- **Project Subdirectory** - specify the name of the catalog, where the files from the

repository will be copied.

Fetching and importing can be performed only if the chosen revision control type allows such operations.

New folder is created in the dependency list of currently active folder.

Dialog Buttons

OK - adds a new folder into the Dependency Tree.

Cancel - closes the dialog without adding a new folder.

Help - shows this Help window.

Last modified: Wednesday, 06-Jun-2001 10:36:07 EDT

Add Files to Project Dialog

- [Overview](#)
- [Create New Target\(s\)](#)
- [Create New Node\(s\)](#)
- [Add Target\(s\)](#)
- [Add Node\(s\)](#)

Overview

Add File to Project dialog can be used to add files to a project, when:

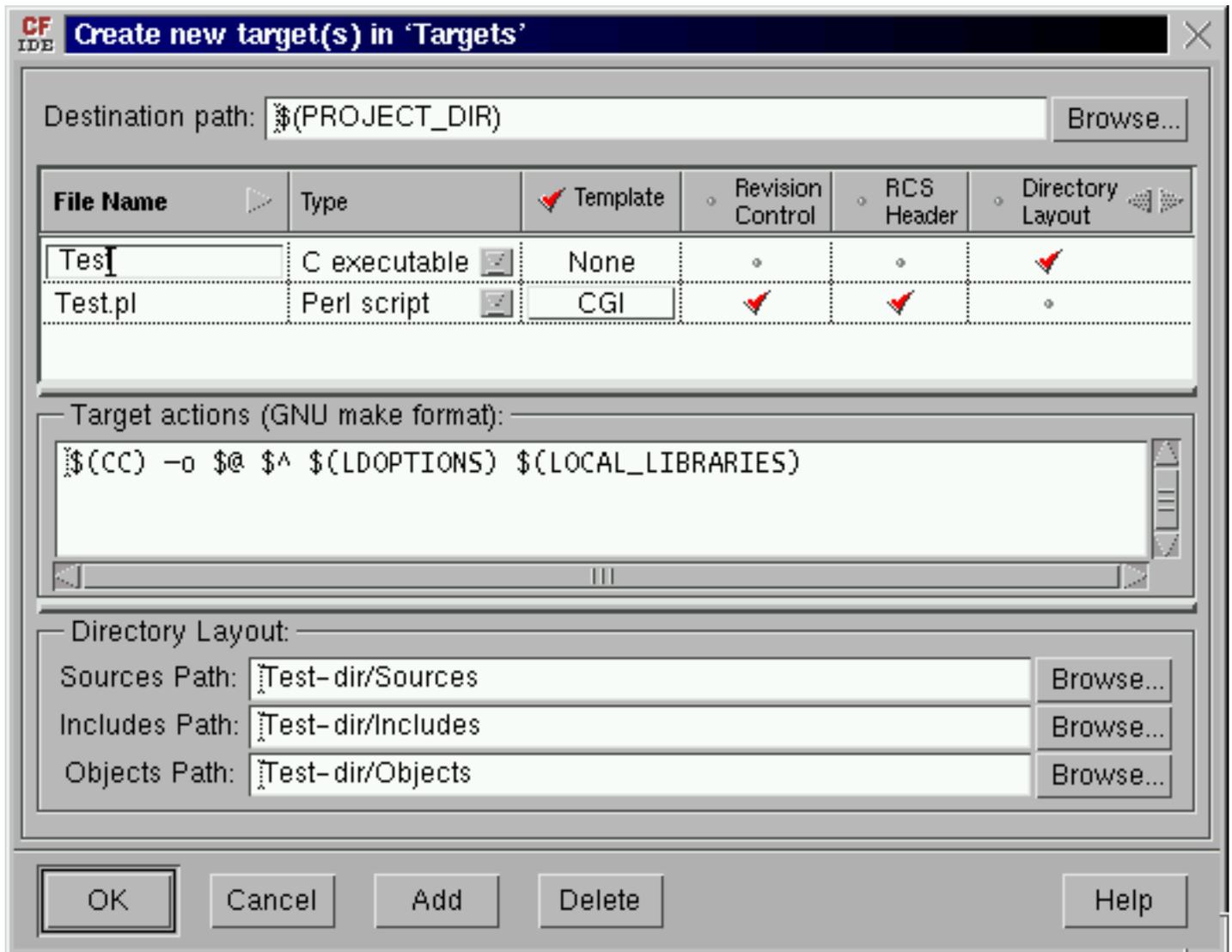
- A new target/script is created in the project (initiated by pressing the  button on the [Project Desktop toolbar](#))
- A new source/include file is created in the project (initiated by pressing the  button on the [Project Desktop toolbar](#))
- Importing existing targets/scripts (initiated by dragging them from the [Disk panel](#) onto one of the folders in the [Dependency Tree](#))
- Importing source/include files (initiated by dragging them from the [Disk panel](#) onto one of the folders in the [Dependency Tree](#))

The dialog appears slightly different in each instance; however, they all have the following common elements:

- **Destination path** - path where the new files will be placed.
- List of nodes being added to the project:
 - **File Name** - name of file being added. It will be created in the directory specified by the Destination path. (The file name can contain a relative path).
 - **Type** - the type of file that is being added. This will be used to determine what tools to use when working with this particular node.
 - **Revision Control** - determines if revision control will be used with this particular node.
 - **RC header** - determines if an RC header will be inserted into the file.
 - Other fields specific to the node type.

Create New Target(s)

This dialog is used to add new targets/scripts to the project:

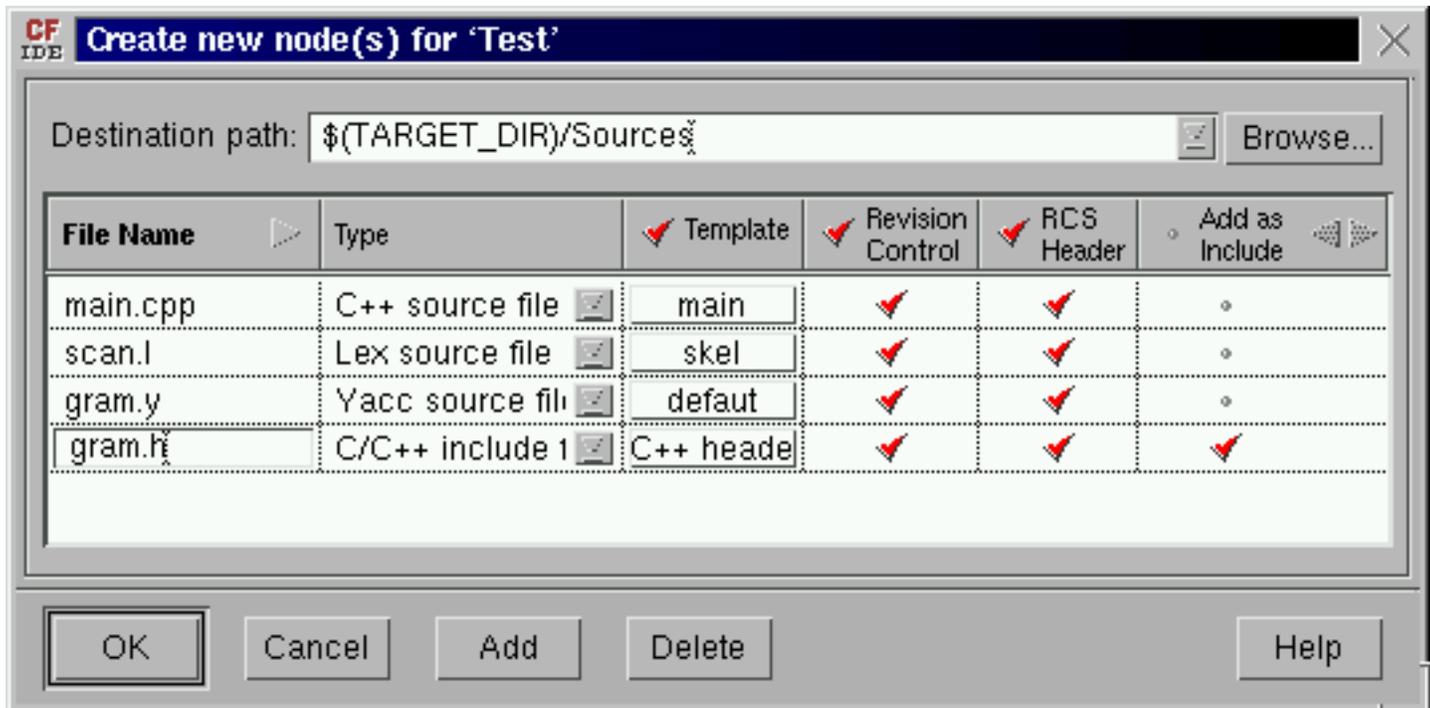


Additional Fields:

- **Target Actions (GNU make format)** - a command that will be used to build the target.
- **Directory Layout** - specifies where target files will reside. Controlled by the "Directory Layout" check field in the target list.
- **Template** - specifies template from which will be created source file. Clicking on push button on that cell opens [Templates](#) dialog

Create New Node(s)

This dialog is used to create new source/include nodes and to add them to the target currently selected in the [Dependency Tree](#).



Additional Fields:

- **Add as Include** - indicates that the file will be added as an include file into the special **Includes** folder.

Types that have predefined conversion rules for the current target are placed in the beginning of the **Type** list.

Add Target(s)

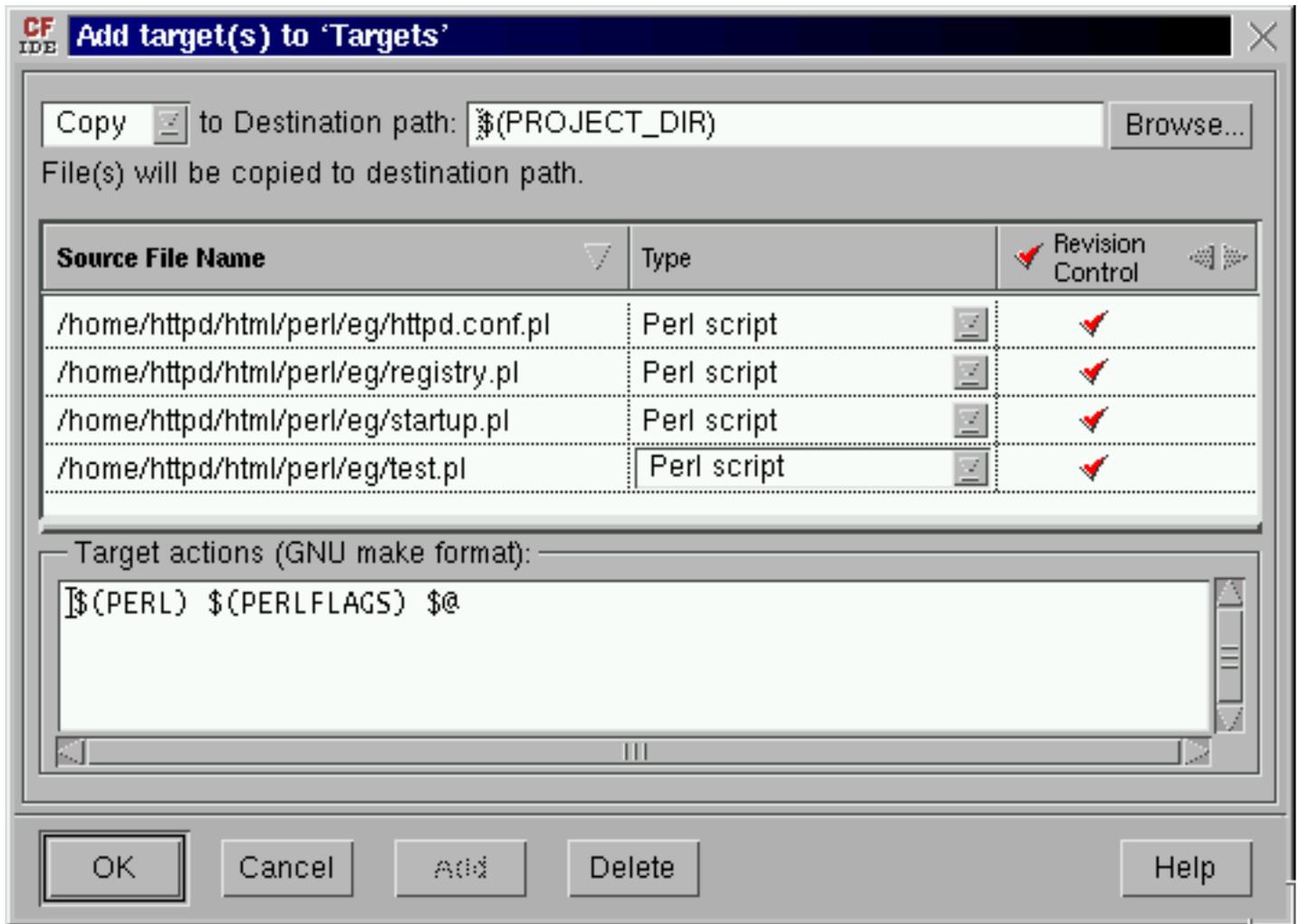
This dialog is used to add (import) already existing targets/scripts to the project..

This action is initiated by dragging one or more targets/scripts from the [Disk panel](#) into a folder in the [Dependency Tree](#).

The following drag and drop modifiers can be used for this operation:

Keys	Operation

-	Move nodes
Ctrl	Copy nodes
Ctrl-Shift	Link nodes



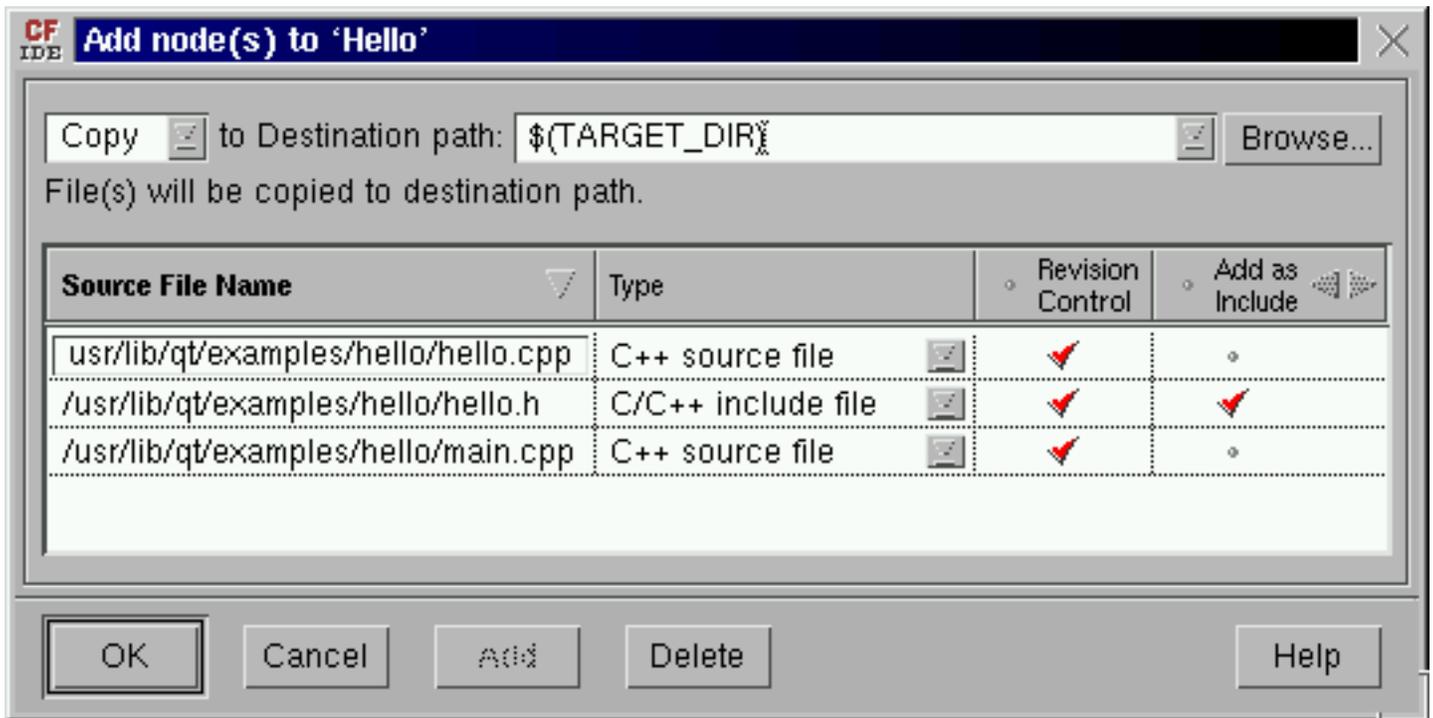
Operation type (Move, Copy, Link) can be changed at any time by selecting it in the listbox at the top left corner of the dialog.

- Move - files will be copied to the **Destination Path** and then removed from their original location.
- Copy - files will be copied to the **Destination Path**.
- Link - files will be used from their original locations.

Add Node(s)

This dialog is used to add (import) already existing source/include files to the project.

This action can be initiated by dragging files from the [Disk](#) panel onto a target in the [Dependency Tree](#). The imported files will be added to the dependency list of the currently selected target.



[Operation type](#) can be changed the same way as when adding targets.

Dialog Buttons

OK - accepts the entered fields and closes the dialog.

Cancel - cancels the creation of the new project and closes the dialog.

Add - adds then new row with such cells as from last row.

Delete - deletes the current row.

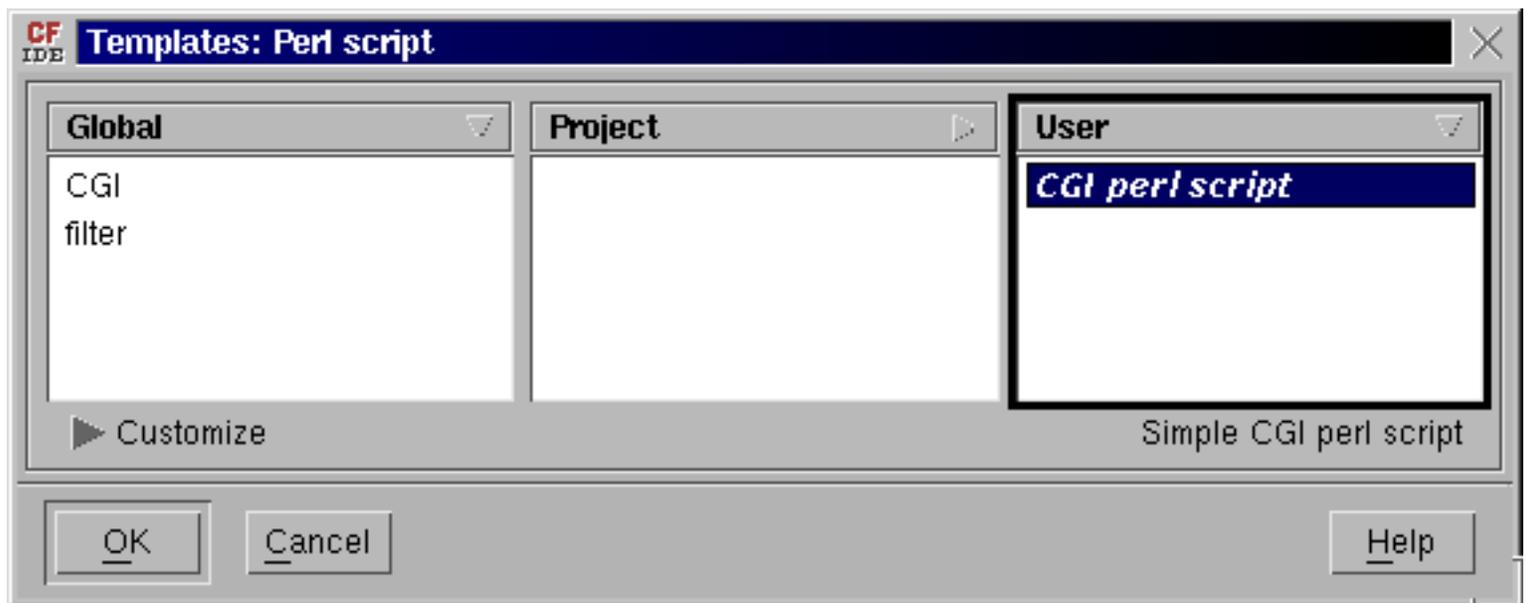
Help - shows this Help window.

Templates

When you often create sources that has identical structure you may wish to use **Templates**. There is a **Template** column in the [Create new target\(s\)](#) or [Create new node\(s\)](#) dialogs. If there are templates available for the file which is being created a push button appears.



Press this button to select template to fill the file with. The **Templates** dialog appears:



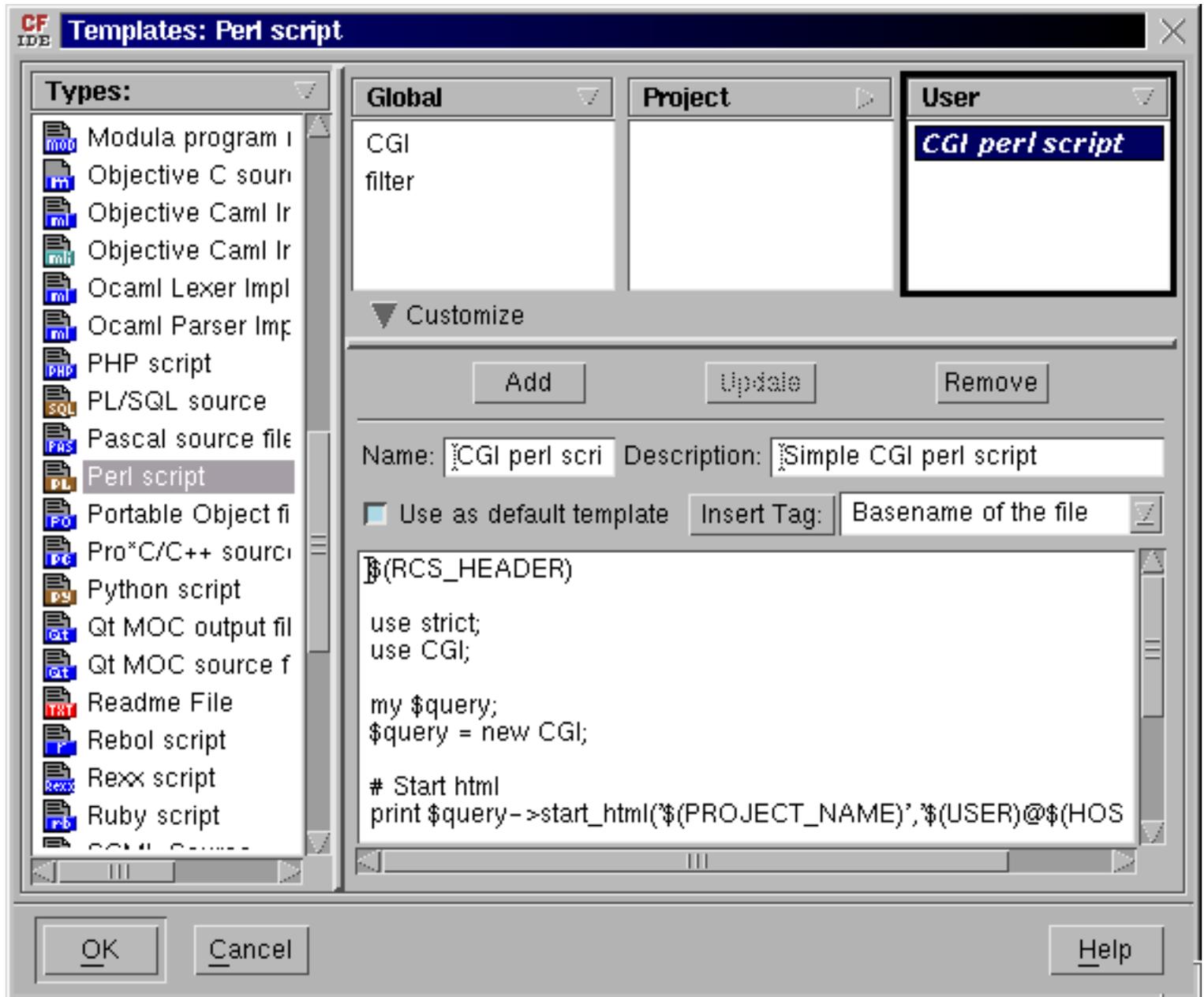
There are four levels of templates - **Global**, **System**, **Project**, **User**. Usually **System** level is hidden - it is displayed only if `$C_FORGE_SYSTEM_TEMPLATES` environment variable is set to the directory with system-wide templates. These templates may be set up by system administrator for all users and may be used instead of templates shipped with Code Forge. **Global** templates are installed with Code Forge and can be customized by system administrator too.

Templates selection precedence is the following:

- default template from **User** list
- default template from **Project** list
- default template from **System** list (if the `$C_FORGE_SYSTEM_TEMPLATES` environment variable defined)

- default template from **Global** list.

To override global or system templates user can copy this template into **Project** or **User** list with drag'n'drop operation, select new template and press **Customize** button. The customization part of the **Template** dialog appears:



Dialog consists from three parts:

- [Types List](#)
- [Template Selector](#)
- [Template Editor](#)

Types List

Types scrolled list consists of supported by Code Forge editable file types.

Template Selector

Template Selector consists of four scrolled lists:

- **Global List** - contains predefined by Code Forge templates placed in *\$C_FORGE/Templates* directory
- **System List** - contains templates placed in *\$C_FORGE_SYSTEM_TEMPLATES* directory. If that environment variable doesn't specified that list doesn't appears.
- **Project List** - contains templates placed in *\$PROJECT_DIR/Templates* directory
- **User List** - contains templates placed in *\$HOME/.c-forge/Templates* directory

To copy template from one list to another just drag'n'drop it to required list.

To add template to other file type just drag'n'drop it to required type.

Template Editor

The top part of the editor contains

- **Add** push button - add new template
- **Update** push button - update changed template
- **Remove** push button - remove current template
- **Name** text field - name of the current template
- **Description** text field - description of the current template
- **Use as default template** check-button - which points what the current template used by default
- **Insert Tag** push button - inserts a tag macro at cursor position in the template edit area.

Tag listbox contains the following macros

- Base name of the file
- Capitalize string
- Cast to identifier
- Current date
- Current time
- Hostname
- Lowercase string
- Project Author
- Revision Control Header
- Suffix of the file
- Uppercase string

- Username

Dialog Buttons

OK - selects highlighted template and closes the dialog.

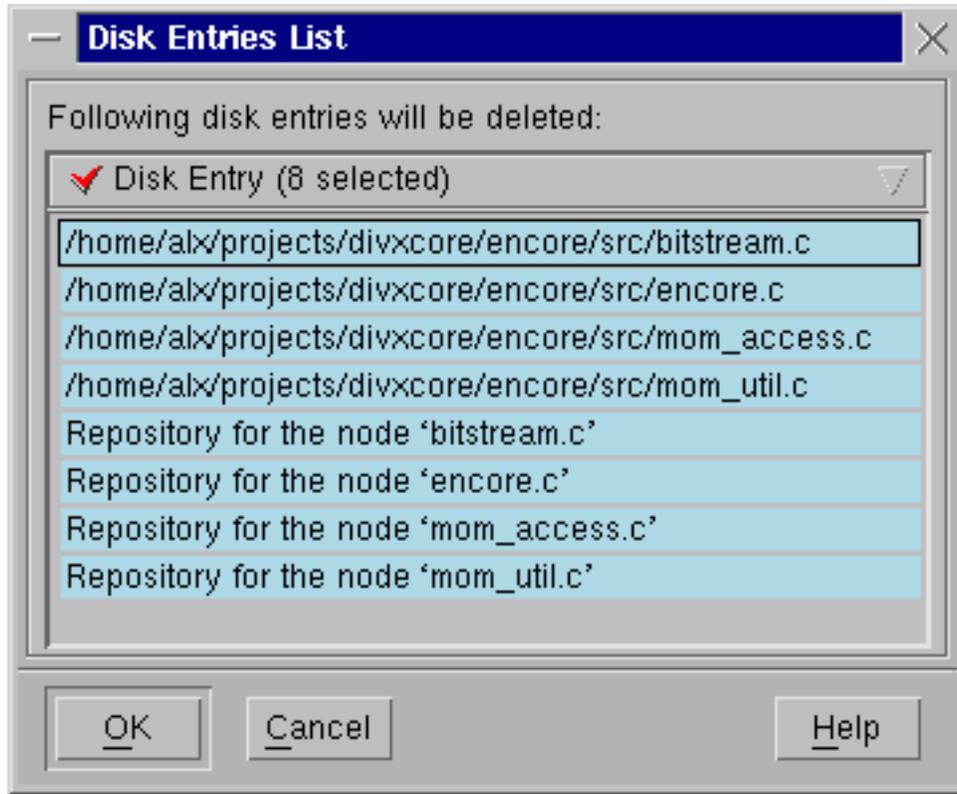
Cancel - closes the dialog without selecting template.

Help - shows this Help window.

Last modified: Monday, 14-May-2001 07:03:19 EDT

Delete Files Dialog

This dialog is used to manage a list of files that are to be deleted:



The mouse button can be used to select/deselect files. To select all files click on the red mark in the title.

Dialog Buttons

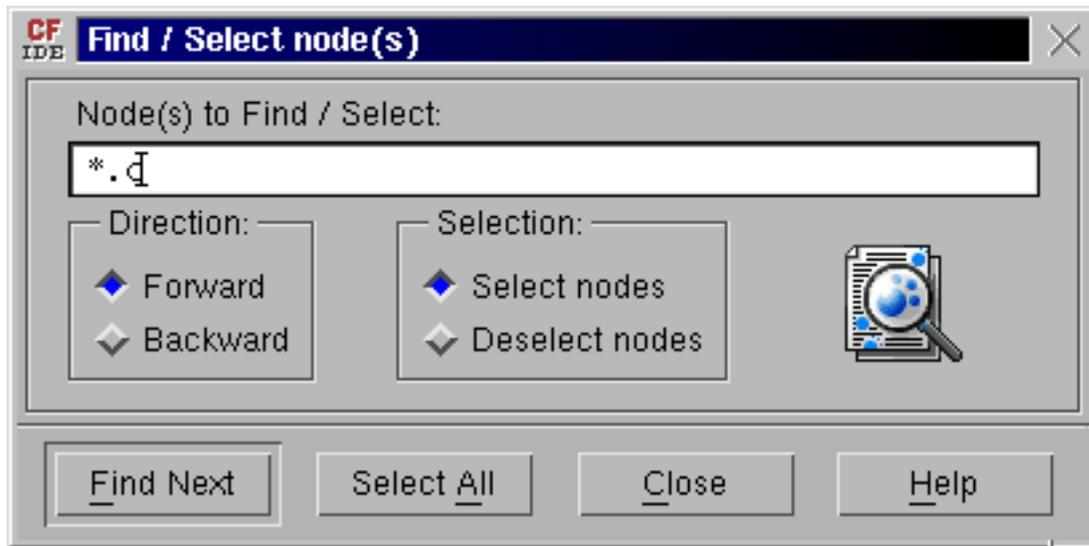
OK - deletes the selected files from disk and closes the dialog.

Cancel - closes the dialog without deleting files.

Help - shows this Help window.

Find Node Dialog

The **Find Node** dialog can be used to search the Dependency Tree for nodes matching a given pattern. It is activated by selecting **Find** in **Actions** menu or by pressing the **Ctrl+F** keyboard shortcut.



Node to Find - specifies the node to find; the name of the node can be entered using a [regular expression](#).

Direction - specifies the direction of the search.

Selection - specifies the options for **Select All / Deselect All** button; the name of the button gets changed depending on the option selected.

- **Select nodes** - selects all the nodes that match the **Node to Find** expression.
- **Deselect nodes** - deselects all the nodes that match the **Node to Find** expression.

Dialog Buttons

- **Find Next** - repeats the search.
 - **Select All / Deselect All** - depending on the **Selection** option - selects/deselects all the nodes that match the **Node to Find** expression.
 - **Close** - closes the dialog without executing the search.
 - **Help** - shows this Help window.
-

Node Properties Dialog

The **Node Properties** Dialog displays information about a node in the Project Desktop. This dialog is activated by choosing **Properties** from the **Actions** menu of the [Project Desktop](#) or by dragging a node and dropping it on the [Node Information Drop Site](#) or by pressing the **Alt-A** shortcut.

Dialog Buttons

OK - closes the dialog.

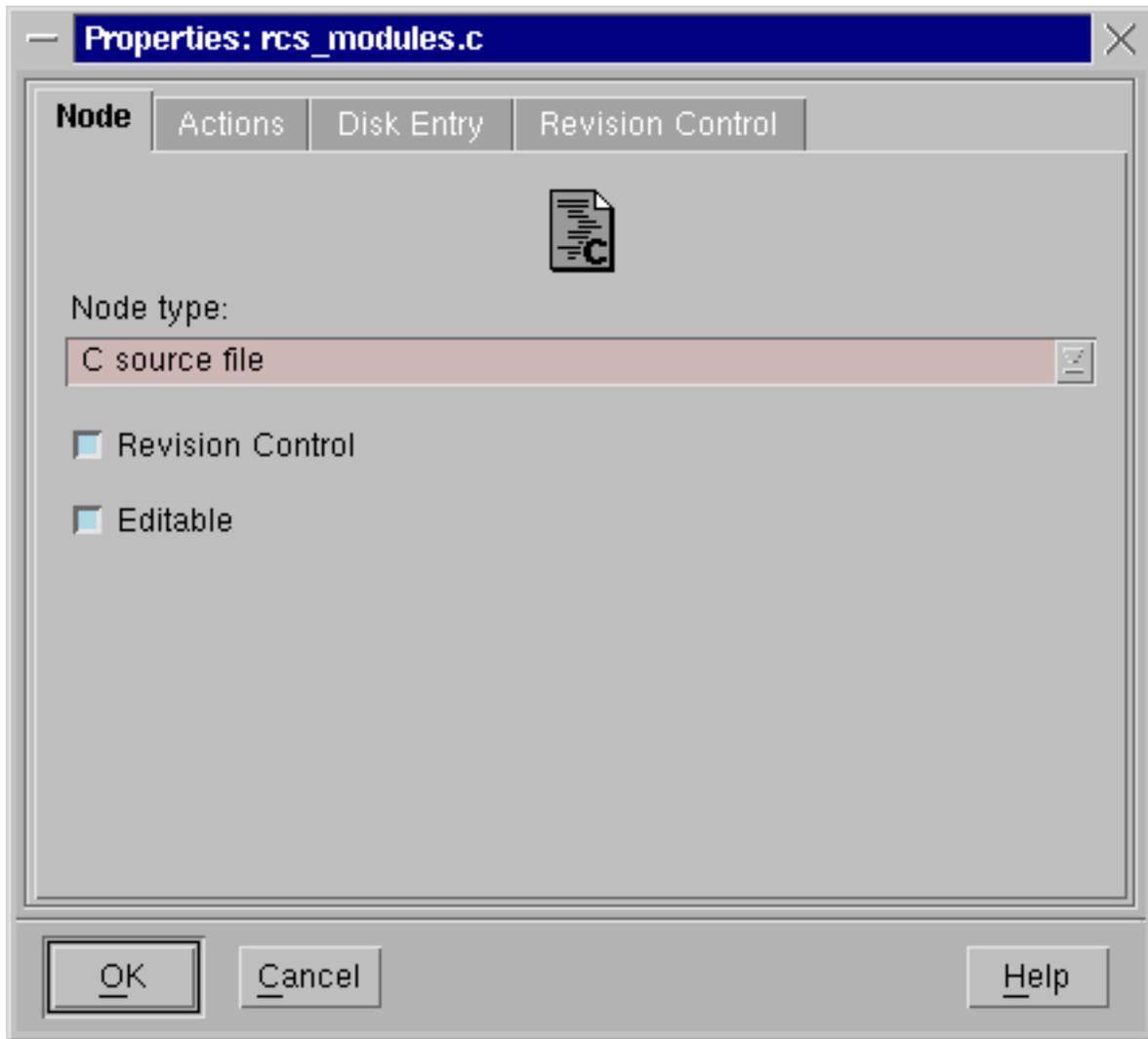
Cancel - closes the dialog without saving changes

Help - shows this Help window.

The dialog includes from one to four tabs depending on the type of a node, folder or directory:

Node Tab

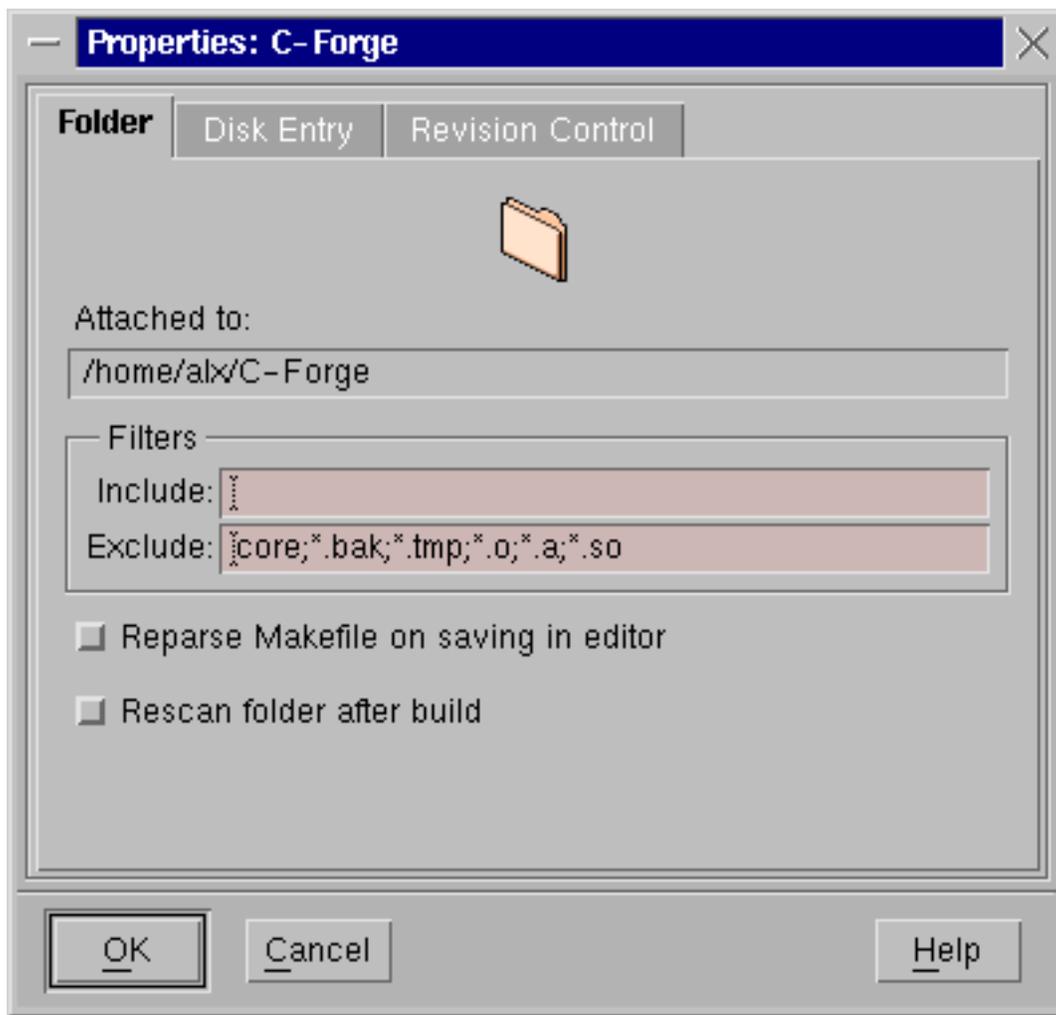
Node Tab can be used for changing the **Type**, subjection to revision control and editability of any node in the [Dependency Tree](#). This Tab can be accessed by selecting **Properties** submenu (**Node** command) from the **Actions** menu of the [Project Desktop](#) or by pressing the **Alt+A** shortcut.



- With the help of **Node type** combo-box node type can be specified or changed
- **Revision control** - subjects node to revision control. If the whole project or physical folder is not subjected to revision control, this check-box will not be shown on the tab.
- **Editable**
 - makes node editable.

Folder Tab

Folder Tab is used for viewing physical folder information, filtering folder entries and specifying the necessity to reparse makefiles of this folder after saving and rescan folder after build. Folder Tab can be accessed by selecting **Properties** submenu (**Node** command) from the **Actions** menu of the [Project Desktop](#) or by pressing the **Alt+A** shortcut keys.



- **Attached to** - specifies the path to the folder
- **Filters** - fields **Include** and **Exclude** specify filter using *sh* wild card pattern and thus indicating files that should be included or excluded while viewing the [Dependency Tree](#)
- **Reparse makefile on saving in editor** - makes the makefiles of the current folder be reparsed after editing and saving
- **Rescan folder after build** - makes the folder be rescanned after build operation (as **Rescan Folder for New** command of **Actions** menu)

Actions Tab

Actions Tab is used to view and edit actions needed to make, execute and debug a target. It can be accessed by selecting **Properties** submenu (**Build Action**, **Exec Action** or **Debug Action** command) from the **Actions** menu of the [Project Desktop](#) or by pressing the following shortcut keys:

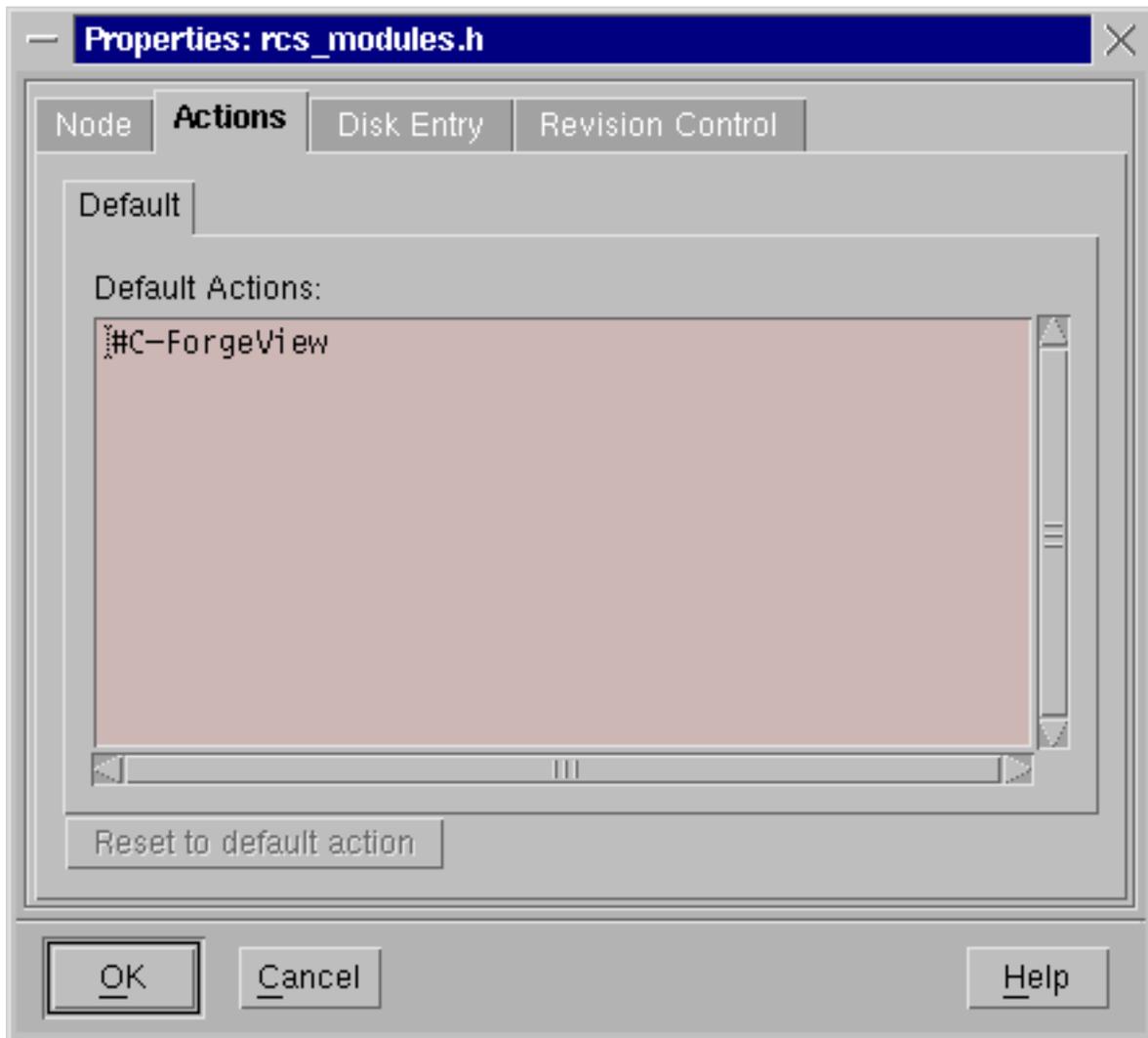
Build Action - action to make target (**Alt+T**)

Exec Action - default action by mouse double click on target (**Shift+Enter**)

Debug Action - action for debugging target (**Ctrl+Shift+D**)

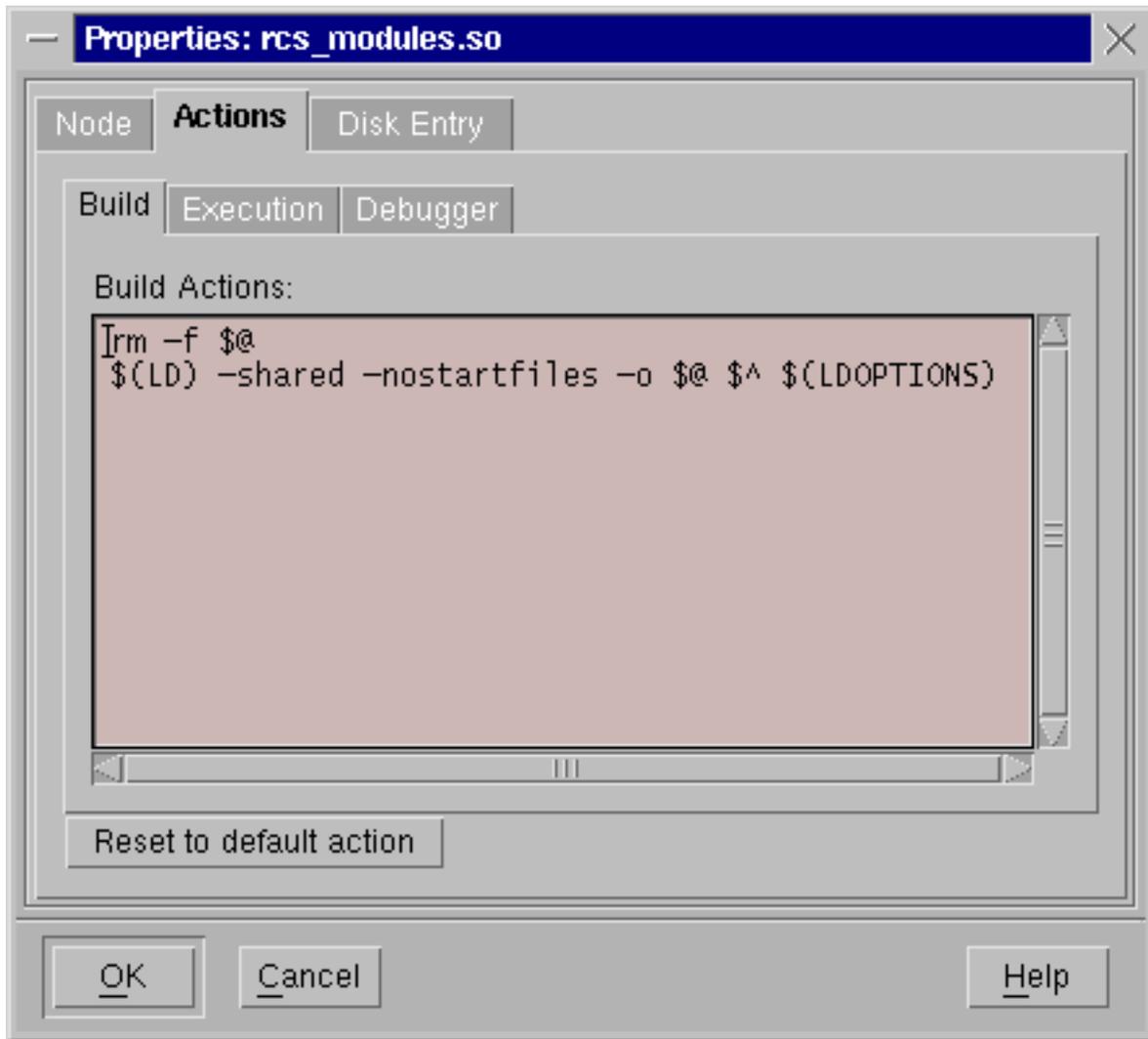
Depending on the type of a node the **Actions Tab** can include one or three tabs.

When the node is not executable, the default action is specified in the **Default** Tab. The default action can be edited. The button **Reset to default action** will change action, specified by user, back to default action.



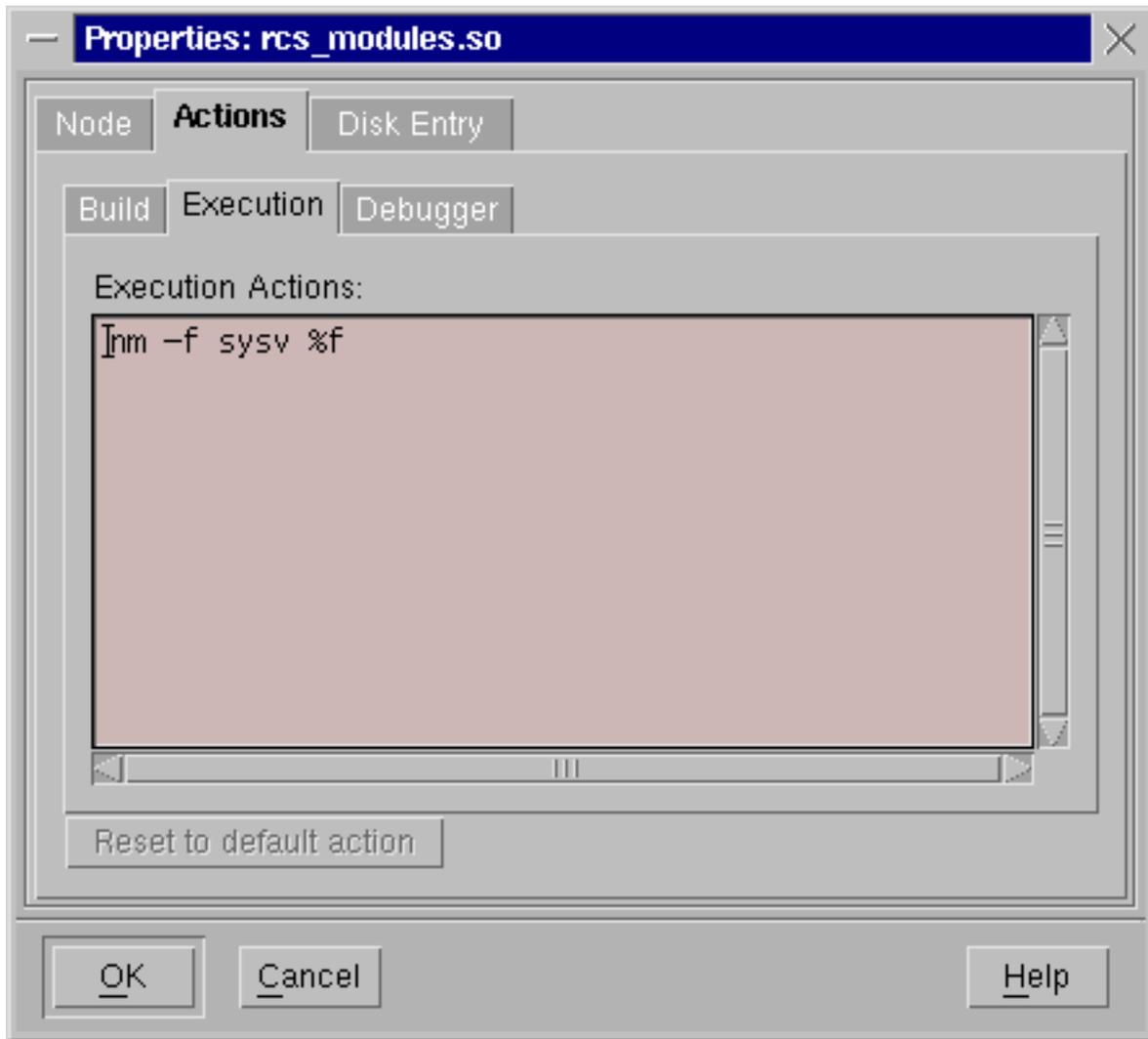
When the node is executable, the **Actions Tab** will include the following three tabs:

Build Tab



This field displays the description of the build target actions for the current node. This description includes macros, [GNU Make Automatic Variables](#), dependent file names and any other shell commands. The default Build Actions for current target you specified when selected **Type** in the [New Target](#) dialog.

Execution Tab



This field displays default (execute) action, performed on the specified target. This is simple shell command, but it can include any [Internal actions](#):

Internal Actions

- **#C-ForgeView** - View the current target using integrated [Smart Editor](#)
- **#C-ForgeExecute(or %f)** - Inserts the name of current target
- **#C-ForgeRunParams** - Pop-up dialog with input field and combo-box for *Run Parameters*
- **#C-ForgePrjWrkDir()** - inserts *C-Forge Working Dir* full path name
- **#C-ForgePrjSrcDir()** - inserts *C-Forge Stable Dir* full path name
- **#C-ForgeNodeName(arg1)** - inserts current node name.
 - Where *arg1*:
 - @ - full node name.
 - < - full node name without extension.
 - * - node name without directory and extension.
- **#C-ForgeTerminal(arg1,arg2)** - useful for debugging - start [Terminal Emulator](#) with the next parameters:

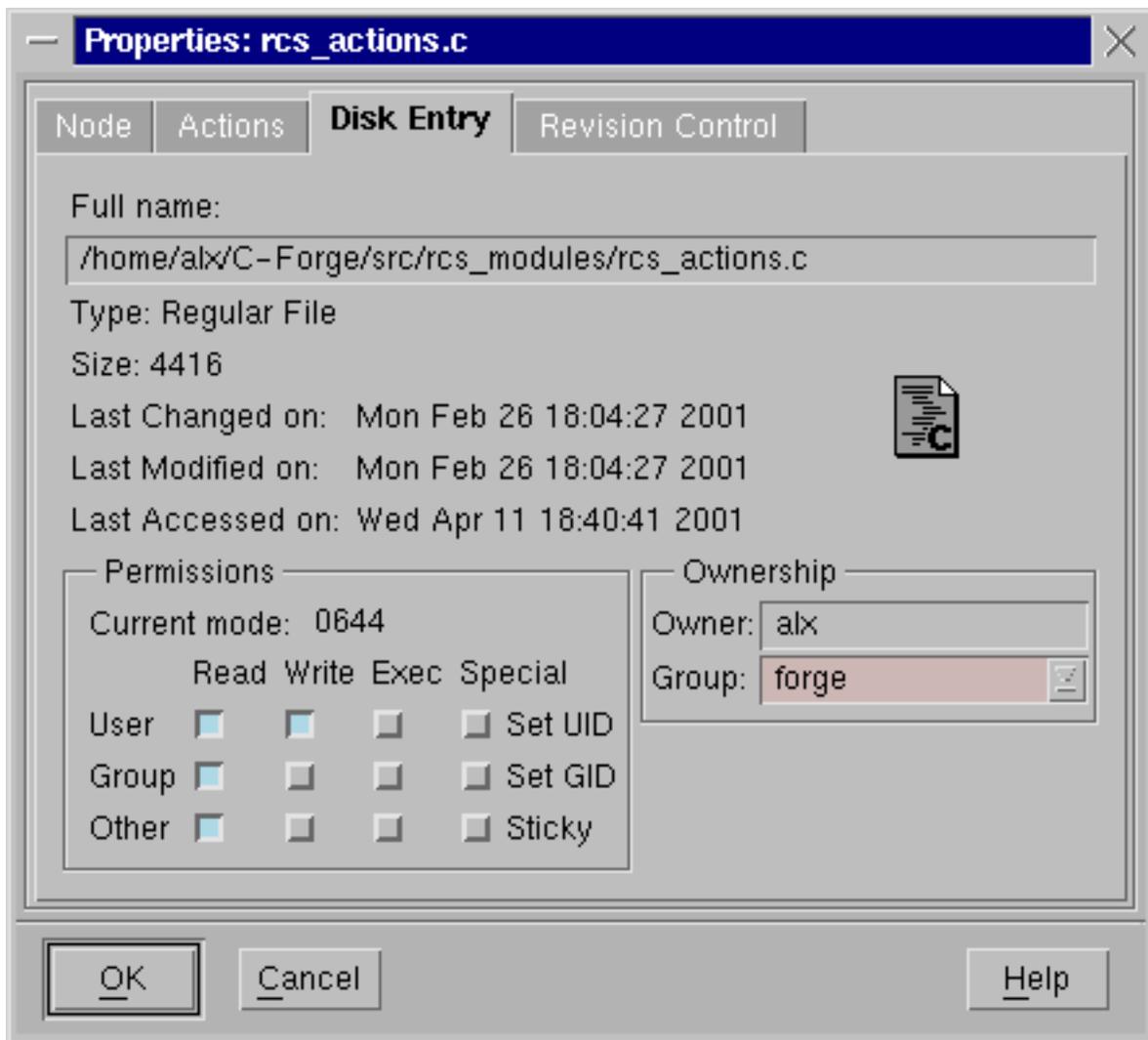
- *arg1* - terminal window name (-T switch is used).
- *arg2* - program to be executed.

Debugger Tab

This action describes the debugger, called on **Debug** action in [Project Desktop](#).

Disk Entry Tab

Disk Entry Tab includes node or folder disk entry information. It can be accessed by selecting **Properties** submenu (**Disk Entry** command) from the **Actions** menu of the [Project Desktop](#) or by pressing the **Alt+I** shortcut.



- **Full name** - displays full name (path) of the node or folder
- **Type** - displays type of the node or folder (directory, regular file etc.)
- **Size** - displays the size of the file (bytes)

- **Last Changed on** - displays date and time of last changes made on the file or directory
- **Last Modified on** - displays date and time of last file or directory modification
- **Last Accessed on** - displays date and time of last access to the file or directory
- **Permissions** - displays/edits the permissions to the current user.
- **Ownership** - displays/edits disk entry owner

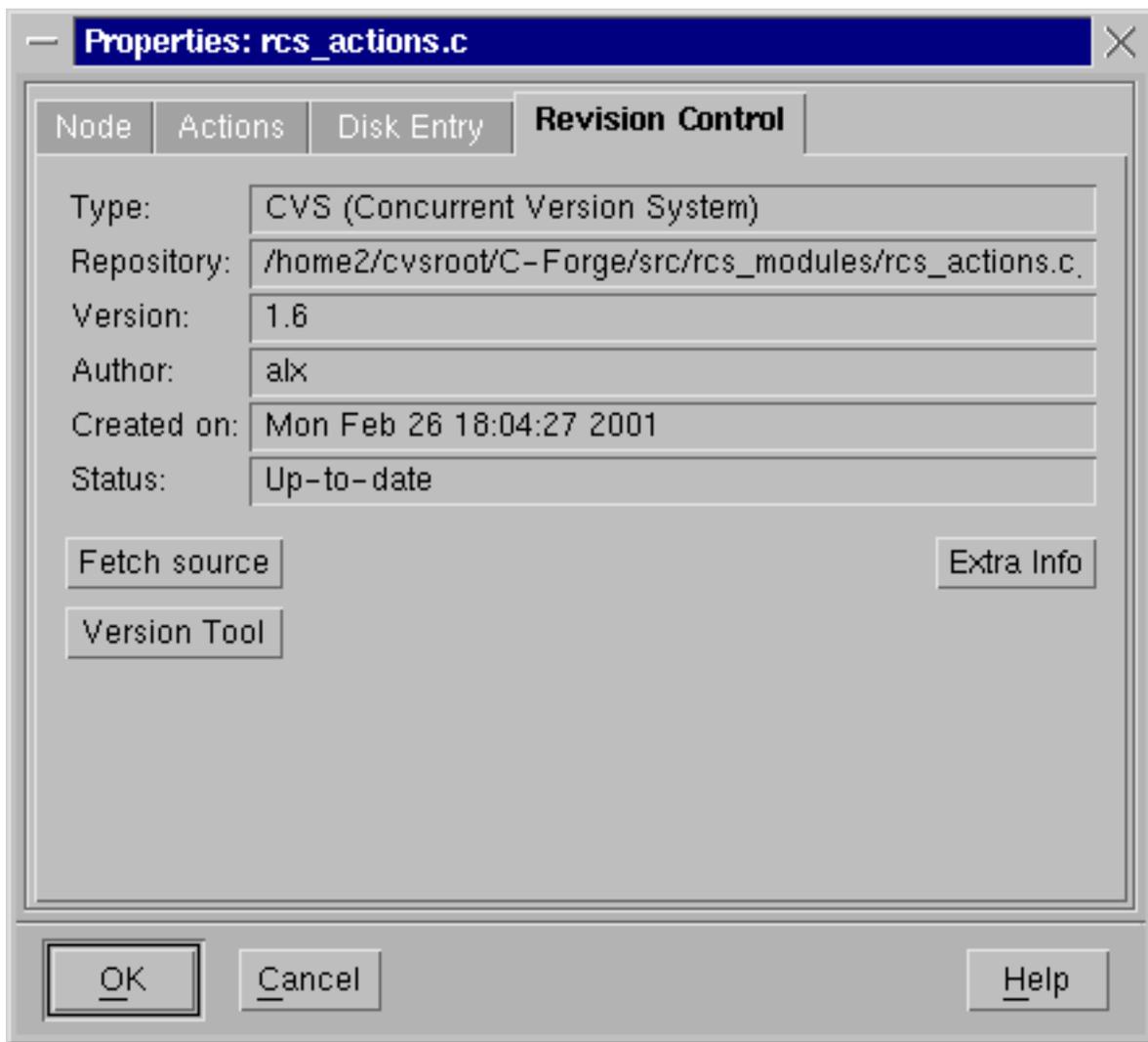
If the project has stable and working versions, the tab will include **Stable file** and **Working file** check-boxes. If **Stable file** check-box is marked, **Full name** field will display the name and location of the [stable version](#) of the node file. If **Working file** check-box is marked, **Full name** field will display the name and location of the [working version](#) of the node file.

If the file does not exist, only its full name and message **the disk entry does not exist** will be displayed.

Revision Control Tab

Revision Control Tab is used for viewing and editing of revision control information on the selected node or physical folder. It can be accessed by selecting **Properties** submenu (**Revision Control** command) from the **Actions** menu of the [Project Desktop](#).

If the file is selected, the Tab will display:

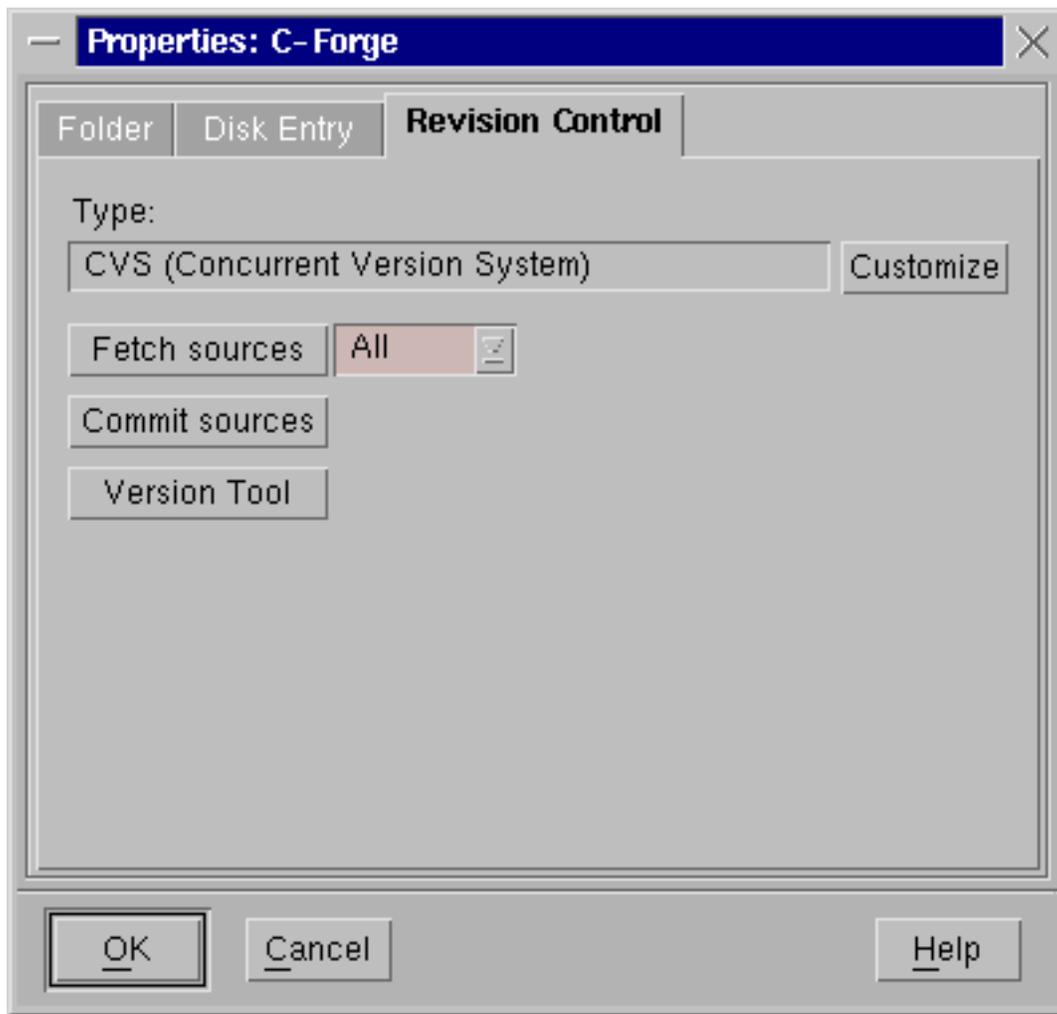


- **Type** - displays type of revision control for the selected file
- **Repository** - displays path to the repository of the selected file
- **Version** - displays the number of the current version of the file
- **Author** - displays file author's name
- **Created on** - displays date and time of file creation
- **Status** - displays status of the current file version

Buttons:

- **Extra Info** (optional) - opens **Extra Info** window containing additional information on file repository and working revisions. The window also contains **Close** (closes **Extra Info** window) and **Help** (shows this help window) buttons.
- **Fetch Source** - fetches file from the repository. **Warning!** All the changes made in the file will be cancelled while performing of this operation!
- **Version Tool** - opens the [Revision Control window](#)

If the physical folder is selected, the Tab will display:



Type - displays type of revision control for the selected folder

Buttons:

- **Customize** - opens **Revision-Control Options** window used for viewing and editing of repository options
- **Fetch sources** - opens [Fetch](#) window containing the list of files. If value **All** is specified in the combo-box next to the button, the **Fetch** window will contain all the files of this folder. Those files are already selected, so as soon as you press **OK** button of the window, all the folder files will be fetched from the repository. **Warning!** All the changes made in the folder will be cancelled while performing of this operation! If value **Missing** is specified, only files missing in your version will be included into the list.
- **Commit sources** - performs check-in operation for each file of the current folder
- **Version Tool** opens the [Revision Control window](#)

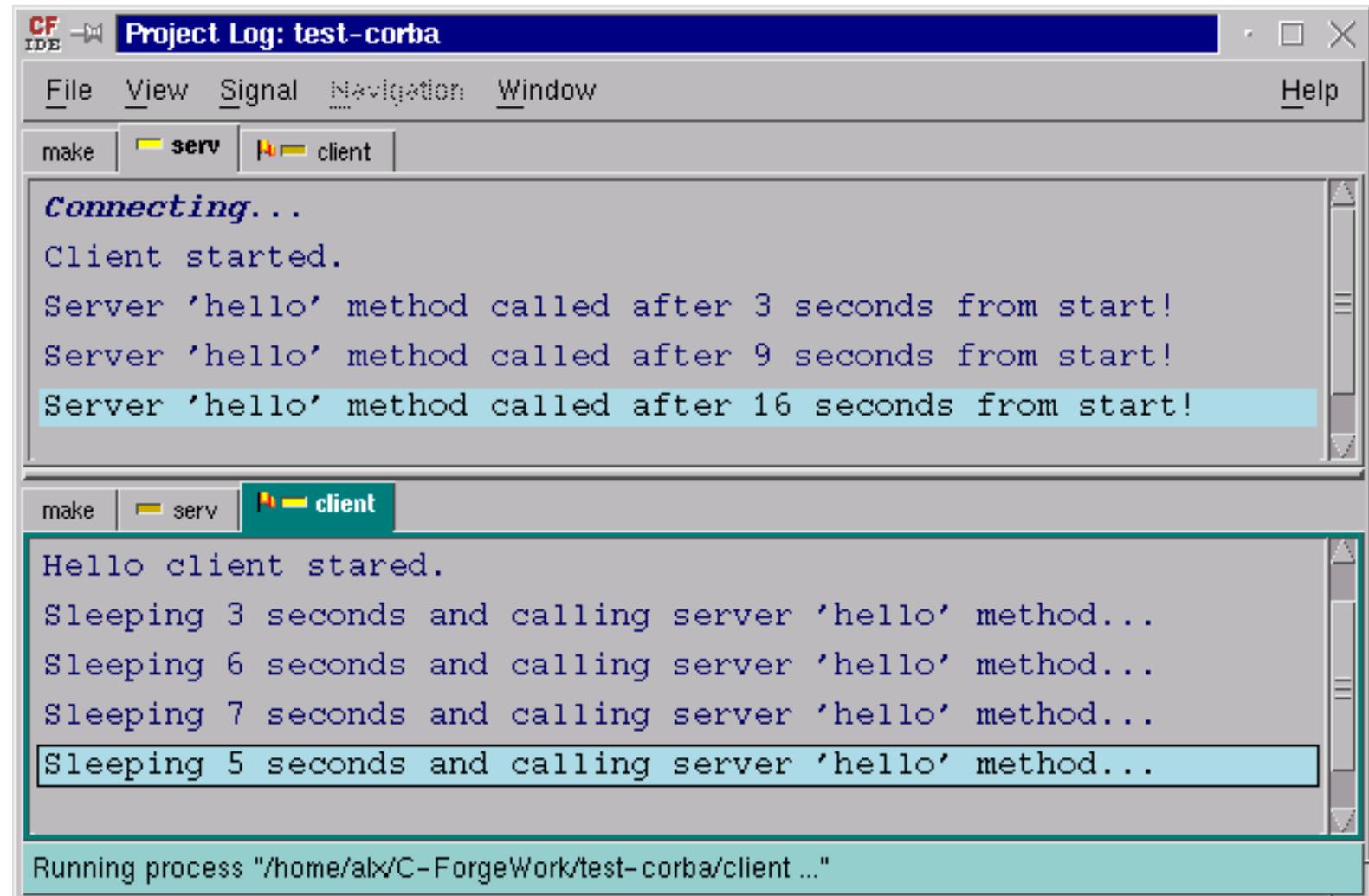
Project Log Window

- [Overview](#)
- [Process Output List](#)
- [Menu Bar](#)
- [Status Bar](#)

The **Project Log** window displays the output of the compiler, linker and other user programs. This screen is divided into three parts: the **menu**, **processed output** list, and **status bar**. Anything sent to **STDOUT**, or **STDERR**, is captured and displayed in this window.

By default **Project Log** is shown as a part of [Project Desktop](#) and can be torn away by selecting **Tearaway** command of the pop-up menu.

Project Log now can control execution of one and more processes:



The yellow indicator on the panel tab flashes while the process is running.

Process Output List

The process output list displays text directed to the **STDOUT**, or **STDERR** devices. If the selected line contains a valid file name, either pressing the **Enter** key or double clicking with the mouse can open the editor. If the line also contains a line number the editor is brought to that line number. Programs can be stopped at any time by pressing the **Ctrl+C** shortcut keys.

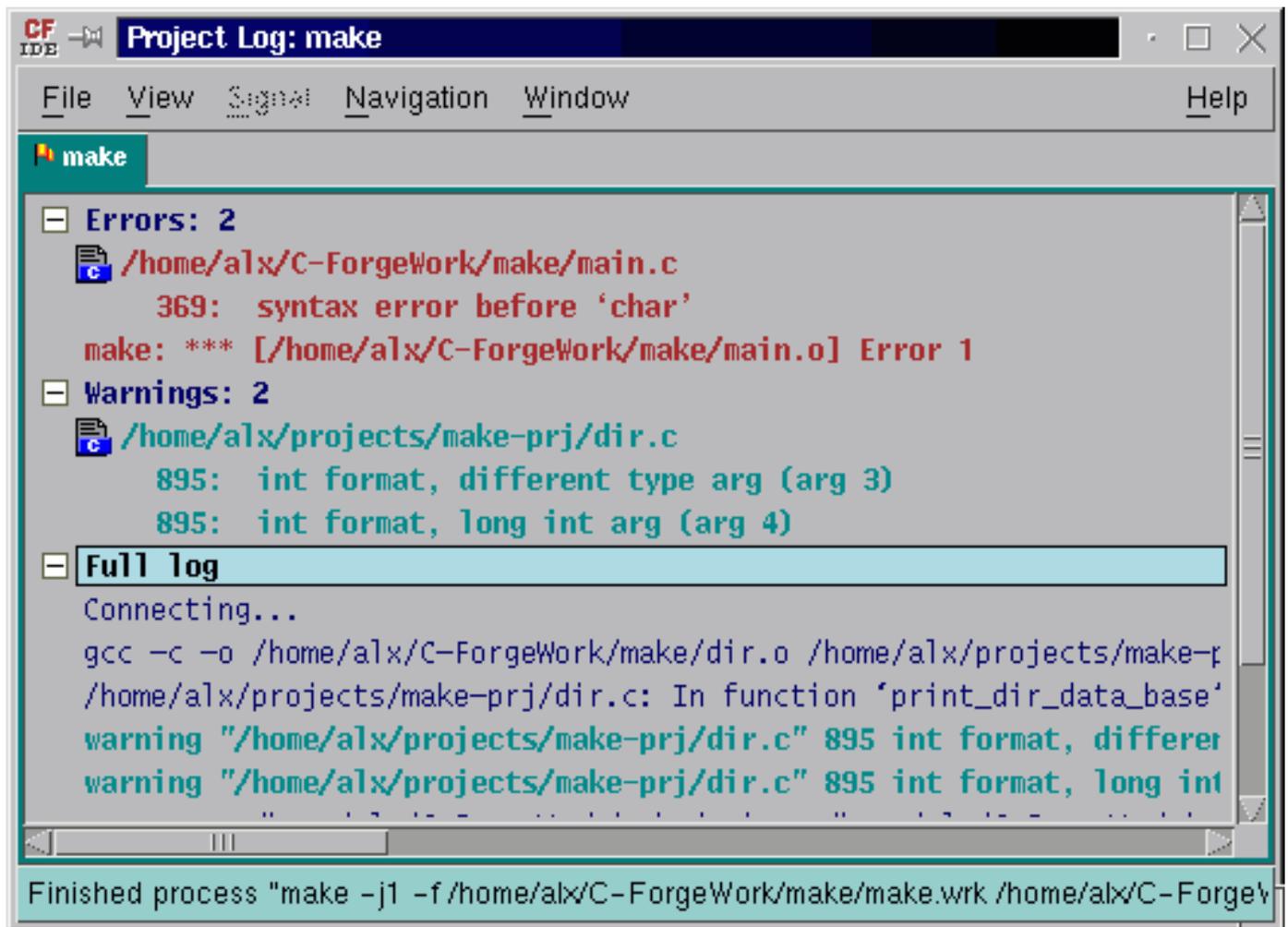
```

CF IDE Project Log: tst
File View Signal Navigation Window Help
make
Connecting...
gcc -c -o /home/ide/C-ForgeWork/tst/main.o /home/ide/C-ForgeWork/tst/main.c
warning "/home/ide/C-ForgeWork/tst/main.c" 11 return-type defaults to 'int'
/home/ide/C-ForgeWork/tst/main.c: In function 'main':
warning "/home/ide/C-ForgeWork/tst/main.c" 12 implicit declaration of function 'printf'
warning "/home/ide/C-ForgeWork/tst/main.c" 13 control reaches end of main function
gcc -o /home/ide/C-ForgeWork/tst/hello /home/ide/C-ForgeWork/tst/main.c
/home/ide/C-ForgeWork/tst/main.o: In function 'main':
error "/home/ide/C-ForgeWork/tst/main.c" 12 undefined reference to 'printf'
make: *** [/home/ide/C-ForgeWork/tst/hello] Error 1
Execution complete
Finished process "make -j1 -f /home/ide/C-ForgeWork/tst/tst.wrk /home/ide/C-ForgeWork

```

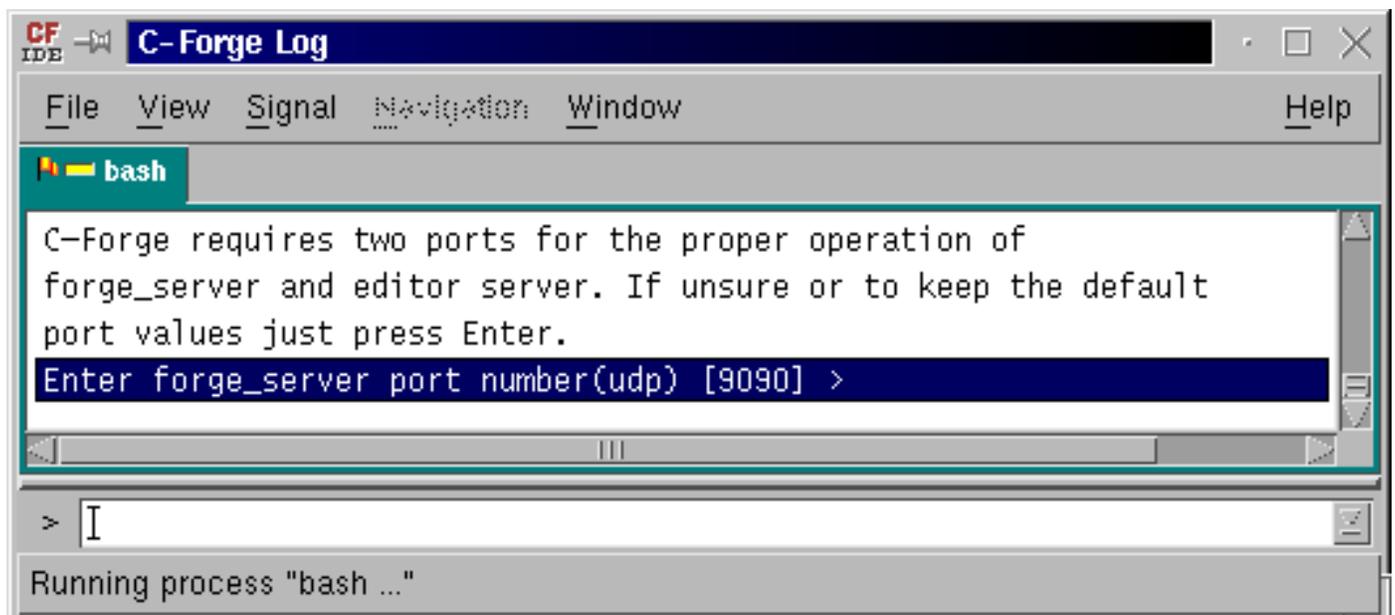
Separated Output

When the **Separate errors/warnings in the Log Window** is enabled in the [Project Options](#), various error types are separated into groups as shown in the screenshot below:



User Input

To send to called process any text or signals use Prompt Bar. To send signal via **Ctrl+<Key>** keyboard focus must be in the combo-box.



Menu Bar

File

- **Options** - opens the Log Options section of the [Options](#) dialog.
- **Edit file** - If the current line includes the name of the file with an error/warning and a line number, an [Editor Window](#) is opened for the file and the appropriate line is bookmarked.
- **Correct error(s)** - opens or raises [Editor Windows](#) for all the files with errors and bookmarks the appropriate lines.
- **Clear all editor bookmarks** - clears all the bookmarks in the Editor Windows previously generated by the **File | Correct errors** command.
- **Add new log** - adds new log and sets it as current.
- **Set as current** - sets current log as default for output.
- **Save Log as** - saves the contents of the log window to a file.
- **Load Log** - loads a previously saved log file into the log window.
- **Retry execution** - retries the execution of the latest process.
- **Stop** - kills the process.
- **Close** - closes the log Window.

View

- **Prompt Bar** - show/hide user input drop-down combo-box.
- **Tearaway** - tears a Log Window from the Project Desktop.
- **Add view** - adds new view.
- **Remove view** - removes current view.
- **Clear lines** - clears lines in current panel.
- **Close panel** - closes one current panel only.

Signal - this menu can be used to send various signals to the process executing in the log window. This menu is available only while the process is active.

Navigation

The **Navigation** Menu allows the traversal of error lines.

Clicking the right mouse button opens pop-up menu with the commands of the **File** menu.

Status Bar

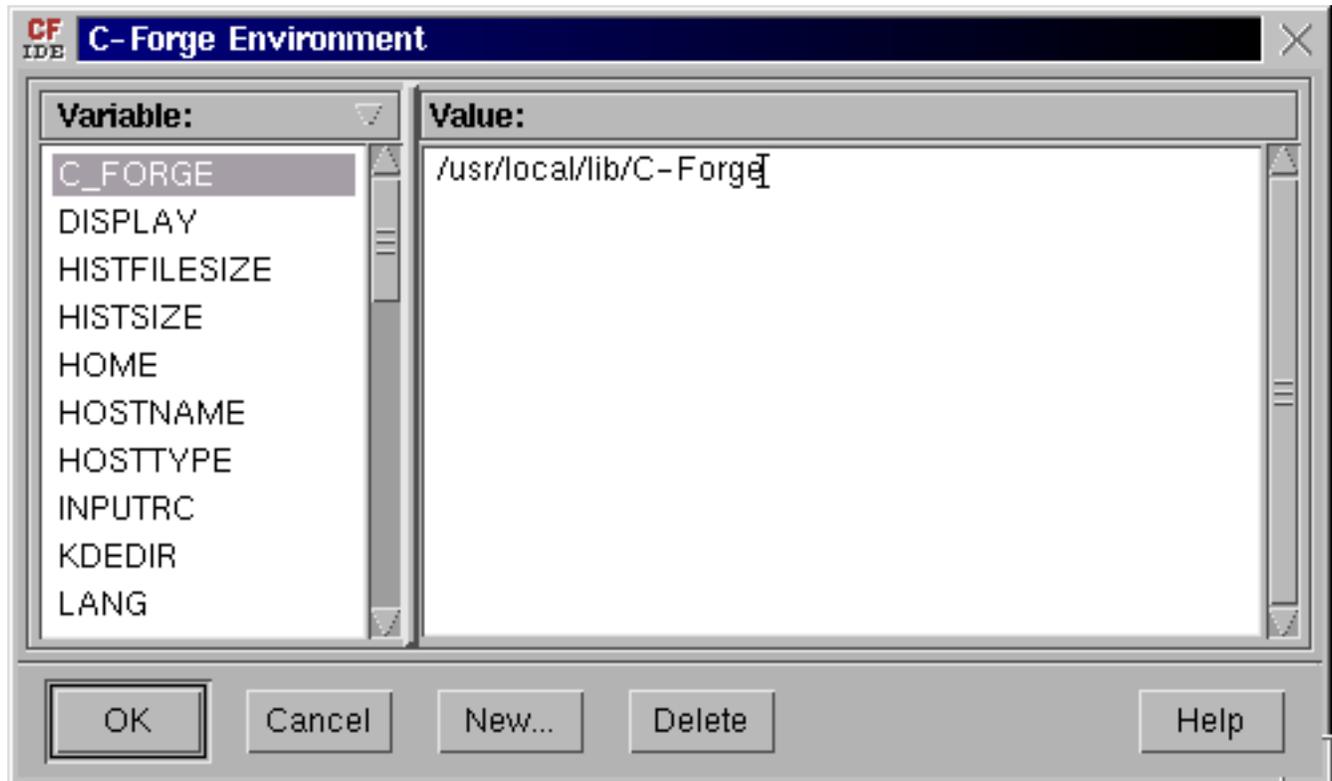
The Status bar displays the command line for the process that is currently running and the tips for the

selected menu items.

Last modified: Monday, 14-May-2001 07:03:19 EDT

Environment

In the **Environment** dialog is used to edit the values of environment variables. Selecting **Environment** in the **Options** menu activates this dialog. When activated from the [Project Manager](#), changes made will affect environment of all projects. If activated from the [Project Desktop](#), only current project specific environment variables would be modified.



The main part of the dialog is divided into two areas. The left area displays a list of variables and the right area displays the value of the selected variable.

For deleting, several variables at a time can be selected by clicking on them with the left [mouse](#) button and **Ctrl** or **Shift**.

Dialog Buttons

OK - saves all the changes and closes the dialog.

Cancel - cancels all the changes and closes the dialog.

New - adds a new variable; Code Forge prompts for the name of the variable.

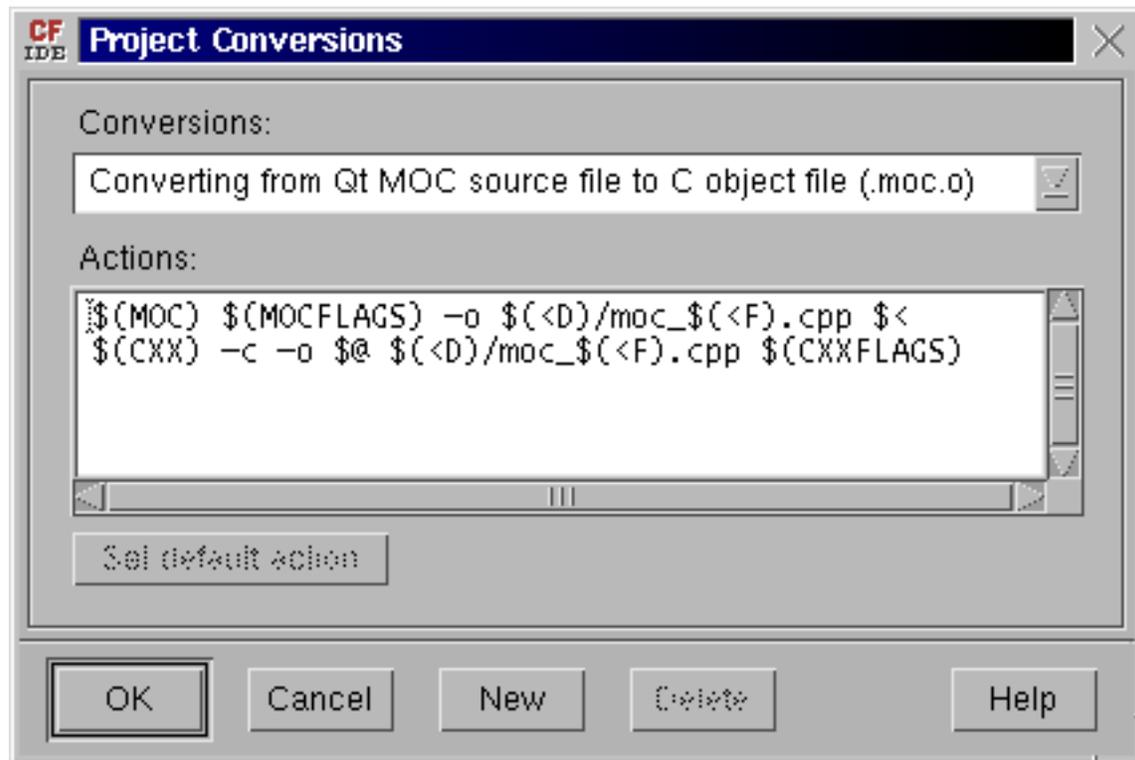
Delete - deletes selected variables.

Help - shows this Help window.

Last modified: Monday, 14-May-2001 07:03:19 EDT

Project Conversions

The **Project Conversions** window displays the actions needed to convert file types. The actions described in **Project Conversions** will be used for all the implicit conversions of file types for this project.



Conversions combo-box includes the list of all the conversion rules defined in the current project.

The **Actions** text box displays the description of the rules of the selected conversion. This description includes macros, [GNU Make Automatic Variables](#) for the target file name and dependent file names, and any other *shell scripts*.

The **Set default action** push button allows reset current action to default action.

Dialog Buttons

OK - saves the changes and closes the dialog.

Cancel - closes the dialog without saving the changes.

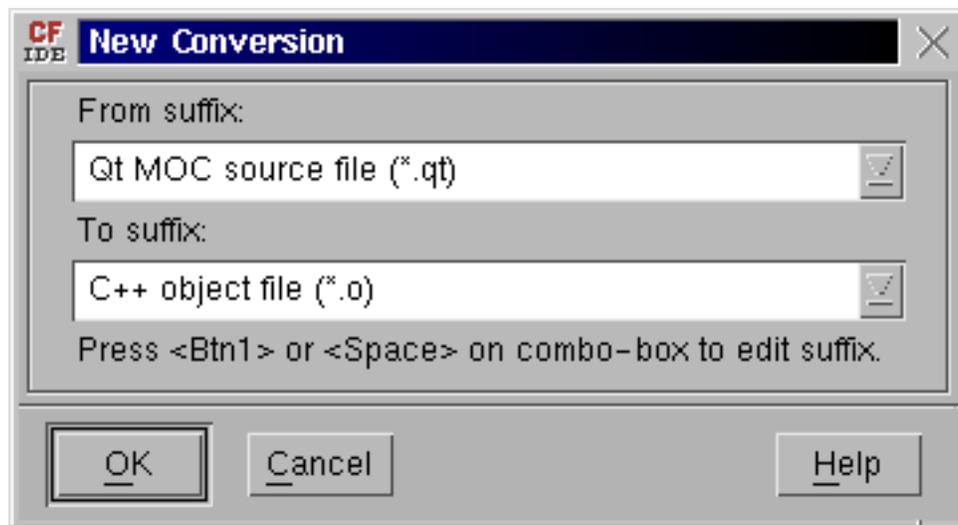
New - allows adding new conversion rules. New Conversion dialog is opened.

Delete - allows deleting non-default conversion rules.

Help - shows this Help window.

New Conversion Dialog

The **New Conversion** dialog is accessed by selecting the **New** button from the [Project Conversions](#) dialog.



From suffix and **To suffix** fields specify the original and final types of the file. Clicking on one of these fields makes it updateable and replaces the description of the file with the suffix describing the file type. When a field is losing focus, the suffix is replaced with the description of the type that is known to C-Forge.

Dialog Buttons

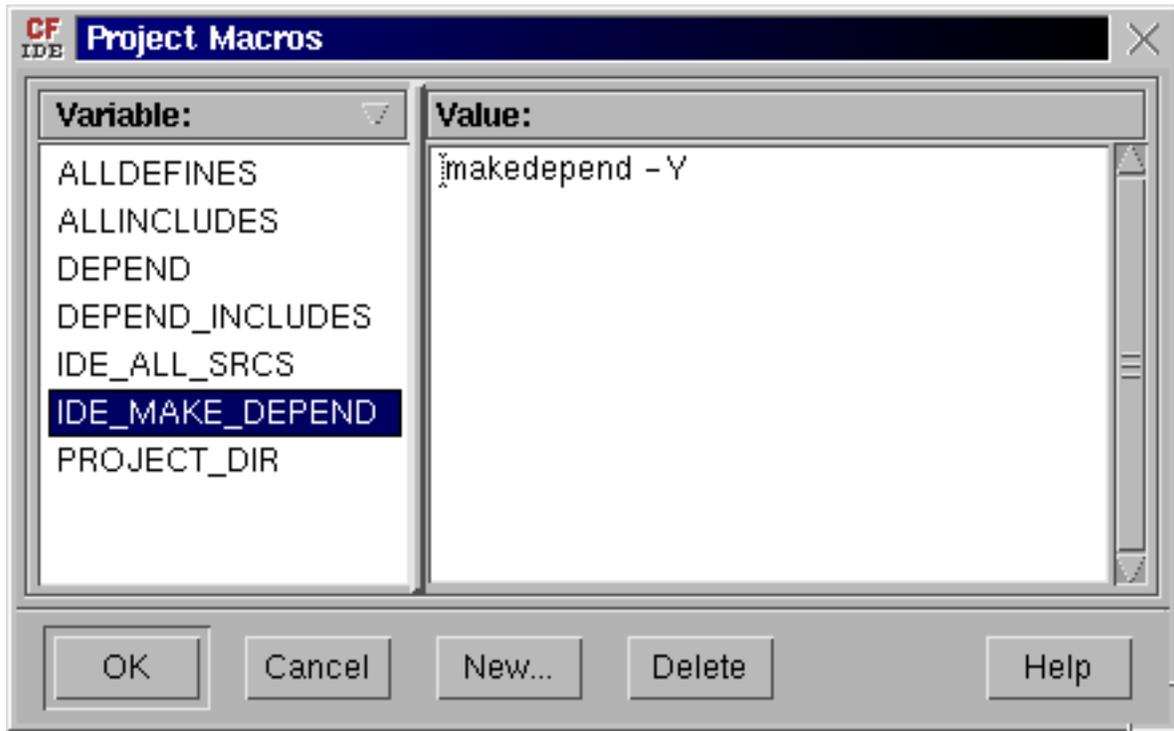
OK - adds the new conversion to the **Conversions** combo-box of the [Project Conversions](#) dialog.

Cancel - cancels adding the new conversion.

Help - shows this Help window.

Project Macros

The **Project Macros** dialog displays macros that are unrelated to project files and do not appear in the [Dependency Tree](#) Macros branch. To activate this dialog select the **Macros Options** menu of the [Project Desktop](#).



The main part of the dialog is divided into two areas. Left area displays the list of the macros; right area displays the description of the selected macro.

Several variables at a time can be selected by clicking on them with the left [mouse](#) button.

Dialog Buttons

OK - saves all the changes and close the dialog.

Cancel - cancels all the changes and close the dialog.

New - adds a new macro; C-Forge prompts for the name of the macro.

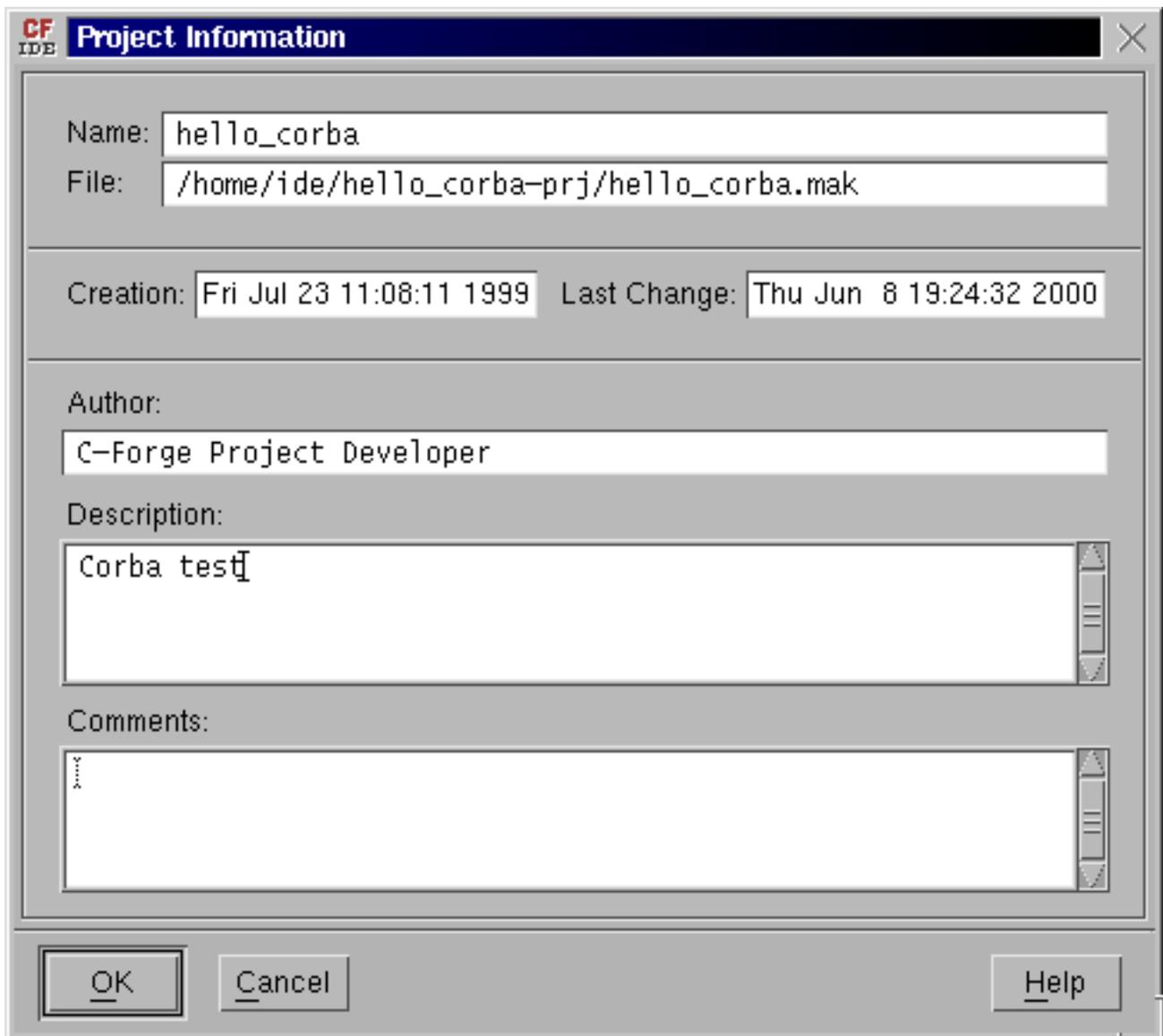
Delete - deletes selected macros.

Help - shows this Help window.

Last modified: Monday, 14-May-2001 07:03:19 EDT

Project Information Dialog

The **Project Information** dialog is used to view and edit Project Information. It can be accessed by selecting **Project Info** from the **File** menu of [Project Manager](#) or by pressing the **Ctrl+I** shortcut key.



Name - displays the name of the project.

File - displays the name and location of the project file.

Creation - displays the date and time when the project was created.

Last Change - displays the date and time when the project was last modified.

Author - displays the author of the project.

Description - displays a brief description of the project.

Comments - displays any additional information about the project.

Only **Description** and **Comments** fields can be modified.

Dialog Buttons

OK - closes the dialog and saves changes in **Description** and **Comments** fields.

Cancel - closes the dialog and does not save changes in **Description** and **Comments** fields.

Help - shows this Help window.

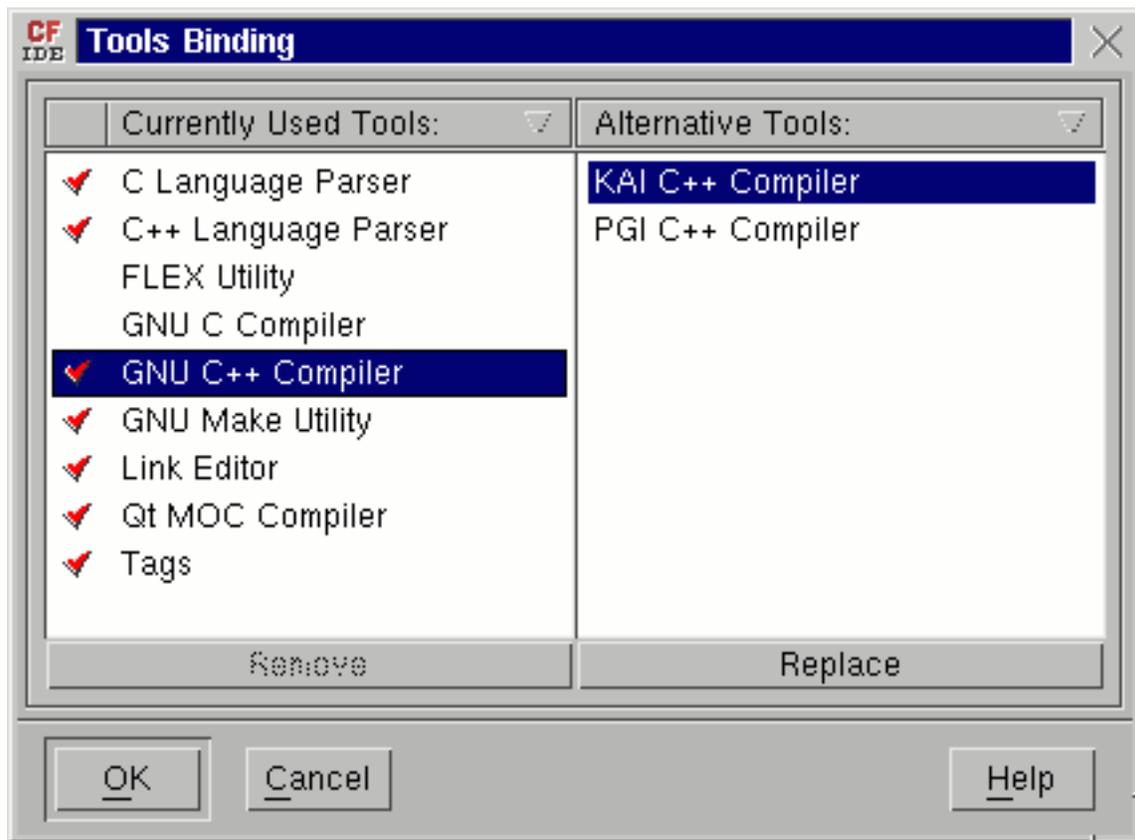
Last modified: Monday, 14-May-2001 07:03:19 EDT

Tool Binding Dialog

The **Tools Binding** dialog is used to manage the tools used in the project.

Tools are automatically added to this list when they are needed to process a newly added target. This dialog can be used to remove unused tools as well as specify alternative ones - if they are available

This action can be initiated by selecting **Tools Binding** from the **Actions** menu on the [Project Desktop menu bar](#) or by pressing the **Ctrl-B** accelerator.



The tools used in the project are listed in the left panel of the dialog. The tools that are **CURRENTLY** in use are marked with a red checkmark. Tools can be removed by selecting them and then clicking on the **Remove** button located below the list.

When a tool is selected in the **Currently Used** panel, a list of alternative tools appears in the panel in the **Alternative Tools** panel. Tools can be switched to their alternatives by clicking on the **Replace** button located below the list

Dialog Buttons

OK - saves the changes in used tools and closes the dialog.

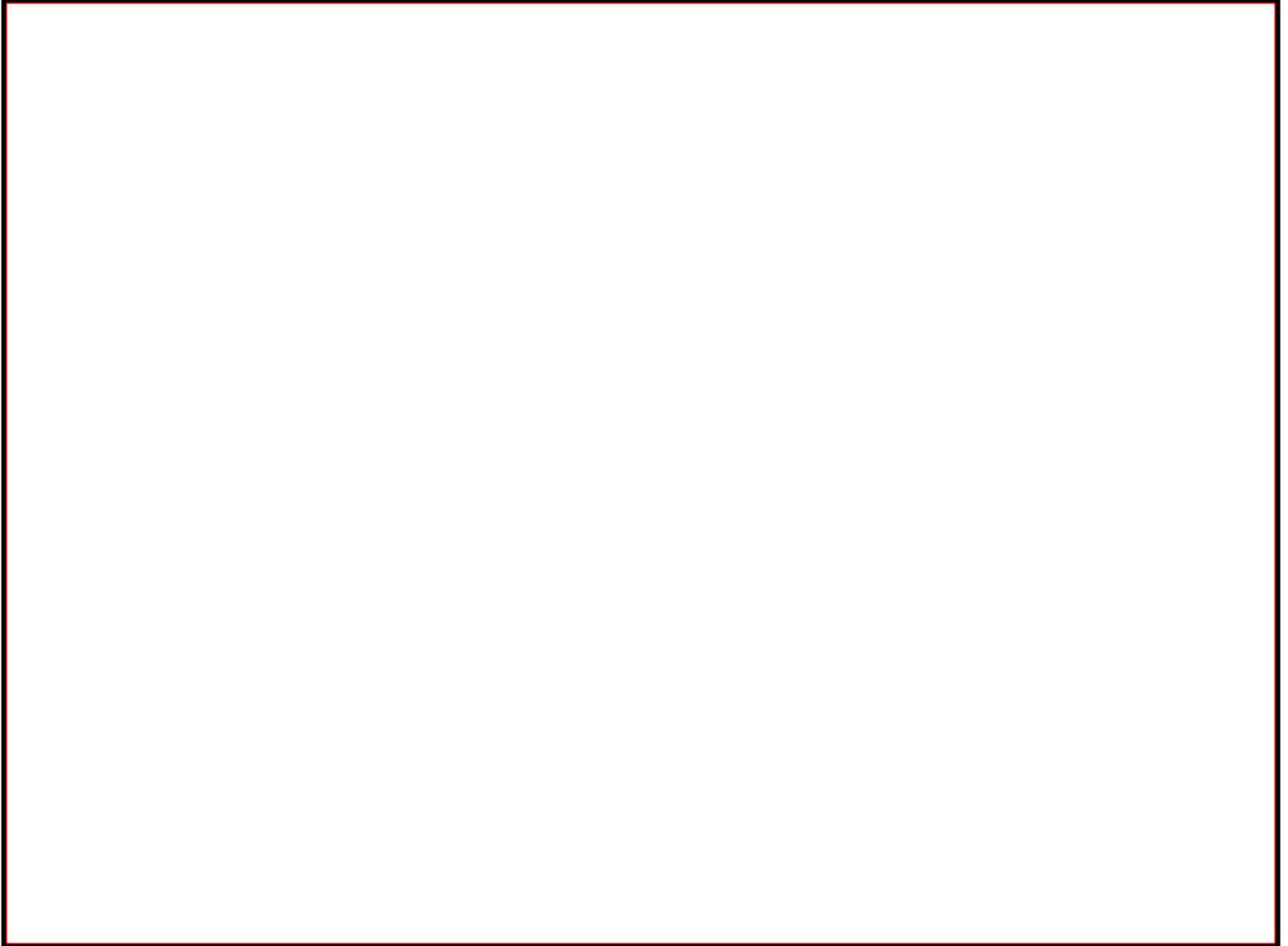
Cancel - closes the dialog without saving the changes.

Help - shows this Help window.

Last modified: Sunday, 01-Jul-2001 11:44:31 EDT

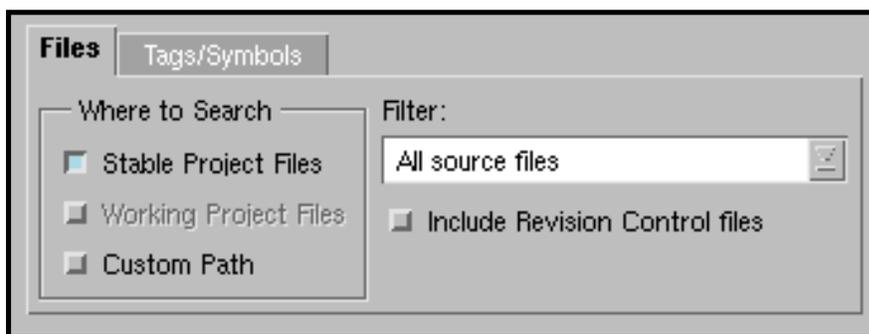
Search/Replace Tool

The **Search/Replace Tool** searches the files for lines containing a match to the given pattern. Selecting **Search/Replace** in the **Actions** menu of the [Project Desktop](#) activates it.



- **Pattern** - the text or regular expression to be search for.
- **Replace With** - the text of expression which will replace the found regular one.
- **Start/Stop** - starts/terminates the search.
- **Replace** - opens the [Replace](#) window on the screen

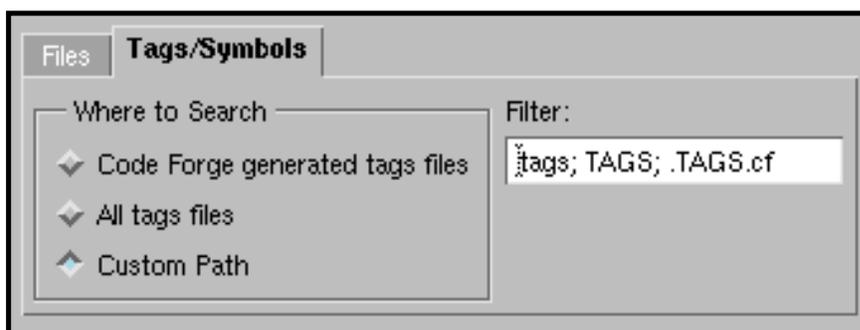
- **Files** tab:
When this tab is selected the tool searches the search pattern in the files.



- **Where to search** group box - is used to specify the directories to search. Files is looked in any combination of [Stable](#), [Work](#) or a [Custom Path](#).
- **Filter** - the types of files to be searched.
- **Include Revision Control Files** - when is selected, Revision Control Files are included into search.

- **Tags/Symbols** tab:

When this tab is selected the tool searches the search pattern in the tags.



- **Where to search** group box is used to specify where to search the tags files:
 - **Code Forge generated tags files** - search the automatically generated tags files only.
 - **All tags files** - search the tags files with names specified in the **Filter** text field.
 - **Custom Path** - search the tags files with names specified in the **Filter** text field in the [Custom Path](#) specified directories.
- **Filter** - the tags files names list.

- **Custom Path** group box:

- **Search directories recursively** option - when is selected, subdirectories found in the search path will also be scanned.
- **Browse** button - opens the [Path Edit](#) dialog, which can be used to compose a custom path.
- **Path** text area - displays the custom path.

- **Search Options** group box:

- **Ignore Case** - when is selected, case distinctions will be ignored in both the pattern and the input files.
- **Match Words** - when is selected, only lines containing matches that form whole words are shown.

- **Exact Match** - when is selected, only those lines exactly matching the whole line will be selected.

The lower part of the dialog displays the table of the matches found. It shows names and locations of the files that contain the matches, and the line numbers. The title bar of the table is the standard element of the [interface](#), that allows changing the order and size of columns and the sorting the their elements.

When collapse all  mode is selected the **Search Results** table contains only paths to the files where the search pattern has been found. When expand all  mode is selected the **Search Results** table also displays the lines where the search pattern has been found.

Double-click on matches in the table will open the corresponding file to be opened in an [Editor Window](#).

 When right mouse button pressed in the **Search Results** table popup menu with **Symbol Navigator** item is displayed. Selecting this item opens the [Symbol Navigator](#) using the definition of the currently selected symbol under cursor.

The yellow indicator on the **Status Bar** of the dialog flashes while the process is running.

Replace dialog

Replaces the regular expression matching to the specified pattern with the expression, specified in the *Replace With* field.



- In the upper part of the window the regular expression found and the expression which will replace it

will be displayed.

- **File name** - displays name of the files found.
- **Permissions** - displays files' access permissions.
- **In Project** - field is ticked if the file belongs to the project
- **Check-Out** - if the field is ticked, automatic check-out will be done before replacing
- **Check-In** - if the field is ticked, automatic check-in will be done before replacing
- **Interactive** - if the field is ticked, the editor window will be displayed for manual replacing of the expression found.
- **Path** - path to the file is specified in this field.
- **Make backup copy** - toggle button is pressed by default. This option allows saving the current file as `file.bak` before replacing.

Double-click on the record opens the [SMED Editor](#) with **Search/Replace** dialog activated.

Dialog Buttons

- **OK** - accepts the entered fields. Process bar appears in the bottom of the window displaying the process going on.
When the process is completed the **OK** button will be replaced by **Done** button.
- **Done** - signals that the replacing has been completed. Closes the current window.
- **Cancel** - cancels replacing and closes the dialog.
- **Remove** - removes the selected record (file) from the list for replacing only. It means that replacing will not be done in this file.
- **Help** - shows this Help window.

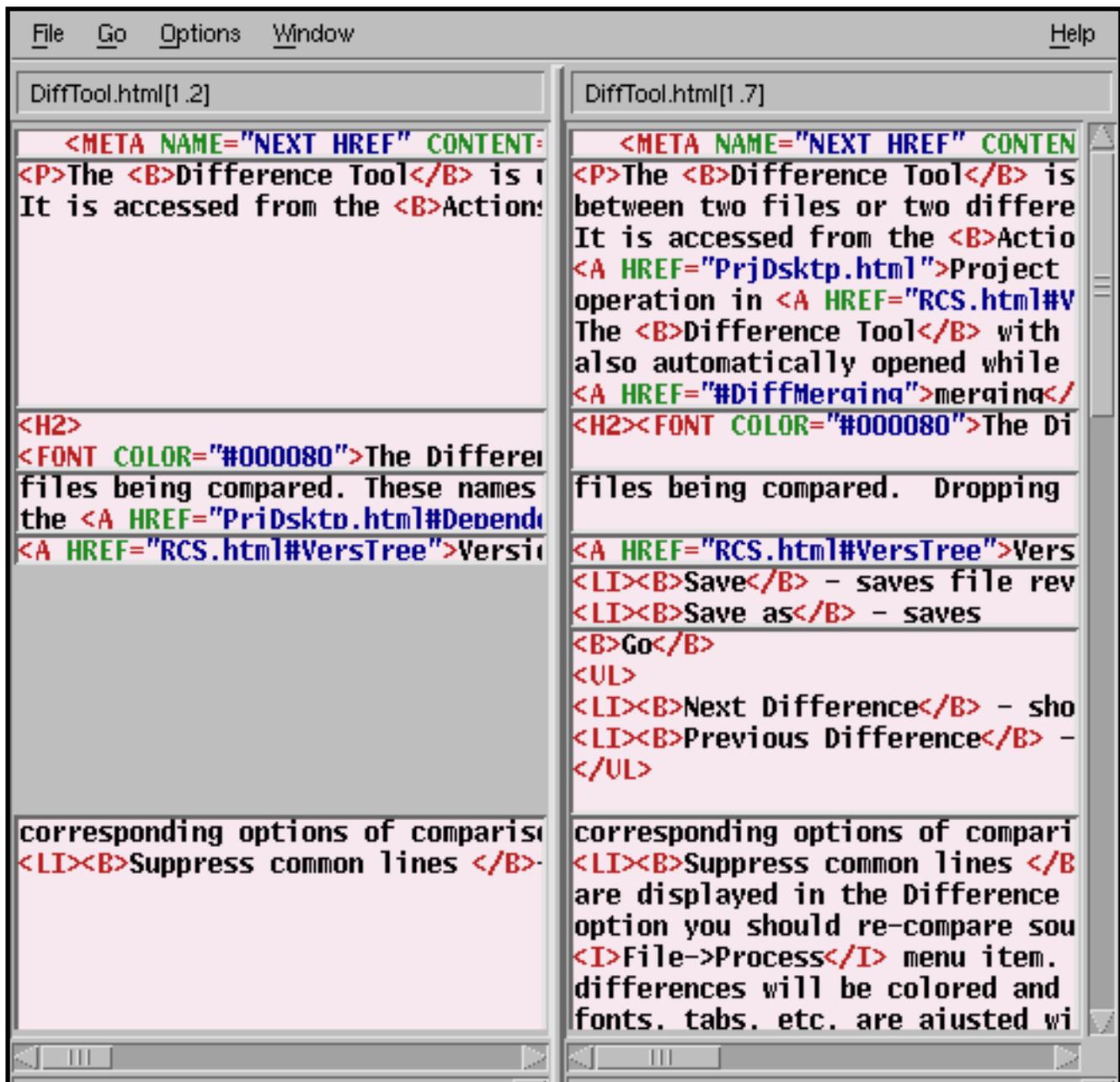
Last modified: Monday, 01-Apr-2002 09:05:26 EST

Diff/Merge Tool

The **Diff/Merge Tool** is used to display the differences between two files or two different versions of the file and to merge that differences to another file. It is accessed from the **Actions** menu in the [Project Desktop](#) or from Drag'n'Drop operation in [Revision Control](#) window. The **Diff/Merge Tool** with **Merge Result** column will be also automatically opened while [merging](#), when [Revision Control](#) conflicts arises.

The Difference Tool Window

This window is divided into four parts: the **comparison area**, the **menu**, the left and right file name **drop sites**, and the **status bar**.





The comparison area is divided into two parts. Each part displays the lines of one file that differs from the lines of another file.

File Name Drop Sites

The Left and Right **File Name Drop Sites** display the names of the files being compared. Dropping a file from the [Project Desktop Dependency Tree](#), [File Selection Dialog](#) or [Version Tool Revision Tree](#) can modify these names.

Menu

File

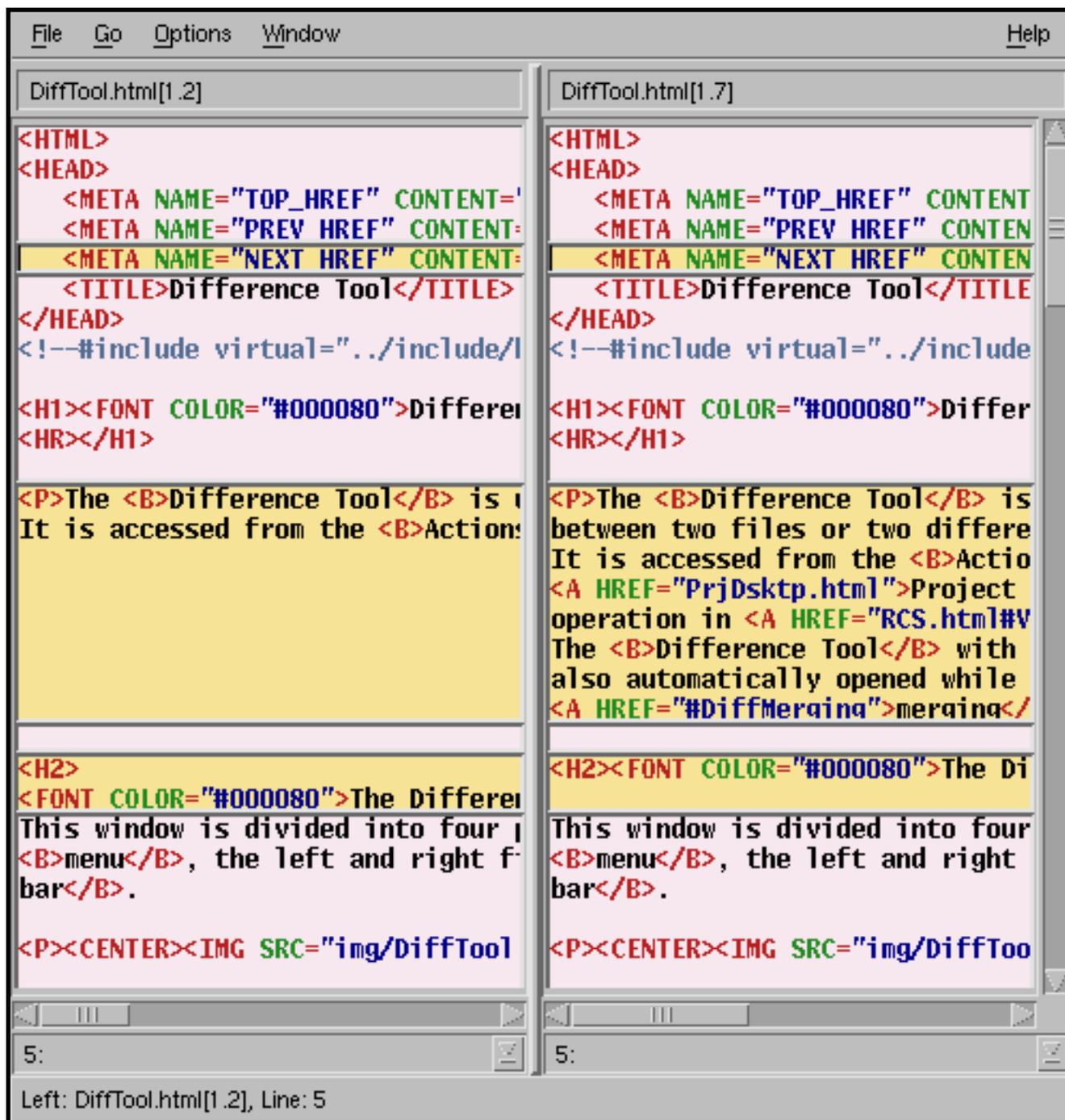
- **Load** - opens the [dialog](#) to select files to be loaded to the Difference Tool.
- **Save** - saves file revision.
- **Save as** - saves
- **Process** - refreshes Difference Tool window.
- **Close** - closes Difference Tool window.

Go

- **Next Difference** - shows the next line where difference is found
- **Previous Difference** - shows previous difference

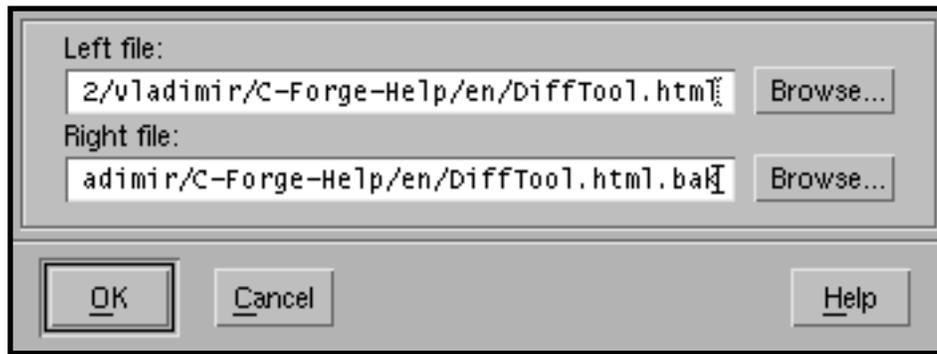
Options

- **Switch to merge/diff** - switch tool form [diff](#) to [merge](#) mode or backwards.
- **Ignore changes in case, Ignore changes in all white spaces, Ignore changes** in amount of white spaces and **Ignore blank lines** - specify corresponding options of comparison. Selecting of each option causes reparsing.
- **Suppress common lines** - specifies if common lines of two files are displayed in the Difference Tool window. After setting this option you should re-compare source files using *File->Process* menu item. If the option is switched off, the differences will be colored and placed in separate frames. Colors, fonts, tabs, etc. are adjusted with the help of [Custom Resources Dialog](#), and can not be configured from **Diff Tool**.



Selecting Files

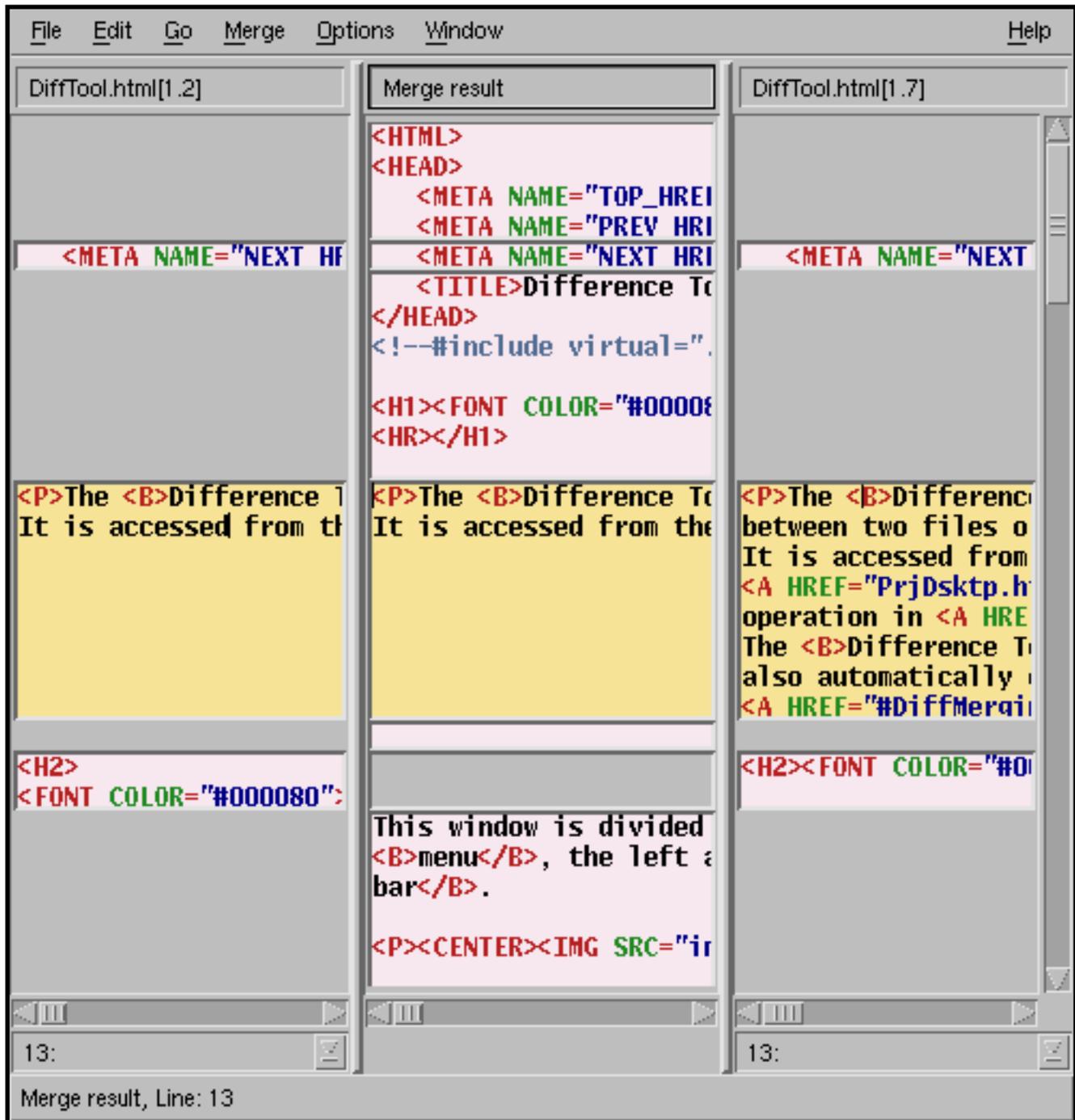
An input dialog appears prompting for two file names. The **Browse** button is available to select the file paths.



Clicking the **OK** button will display the differences, clicking **Cancel** will exit the dialog.

Merging

When several users work on the same file under CVS repository, conflicts may arise while committing. In this case **Diff/Merge Tool** window will be automatically opened. The window may be also called by user from the [Revision Control](#) window by dragging and dropping one of the file versions to another, while **Ctrl** button being pressed:



The window is divided into 3 parts.

- The **left part** displays differences found in one of file versions while comparing contents of two files. File name and path is specified in the header of the left part.
- The **right part** displays differences found in the other file version. The file name and path is specified in the header of the right part.
- The **middle part** at first displays only lines, common to both file versions. When no common lines are found, it displays left part contents. Field header is **Merge result**.

Suppress common lines option of the **Options** menu does not work here.

Each difference found is divided from other ones by horizontal frame line.

To resolve the conflict in favor of the left part file content, choose **Merge all for left** command from the **Merge** menu. The **Merging result** field will include all the differences found in the left part file.

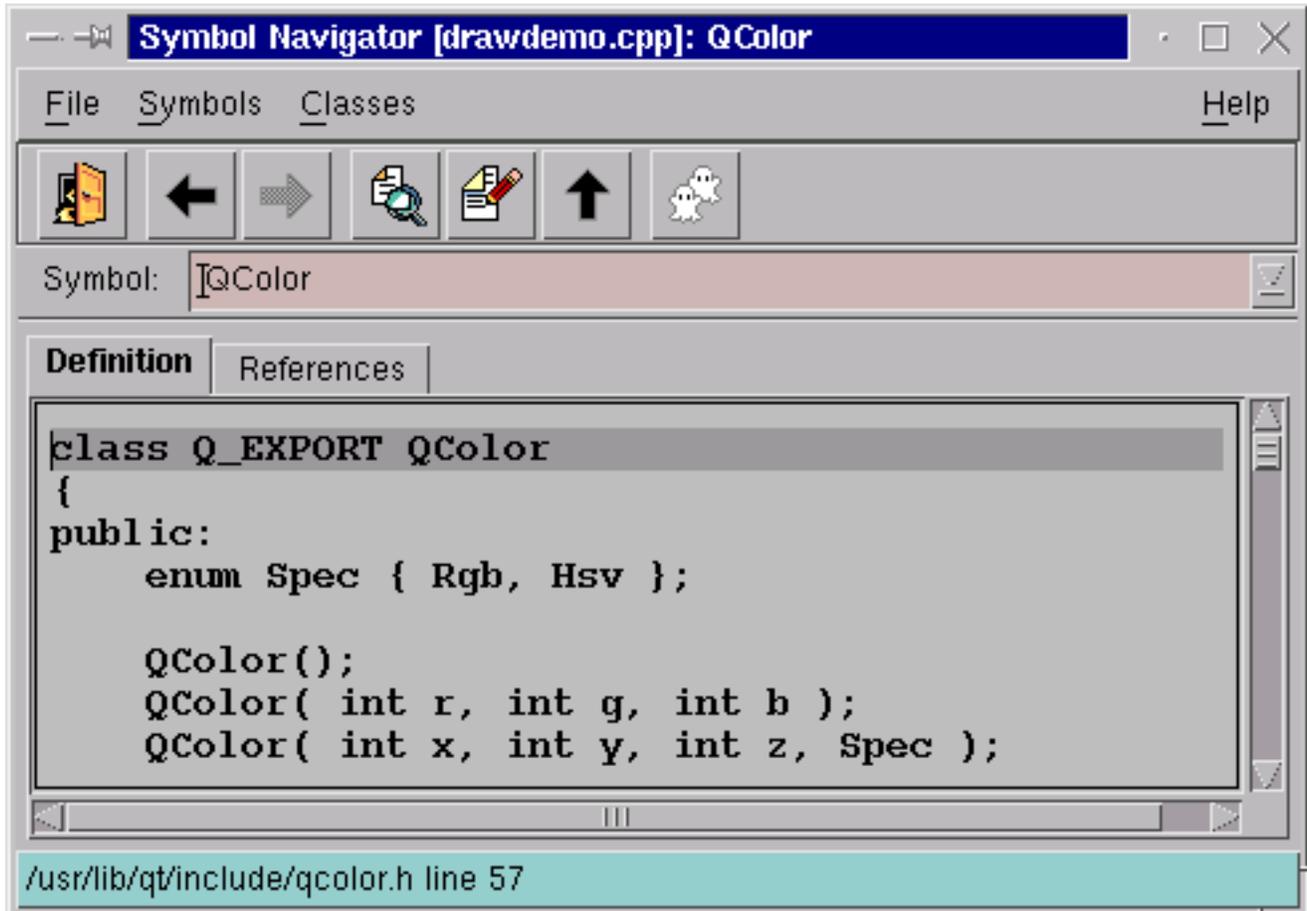
To resolve the conflict in favor of the right part file content, choose **Merge all for right** command from the **Merge** menu. The **Merging result** field will include all the differences found in the right part file.

To add to the resulting file, displayed in the **Merging result** field, the content of a single difference found in the left or right file, click the right mouse button on this difference frame and choose the "<<<<<<<<<" (for the right part) or ">>>>>>>>" (for the left part) command of the context menu. The middle part of the window will display the right or left part frame content respectively. The combo-box in the bottom of the part will display **merged** value.

Last modified: Monday, 01-Apr-2002 09:05:26 EST

Symbol Navigator

The Symbol Navigator allows symbol browsing within the confines of the module loaded into the editor and its include files. It is launched by selecting **Popup Symbol Navigator** (or **Show Symbol/Show Symbol in Context**) in the pop-up menu of the [Editor Window](#).



The Navigator window is divided into next areas:

Navigator toolbar

Used for quick navigation:

- **Close** - closes current **Navigator** window.
- **Previous symbol** - shows the previous viewed symbol.
- **Next symbol** - shows the next symbol in history.
- **Lookup** - looks for symbol, with the name, entered in the **Symbol:** text field.
- **Open in editor** - opens the definition of current symbol in [Editor Window](#).
- **Find next** - shows the next symbol, with the same name, defined early.

- **Clone** - clones current window, the cloned window inherits history and current symbol definition.

Input field with combobox

It can be used to search for another symbol and walking through history using combobox.

Hint: Clicking to **Symbol:** label clears the text field. It can be very useful for mouse cut/paste operations.

Symbol definition and references

Shows the part of source file, where is located the symbol definition.

References List in separate folder displays the list of symbol references. Double-clicking on a reference line will cause the editor to jump to the corresponding line in the file.

Status line

Shows the name of file and line number, where is symbol is defined.

Menu

File

- **Clone** - clones the Symbol Navigator window.
- **Close** - closes the Symbol Navigator.

Symbols

- **Show Symbol** - displays symbol definition.
- **Show Symbol in Context** - displays symbol definition in context in the [Editor Window](#).

Classes

- **Hidden, Top, Bottom, Left, Right** - displays class tree on specified orientation relatively **Definition** area.
- **Switch Class Tree View** - changes class tree view.

The screenshot shows the Symbol Navigator window for the file `drawdemo.cpp`, specifically for the `QColor` class. The window has a menu bar with `File`, `Symbols`, `Classes`, and `Help`. Below the menu bar is a toolbar with icons for home, back, forward, search, edit, up, and help. The `Symbol:` field contains `QColor`. The left pane shows a tree view of classes, with `QColor` selected. The right pane shows a table of symbols for `QColor`.

Stor	Acc	Type	Name
		Constructor	<code>QColor()</code>
		Method	<code>alloc()</code>
		Method	<code>blue()</code>
		Method	<code>cleanup()</code>
		Method	<code>currentAllocContext()</code>
		Method	<code>dark()</code>
		--	

The `Definition` tab is active, showing the following code:

```
inline QColor::QColor( int r, int g, int b )
{ setRgb( r, g, b ); }
```

The source file is identified as `/usr/lib/qt/include/qcolor.h line 131`.

Last modified: Friday, 15-Mar-2002 11:58:38 EST

SMED Editor

Overview

C-Forge includes a full-featured programmer's editor (SMED - short for Smart Editor), that supports the following features:

- [Client/Server Design](#)
- [Saving Files](#)
- [Returning to Previous Versions](#)
- [Context Highlighting](#)
- [Automatic Indentation](#)
- [Bookmark System](#)
- [Finding text](#)
- [Replacing text](#)
- [Working with Symbol Navigator](#)
- [Drag'n'Scroll](#)
- [Block Selection](#)
- [Integration with the IDE](#)
- [Views](#)
- [Function Panel](#)
- [Interactive Configuration](#)
- [Macro Record/Playback](#)
- [Collapsing Text Areas](#)

Client/Server Design

SMED is based on a Client/Server architecture. The current version creates a single management process for all the user's [Editor Windows](#). This architecture is extremely quick and efficient. The sharing of resources between editor buffers results in the decreased use of memory and greatly improves editor start-ups times.

Saving Files

To save a file you are working on, select **Save** in the **File** menu. SMED is set to automatically save files. When the interval in **Auto Save Delay** text-box of the [custom resources](#) dialog is not zero, the changes you make to a file are saved in a recovery file based on the **Auto Save Delay** interval.

Returning to Previous Versions

To return a file to one of the previous versions select **Revert to File** from the **File** menu. SMED shows [Revert to File](#) dialog that allows selection one of the following versions:

- **Original File** - the copy of file that was loaded to the editor at the beginning of the editing session.

- **Backup File** - the backup copy of the file.
- **Auto Save File** - the result of the most recent [auto save](#) operation.

Select the version that you need and press **Revert** button.

Context Highlighting

Context Highlighting uses color and font styles to distinguish different parts of a file you are working on. **Comments**, **reserved words**, **symbols**, **constants** and **preprocessor directives** are shown in different colors so the code is more readable. SMED supports three different modes of context highlighting:

- **Dumb** - no context highlighting.
- **Default Language Context Highlighting** - 6 colors and 2 fonts, comments get their own font.
- **Extended Language Context Highlighting** - 6 colors and 6 fonts - all contexts get different fonts and colors.

Context Highlighting is turned on/off by **Language Context Outlining** option in [Custom Resources](#) dialog. The type of highlighting is specified by **Custom All** option in the [Fonts](#) dialog.

To change the font of a context:

1. Select **Custom Resources** from the **Options** menu.
SMED shows [Custom Resources](#) dialog.
2. Select the button **Set Fonts**.
SMED shows [Fonts](#) dialog.
3. Select one of the six context types.
4. In the left part of the dialog change the parameters of the font.
5. To save the formatting for the current editing session select **Set Fonts**. To save the formatting for all the editors of the display, select **Set Fonts in All Editors** button.
6. Select **Dismiss Dialog** or **Save Customization** button in [Custom Resources](#) dialog.

Automatic Indentation

There are four indentation modes available.

- No indentation.
- **Auto indentation** - pressing the **Enter** key brings the cursor to the same tab-stop on the following line.
- **Language Specific** - the new line is indented according to the indentation and brackets of the previous lines in the style of current language, loaded in Editor. Pressing the **Tab** key or **Ctrl+T** indents the current line in accordance with the previous lines; the cursor does not need to be at the beginning of the line. To indent the selected block, press **Ctrl+T**.
- **WysiWyg** - the entire file is indented, the new lines are indented automatically.

The WysiWyg indentation mode described at the next two images.

Now, source is indented correctly:

```

if (ready)
    return new;
else |
    printf(msg);
    return NULL;
}

int main(int argc, char **argv) {
    unsigned char *data;
    char *str;

```

But after inserting opening brace, ALL following source is re-indenting according to the indentation rules.

Note, that at the same moment, following functions are trying to occupy the true position:

```

if (ready)
    return new;
else { ← Inserted '{'
    | printf(msg);
    | return NULL;
} ← One level
← Error recovery
int main(int argc, char **argv) {
    unsigned char *data;
    char *str;

```

The indentation mode is specified in the formatting section of the [Custom Resources](#) dialog.

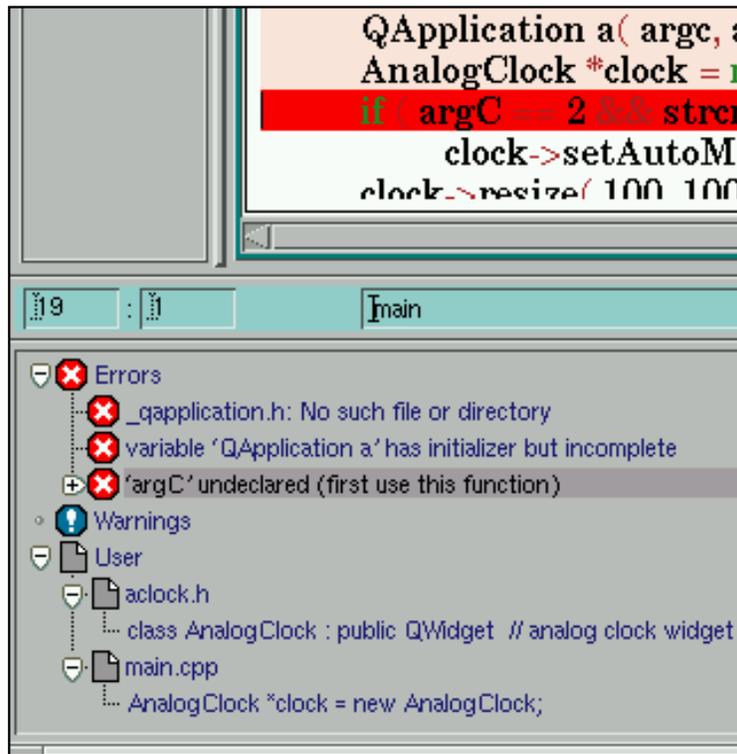
Bookmark System

Bookmarks are used to mark the lines in a file, so they can be found easily.

To bookmark the current line press **Shift + Enter**. Setting a bookmark changes the background color of the line. The bookmarks can be also loaded to the Editor Window from the [Log-window](#), if **Correct Error(s)** or **Edit file** was selected in **File** menu of the Log-window.

Bookmarks are attached to the lines, not the line numbers, so they are unaffected by deletes or inserts of lines.

Selecting **Show Bookmarks Pane** in the **Options** menu or pressing **Show Bookmarks Info** button opens the section at the bottom of the editor window that displays all the bookmarks in the files edited.



Bookmarks pane consists of a tree with three root folders - **Errors**, **Warning** and **Users**. Compilation errors and warnings are loaded into the **Errors** and **Warning** folders from the [Log-window](#). **Users** folders contains user-defined bookmarks. Bookmarks in this folder can be grouped by file by turning on the **Group Bookmarks by File** option.

Selecting a bookmark in the tree and pressing *Enter* cause file containing bookmark to be opened and cursor set to this bookmark. Keyboard focus remains in the bookmarks tree. Double-click on the bookmark behaves the same way, but the keyboard focus moves to the file view panel. Pressing *BackSpace* or *Delete* when the bookmark tree has keyboard focus cause selected bookmark to be removed.

Finding text

To find a text:

1. Select **Search** from the **Search** menu of the [editor window](#).
SMED shows [Search dialog](#).
2. In the **Search Text** box, enter the text you want to search for.
3. Specify search options.
4. Click **Find Next** button.
SMED will highlight the first matching of the text.

If the text is not found, and you start finding text not from beginning of file, SMED will show the following message:



Select **Yes** to start searching from the beginning/end of the text.

Select **No, Go Back** to return to the position that was current when Find Dialog was risen.

Select **No** to close the dialog.

5. To find the next matching select **Search/Replace Next** in the **Search** menu.

Replacing text

To replace some text with another text:

1. Select **Replace** from the **Search** menu of the [editor window](#).

SMED shows [Replace dialog](#)

2. In the **Search Text** box, enter the text you want to search for.
3. In the **Replace Text** box, enter the replacement text.
4. Specify search options.
5. Click **Replace Next** button.

SMED will highlight the first matching of the text and prompt you to confirm the replacement.

6. If the text is not found, SMED will show the message similar to the [message](#) for the **Find** dialog.
7. To find the next matching select **Search/Replace Next** in the **Search** menu.

To replace all the matching select **Replace All** button.

Working with Symbol Navigator

[Symbol Navigator](#) allows symbol browsing within the confines of the module currently loaded into the editor and its include files. It allows you also to see the definition of symbols in context and to get the list of all the lines that include a certain symbol.

Working with the Symbol Navigator you should remember that a highlighted area is loaded to the Navigator. If nothing is highlighted, then the symbol you click on or the current symbol is loaded to the navigator.

To pop up the Symbol Navigator select **Popup Symbol Navigator** from the **Tools** menu, or click with the right mouse button on the text and select **Popup Symbol Navigator** in the pop-up menu.

To find a symbol definition select **Show Symbol** from the **Tools** menu or popup menu.

To see the definition in context select **Show Symbol in Context** from the **Tools** menu. In this case the part of the text that

includes the definition of the selected symbol will be displayed in the Editor Window.

You can use internal [Symbol Navigator](#) to browse the symbols.

To view the definitions of two different symbols:

1. Highlight the symbol in the text and select **Show Symbol** from the **Tools** menu to get its definition.
2. Select **Clone** from the **File** menu to open the clone copy of the Symbol Navigator.
3. Enter the another symbol to the **Symbol:** text-field of the Symbol Navigator and press **Enter**.
4. You can walk through history of symbols using buttons with left & right arrows.

Drag'n'Scroll

Using the scroll-bars for the large files is not very effective because even a slight movement of the scroll box causes the considerable scrolling. So, in addition to regular scroll-bar SMED supports two Drag'n'Scroll modes that allow to *grab* the editor window and scroll text in the direction indicated by the current mouse position.

Using Ctrl+Shift and the middle mouse button *activates* accelerated scrolling. This action scrolls the screen continuously at the speed determined by the current distance and direction of the mouse pointer, relative to the center of the editor window.

Un-Accelerrated scrolling is activated by using **Shift** and the middle mouse button. The screen is scrolled in the direction determined by the mouse pointer movement.

Block Selection

SMED supports two modes of block selection, line and rectangular. By default blocking is performed on a line-by-line basis. However, text can be selected as a rectangular block by pressing the **Ctrl** key and holding the left mouse button to regulate the size of the rectangle desired.

```

if (evh = FindHandler (dpy, w, event_type, w
    return (int) evh;
evh = malloc (sizeof (EventHandlerRec));
evh -> dpy = dpy;
evh -> event_window = w;
evh -> event_type = event_type;
evh -> window_offset = window_offset;
evh -> handler_proc = handler;
evh -> data = data;
evh -> next = HandlerList;
HandlerList = evh;
return (int) evh;
}

```

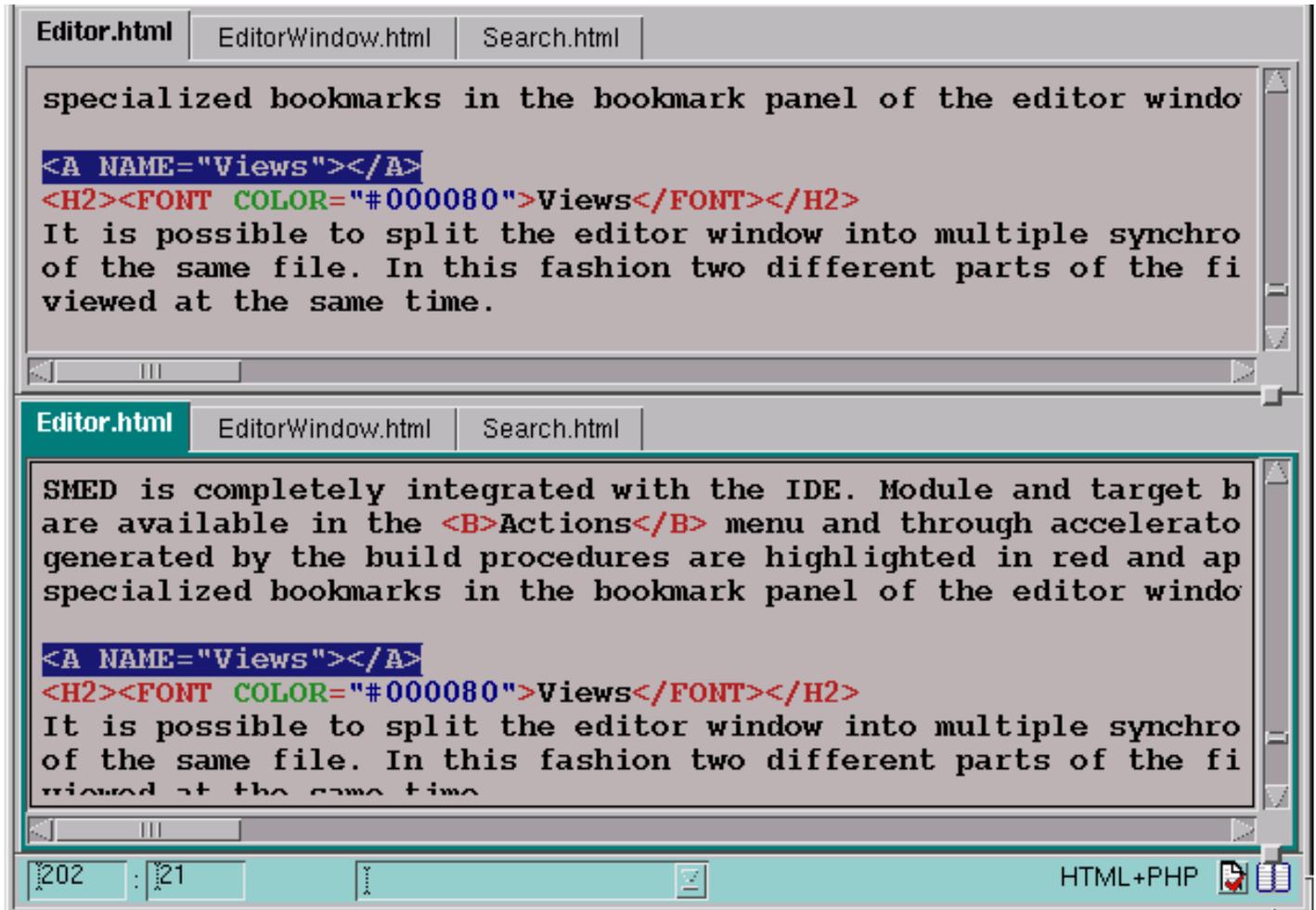
You can cut, copy, paste and move rectangular blocks the same way you work with regular blocks. You can paste a regular block as a rectangular block using **Ctrl** and middle mouse button.

Integration with the IDE

SMED is completely integrated with the IDE. Module and target build actions are available in the **Actions** menu and through accelerators. Errors generated by the build procedures are highlighted in red and appear as specialized bookmarks in the bookmark panel of the editor window.

Views

It is possible to split the editor window into multiple synchronized views of the same file. In this fashion two different parts of the file can be viewed at the same time.

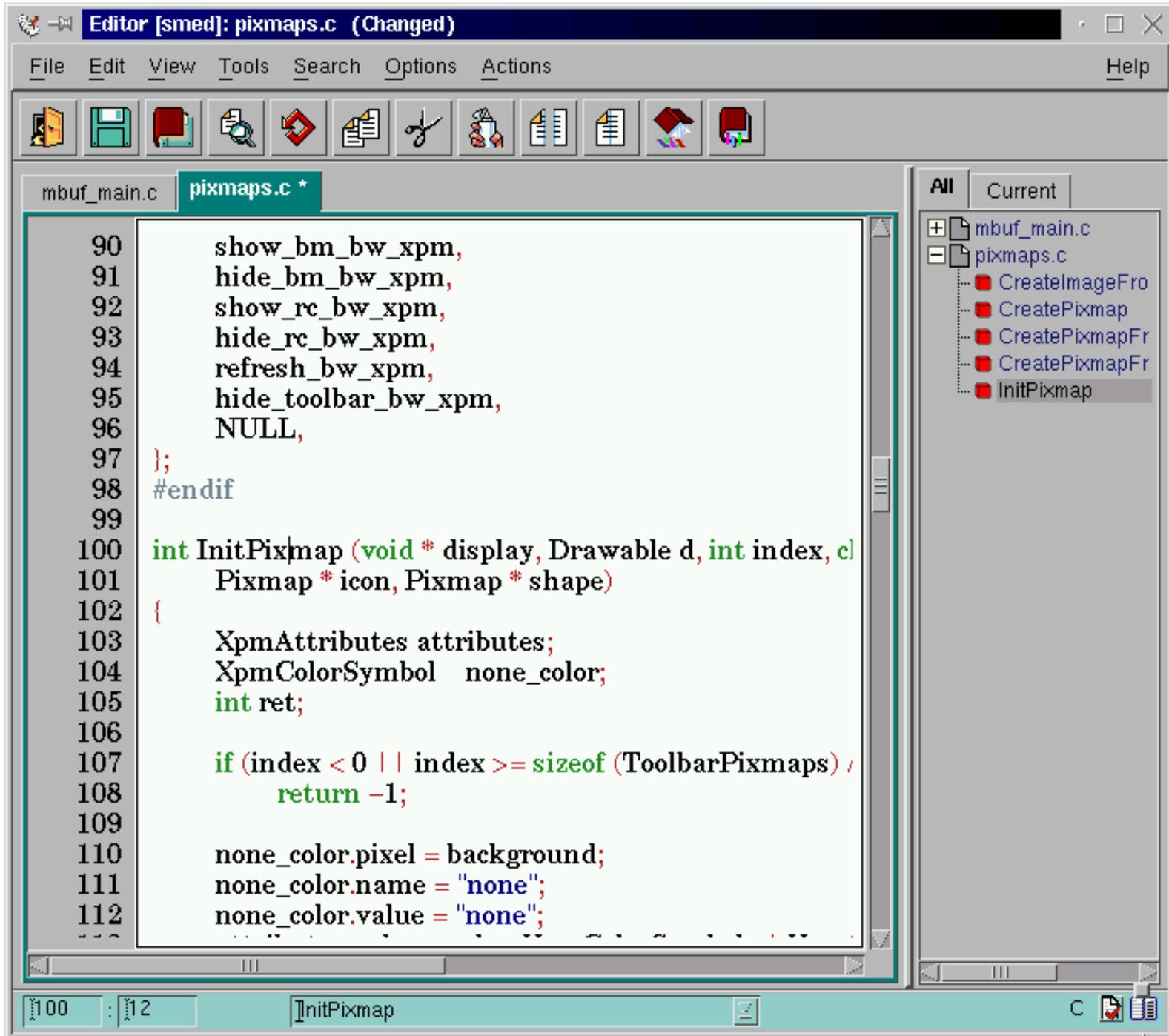


To split the Editor Window select **Split Current View** from the **View** menu or press **Split Current View** button.

To withdraw the current view select **Remove Current View** from the **File** menu. Selecting **Make One View** button removes all the views except one.

Function Panel

Depending on the customization (see [Options menu](#)) **Function Panel** appears on the top, bottom, left or right side of the text area. **Function Panel** is on the right-hand in the image below:



Function Panel consists of two folders - **All** and **Current**.

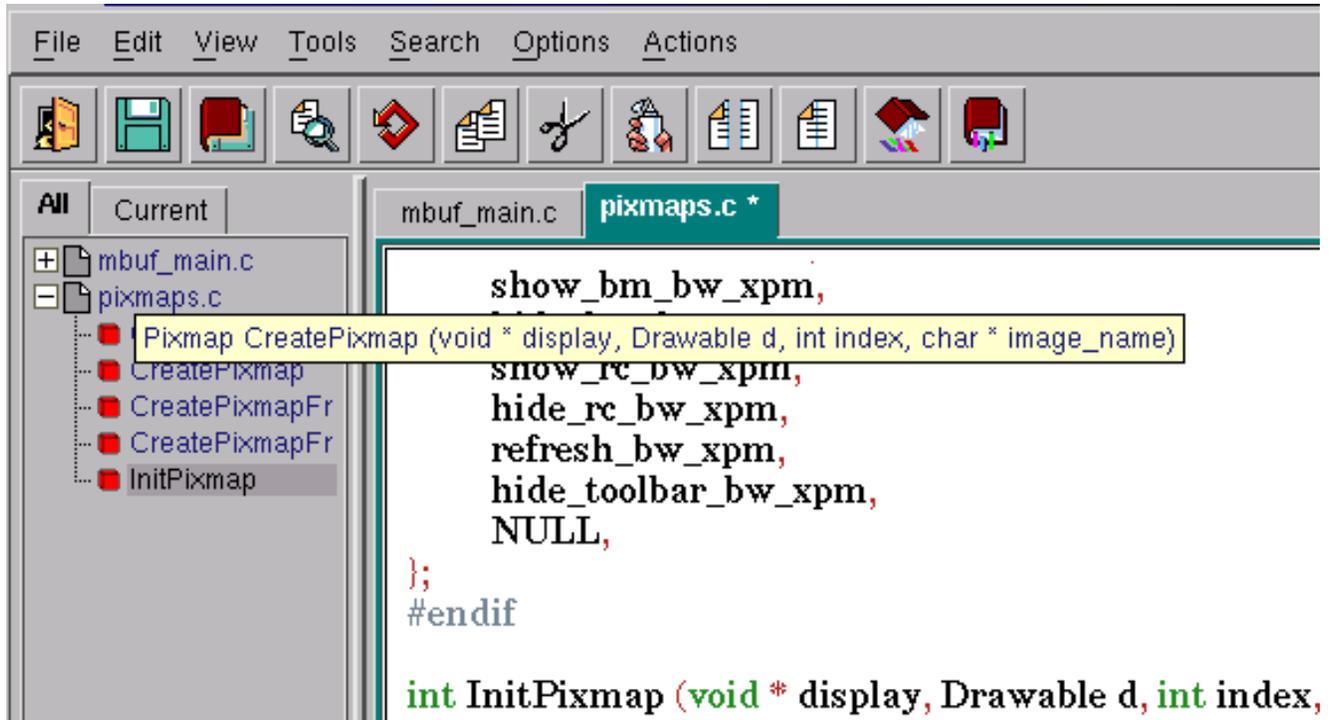
Folder **All** contains a tree with all files opened in this editor window. Each file node contains a list of objects defined in this file.

Folder **Current** contains a list of objects defined in the file that is "current" in this editor window.

Double clicking on the file or function node in the trees makes this files current in the editor window and positions cursor at the beginning of the function.

Pressing tab will cause the focus to shift from the function tree to the text area.

Pausing the mouse pointer over a function node for a little while causes a floating tip with this function's prototype to appear.



Interactive Configuration

You can specify a number of SMED options in the [Custom Resources](#) dialog. To open it, select **Custom Resources** from the **Options** menu.

In addition, toolbar and status line can also be disabled from **Options** menu.

The editor window geometry is saved automatically.

Macro Record/Playback

To save a sequence of keystrokes as a macro:

1. Select the **Macro** from the **Edit** menu in the [Editor Window](#).
SMED shows [Editing Macros](#) dialog.
2. Select **Start Record** button and perform a sequence of actions using the keyboard.
Red flashing **R** letter at the status bar of the Editor window indicates the recording mode.
3. To stop recording, press Stop button at the status bar of the Editor window.
SMED shows [Name&Accelerator](#) dialog.
4. Enter the name of a macro and its accelerator, then press **OK** button.

To run a macro, select its name from the **Edit** menu in the [Editor Window](#) or select the macro in the [Editing Macros](#) dialog and press **Execute Macro** button.

Last modified: Monday, 01-Apr-2002 09:05:26 EST

Collapsing Text Areas

SMED has possibility to hide some pieces of the file to help programmer focus on the text he works on currently. Text area which is to be collapsed may be selected text or block between matching brackets or language object such as function, method or class. Collapsed areas may include one another or even overlap. Collapsed areas are saved per file when editor closed and restored when file is opened again.

```

QMetaObject *AnalogClock::metaObj = 0;
static QMetaObjectCleanUp cleanUp_AnalogClock;

#ifdef QT_NO_TRANSLATION
▶ Method 'AnalogClock::tr' (31:37)
#ifdef QT_NO_TRANSLATION_UTF8
▼ QString AnalogClock::trUtf8( const char *s, const cha
{
    if ( qApp )
        return qApp->translate( "AnalogClock", s,
    else
        return QString::fromUtf8( s );
}
#endif // QT_NO_TRANSLATION_UTF8

#endif // QT_NO_TRANSLATION

▶ Method 'AnalogClock::staticMetaObject' (50:80)
▶ Method 'AnalogClock::trUtf8' (82:86)

```

Collapsing is done by *Collapses* buttons in the **View** pulldown menu:

Collapse <u>A</u> ll	Ctrl-Alt-a
<u>E</u> xpand All	Ctrl-Alt-e
Remove All Collapse <u>M</u> arks	Ctrl-Alt-o
Collapse <u>S</u> election	Ctrl-Alt-c
Collapse Current <u>B</u> lock	Ctrl-Alt-b
Collapse Current <u>F</u> unction	Ctrl-Alt-f
<u>T</u> oggle Current Function Mark	Ctrl-Alt-t
<u>R</u> emove Collapse Mark	Ctrl-Alt-m
Function	▷
Class	▷
Operator	▷
Method	▷

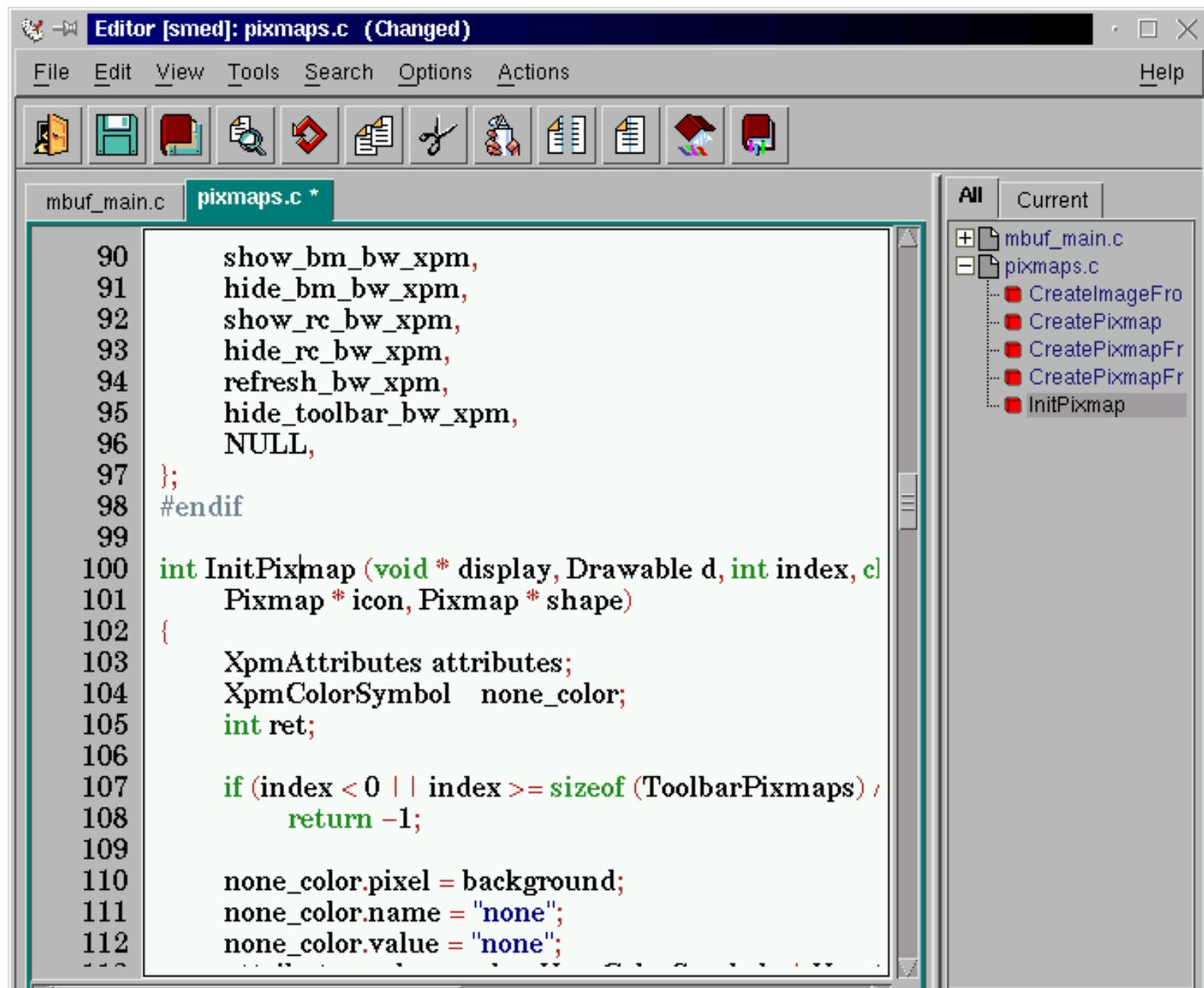
- **Collapse All** collapses all language specific objects in the file
- **Expand All** opens all collapsed areas
- **Remove All Collapse Marks** opens all collapsed areas and removes all collapse marks
- **Collapse Selection** collapses selected text
- **Collapse Current Block** collapses block between nearest to the cursor matching brackets
- **Collapse Current Function** collapses current function, if there is such a object in the language
- **Toggle Current Function Mark** Close/Open current function
- **Remove Collapse Mark** remove collapse mark and hence remove collapsed area
- Below are the language specific cascade buttons, which opens menus for the each language object.

Editor Window

- [Overview](#)
- [Menu Bar](#)
- [Tool Bar](#)
- [Status Bar](#)

Overview

The Editor window is the main window of the [SMED](#) editor. It holds into separate folders all sources, which are currently being edited or viewed. A row of tabs at the top of the panel represents the list of buffers in the "stack". Clicking on a tab with the mouse, brings the corresponding buffer (file being edited or viewed) to the top. For more information regarding accelerator keys that help to switch between folders, see [Folders](#) section.





The source code window displays the text of the file currently being edited. For more information regarding the Editor Window please refer to the section on [SMED Editor](#).

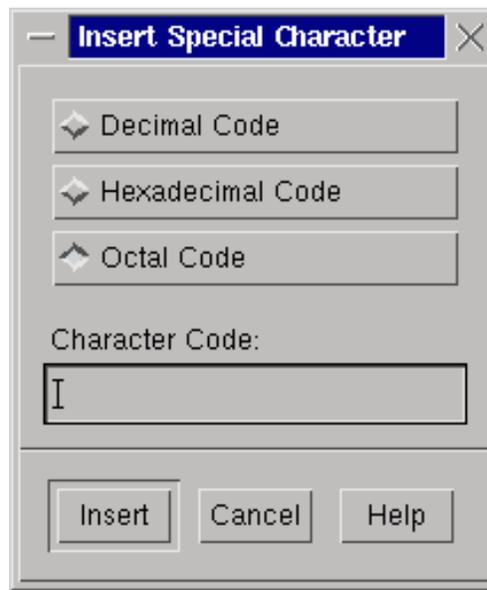
Menu Bar

File

- **Insert** - opens [File Selection Dialog](#) and inserts the selected file at the current cursor position.
- **Revert to Files...** - [reverts](#) to the previously saved file, backup file or auto saved file.
- **Print File** - print current file to printer.
- **Save All Buffers** - writes changes in all buffers to disk.
- **Save Current Buffer** - writes changes to disk.
- **Save To File** - writes current buffer to file with other name. The name and context of current file is unchanged.
- **Open File** - shows [File Selection](#) dialog and opens selected file in new buffer.
- **Close Current** - closes the current file.
- **Close All** - closes all files and exits from SMED.

Edit

- **Undo** - reverses the last command or action (Multiple undo commands are possible).
- **Redo** - reverses the last undo.
- **Cut** - deletes the current selection and copies it to the Clipboard.
- **Copy** - copies the current selection to the Clipboard.
- **Paste** - inserts the contents of the Clipboard at the current cursor position OR replaces the current selection.
- **Select All** - selects all text in the current file view.
- **Convert to Upper Case** - changes all selected text to capital letters.
- **Convert to Lower Case** - changes all selected text to lowercase letters.
- **Insert Character** - opens the next dialog and inserts character with entered code into text.



- **Complete Word** - completes word that is currently left from cursor. Completion performs the word search through current buffer and then all used include files. Search stops on finding 32 symbols. If there is more than one symbol found, popup menu with alphabetically sorted symbols is shows up and user selected word completes the current word.
Note: The default include path for current language must be specified in [Parsers Tool](#).
- **Macro** - opens the [Editing Macros](#) dialog.
- **Run Macro** - displays the list of all currently recorder macros and allows their playback.
- **Language Mode** - submenu, changes the current syntax highlight and indentation mode "on the fly". **SMED** remembers this mode for given source.
- **Edit Mode** - submenu, changes current keybinding mode.
- **Run Macro** - optional submenu which holds all your current available macros.

View

- **Split Current View** - [splits](#) the current view into two.
- **Change Orientation** - changes orientation of the split panels from vertical to horizontal and vice versa.
- **Remove Current View** - removes the current view.
- **Navigate to the Left**- Moves keyboard focus in the following sequence: file view - bookmarks tree - object list - file list - file view.
- **Navigate to the Right**- Moves keyboard focuse in the opposite direction.
- **Navigate Up**- Moves keyboard focus in the split panels if focus was in a file view and move it in the following sequence: bookmarks tree - file list - object list - bookmarks tree if keyboard focus is in one of these trees.
- **Navigate Down**- Moves keyboard focuse in the opposite direction.
- **Goto Current File View**- Moves keyboard focus from bookmarks tree or objects list or files list to the file view.
- **Collapse All** - collapses all objects in the current file.
- **Expand All** - expands all collapsed blocks.
- **Remove All Collapse Marks** - expands all collapsed blocks and remove all collapse marks.
- **Collapse Selection** - collapses the current selection.
- **Collapse Current Block** - collapses the current { ... } block.
- **Collapse Current Function** - collapses the current object (function).
- **Toggle Current Function Mark** - toggles the collapse mark on current function.
- **Remove Collapse Mark** - removes the current collapse mark.

If there is more than one type of collapsed objects for current language, for each object the submenu with the following items are creates:

- **Collapse Current** - collapses the current object with specified type.
- **Collapse All** - collapses all objects of specified type.
- **Expand All** - expands all objects of specified type.
- **Remove Marks** - removes collapse marks from all objects of specified type.

Tools

- **Manual Page for Symbol** - shows the [Manual page](#) for symbol under the cursor.
- **Filter** - applies a filter to selected text.

Search

- **Search** - opens the [Search](#) dialog to search for the selected string.
- **Replace** - opens the [Search and Replace Dialog](#).
- **Search/Replace Next** - finds and selects the next occurrence of the text specified in the [Search Dialog](#) or [Search and Replace Dialog](#).
- **Previous Bookmark** - moves the cursor to the previous [bookmark](#).
- **Next Bookmark** - moves the cursor to the next [bookmark](#).
- **Popup Symbol Navigator** - launches the [Symbol Navigator](#).
- **Show Symbol** - opens the [Symbol Navigator](#) using the definition of the currently selected symbol or symbol under cursor.
- **Show Symbol in Context** - jumps to the selected symbol's definition. (Opens a new editor window if necessary)
- **Flash Matching Brackets** - flashes the matching block bracket.
- **Jump to Matching Bracket** - moves the cursor to the matching block bracket.
- **Jump to Tag** - performs a search for the symbol under cursor or selection in tag files. The list of tag files to be searched is specified in "Tag" [Tool](#) (the default files to be search are 'tags' and 'TAGS' files in the project root directory). The user, using ctags or etags utilities, should create this file.
- **Previous Function** - moves the cursor to the previous function in the text.
- **Next Function** - moves the cursor to the next function.

Options

- **Clear All Bookmarks** - clears all [bookmarks](#).
- **Show Bookmarks Pane** - opens/closes a panel showing all current [bookmarks](#).
- **Group Bookmarks by File** - creates folders for files being edited in the **User** folder of the bookmarks tree ([bookmarks](#)).
- **Show Status Line** - shows/hides the status bar.
- **Show Toolbar** - shows/hides the tool bar.
- **Show Line Numbers** - shows/hides the line numbers on the left side of the file view.
- **Keep Bak File**- do not remove *bak* file after saving.
- **Functions Panel Layout** - custom [function panel](#) layout.
- **Custom Resources** - opens the [Custom Resources Dialog](#).

Actions

- **Make target** - makes the target that depends on the current file. The user is prompted to select the target if several targets depend on the current file.
- **Run target** - runs the target.
- **Debugs target** - debugs the target.
- **Collect includes** - checks the implicit dependencies.
- **Node info** - shows current [node information](#).
- **Diff Tool** - runs the [Diff Tool](#).
- **Grep Tool** - runs the [Grep Tool](#).
- **Version Tool** - run the [RCS](#) tool.
- **Check in RCS revision** - save the changes and create a new [RCS](#) revision.

Tool Bar

The Editor Tool bar includes the following buttons:



- **Close Editor** - closes the all edited files.
- **Save Changes** - writes current file to disk.
- **Insert File** - opens [File Selection Dialog](#) and inserts the selected file at the current cursor position.
- **Search** - opens the [Search Dialog](#) to search for a selected string.
- **Undo** - reverses the last command or action (multiple undo's are possible).
- **Copy to Clipboard** - copies the selection to clipboard.
- **Cut to Clipboard** - deletes the current selection and places it into the Clipboard.
- **Paste** - inserts the contents of the Clipboard at the current cursor position OR replaces any current selection.
- **Split Current View** - [splits](#) the current view into two.
- **Remove Current View** - removes the current view.
- **Clear All Bookmarks** - deletes all [bookmarks](#).
- **Show Bookmarks Info** - opens a panel showing all current [bookmarks](#).
- **Indent Line or Selected Area** - indents the selected area/current line (available, when [Language Specific Indentation](#) check-box is on).

Status Bar



The Status bar displays cursor position, current function name, language mode name, current editor window for newly open files and flag, that current buffer is modified.

Search and Replace Dialogs

- [Search Dialog](#)
- [Replace Dialog](#)

Search Dialog

Selecting Search in the Search menu of the [Editor Window](#) activates the **Search Dialog**. (See [Finding Text](#)).



Search Text - specifies the text to be found.

Direction - specifies the direction of search.

Search For - specifies one of the following modes of matching:

- **Substring** - searches for occurrences that are whole words or part of a larger word; distinguishes between uppercase and lowercase characters.
- **Whole Word** - searches for occurrences that are whole words and not part of a larger word; distinguishes between uppercase and lowercase characters.
- **Substring Regardless Case** - searches for occurrences that are whole words or part of a larger word; does not distinguish between uppercase and lowercase characters.
- **Whole Word Regardless Case** - searches for occurrences that are whole words and not part of a larger word; does not distinguish between uppercase and lowercase characters.
- **Regular Expression** - [regular expression](#) is entered into **Search Text** field.

Dialog buttons

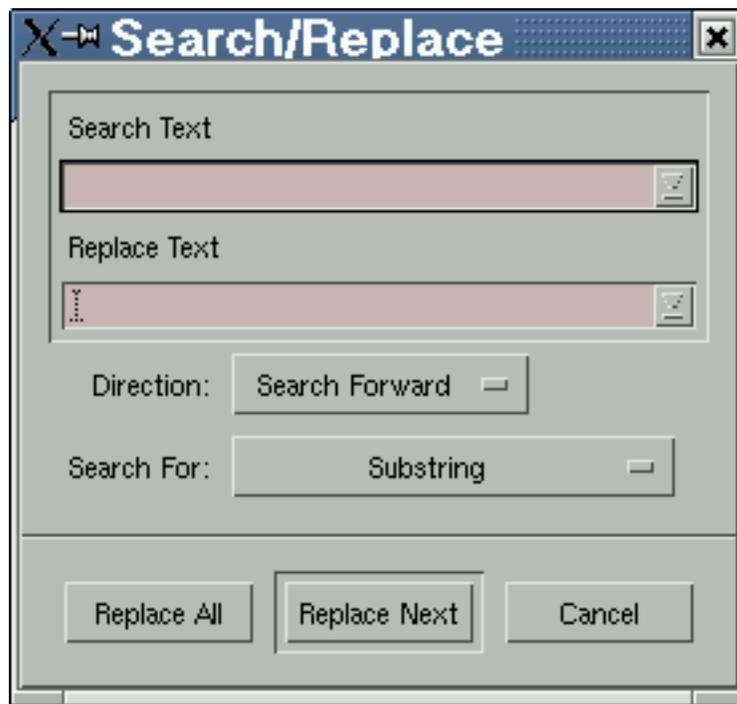
Find Next - finds the next occurrence of the text, when the end/beginning of the text is reached, a warning [message](#) will appear.

Cancel - closes the dialog.

Help - shows this Help window.

Replace Dialog

Replace Dialog is similar to Find dialog. It is also located in the **Search** menu of the [Editor](#). (See [Replacing Text](#)).



It includes all the elements of [Search Dialog](#) and **Replace Text** field that specifies the text to be replaced with.

Dialog Buttons

Replace All - replaces all the matching.

Replace Next - finds the next occurrence of the text. SMED will prompt you to confirm the replacement.

Cancel - closes the dialog.

Help - shows this Help window.

Last modified: Friday, 15-Mar-2002 11:58:38 EST

Custom Resources Dialog

- [Overview](#)
- [Formatting Options](#)
- [Tab Options](#)
- [Keybindings configuration](#)
- [Fonts](#)
- [Colors](#)
- [Other Options](#)
- [Saving Customization](#)

Overview

The **Custom Resources** dialog is used to configure SMED options. It can be accessed from the [Editor Window](#), by selecting the **Custom Resources** option found in the **Options** menu.



Formatting options

Formatting options - allows access of the [indentation modes](#) and turns on/off context highlighting of the text.

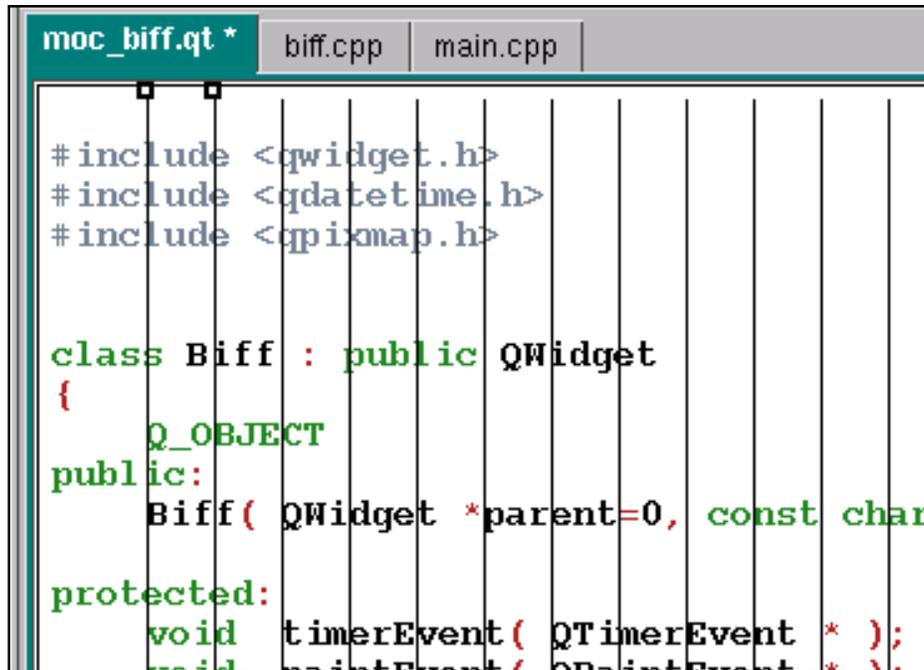
- **Auto Indent** - turns on/off *Auto* [indentation mode](#).
- **Language Context Outlining** - turns on/off context highlighting; the colors and the fonts of the highlighting are specified in [Fonts](#) dialog.
- **Language Specific Indent** - turns on/off *Language specific* [indentation mode](#) and context highlighting.
- **Language WysiWyg Indent** - turns on/off *WysiWyg* [indentation mode](#) and context highlighting.

Tab Options

There are two modes for handling tab stop marks - tab stops aligned on the X-coordinate and tab stops aligned on a character boundary. You can turn on the alignment on the character boundary by pressing on the **Char Width-Aligned Tabs** button. This mode implies use of constant width fonts in the editor. Editor has separate fonts and font selection dialogs for each tab stop mode.

Pressing the **Show Tab Grid** button will enable the tab stop configuration panel that identifies measurements for tabs.

Pressing the button displays currently defined tab stops as sashes running vertically across the editor window.



```

moc_biff.qt *  biff.cpp  main.cpp
#include <qwidget.h>
#include <qdatetime.h>
#include <qpixmap.h>

class Biff : public QWidget
{
    Q_OBJECT
public:
    Biff( QWidget *parent=0, const char
protected:
    void timerEvent( QTimerEvent * );
    void paintEvent( QPaintEvent * );

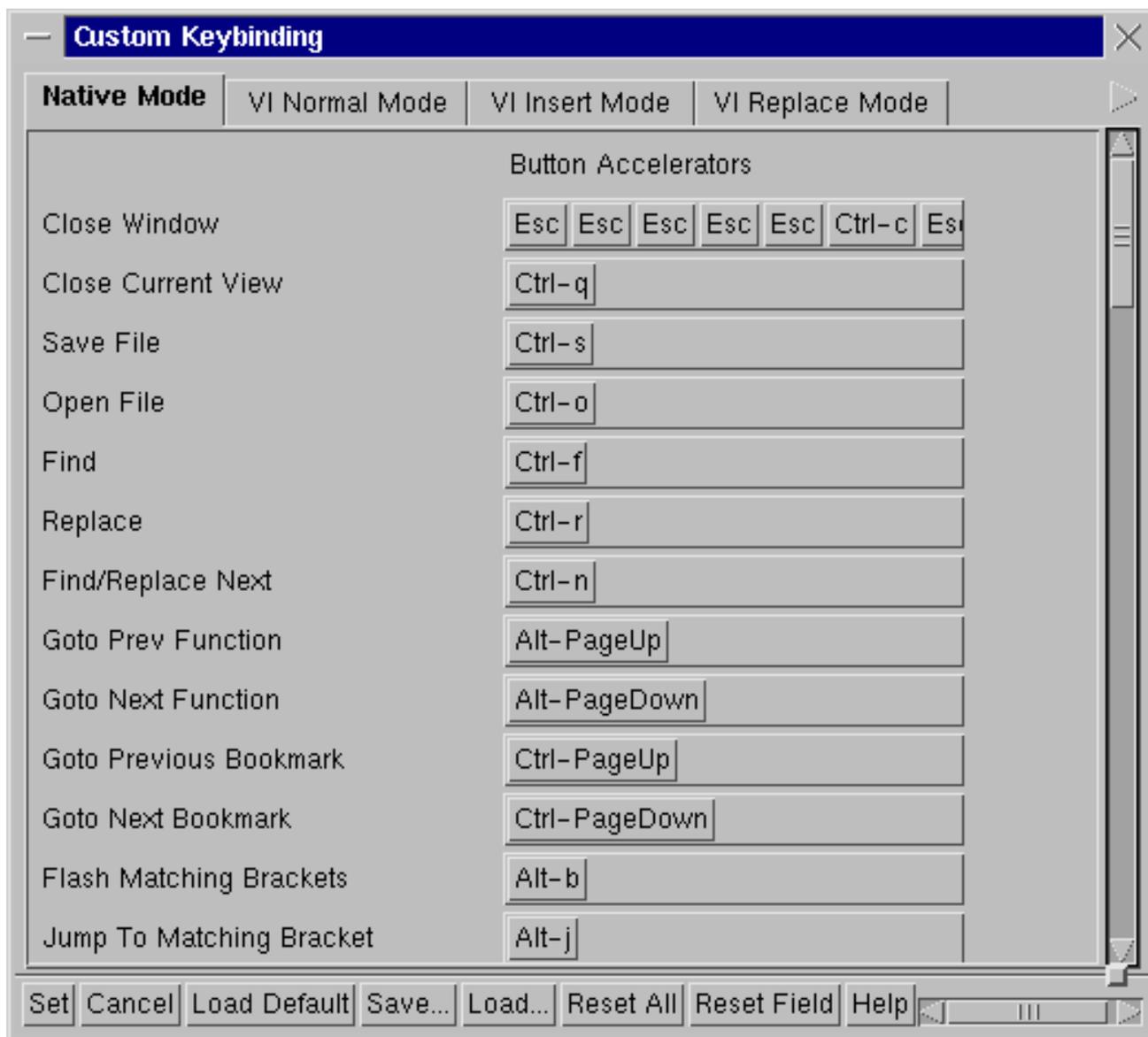
```

Dragging the marks, positioned at the root of every sash, using the left mouse button, can change tab sizes.

Sashes can be added, or removed, by clicking on the **Add** or **Delete** Sash buttons.

Keybinding Configuration

Pressing the **Set Keybinding** button displays the following dialog:



There are two sets of default keybindings: Default **Native mode** and the various **VI** modes.

All menu accelerators and editor action keybindings can be modified from here by selecting an operation in the dialog and pressing a new key combination.

Dialog Buttons

Set - set the current selected mode, applies the changes in keybindings and close the dialog.

Cancel - cancels all the changes and close the dialog.

Load Default - loads default keybindings from a system defaults file.

Save... - saves the keybindings in a file. The location of the file is selected with the [File Selection Dialog](#).

Load... - loads keybindings from a file, the file is selected with the [File Selection Dialog](#).

Reset All - resets all keybindings to their values at the beginning of the configuration session.

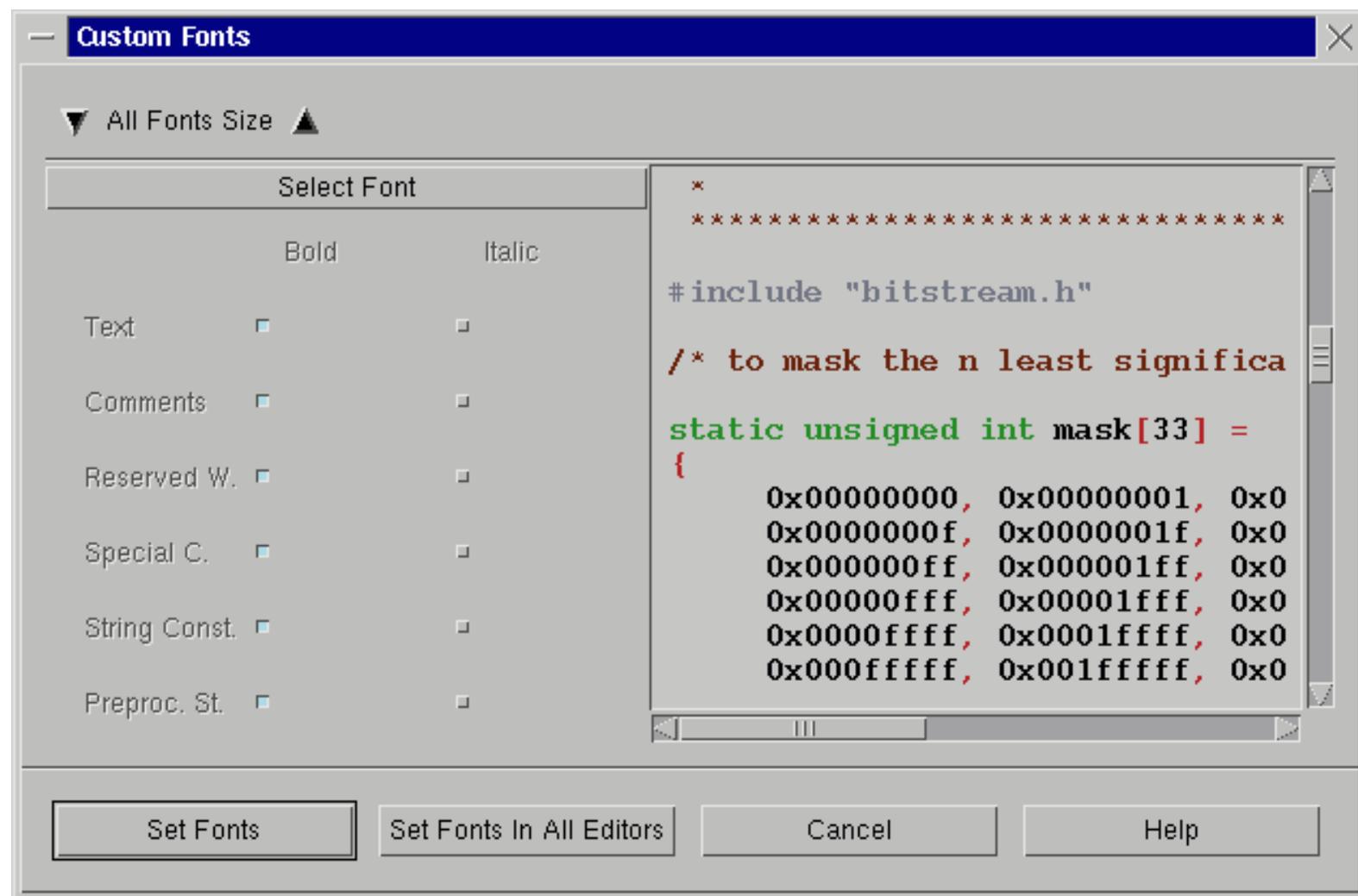
Reset Field - resets the keybinding of the current field to its value at the beginning of the configuration session.

Help - shows this Help window.

Fonts

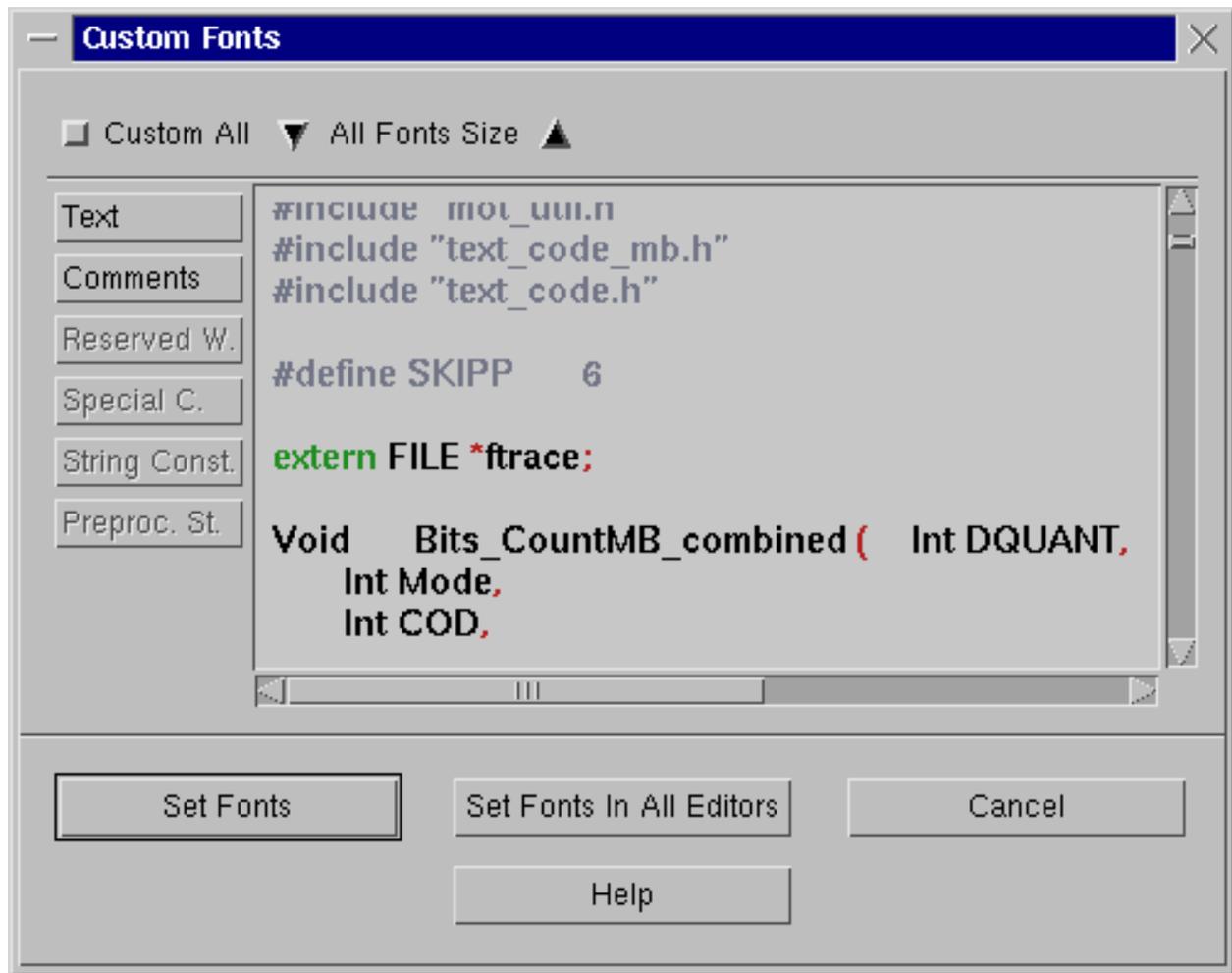
Pressing the **Set Fonts** button displays the following dialogs, depending on the **Char Width-Aligned Tabs** button state:

When **Char Width-Aligned Tabs** button is turned on:



You can select one fixed size font for all contexts but set different font properties to different context. Press **Select Font** button and select font in the [Font Browser](#). Then set font properties for different contexts and press **Set Fonts** button or **Set Fonts In All Editors** to set selected font in all editors on your display.

When **Char Width-Aligned Tabs** button is turned off:



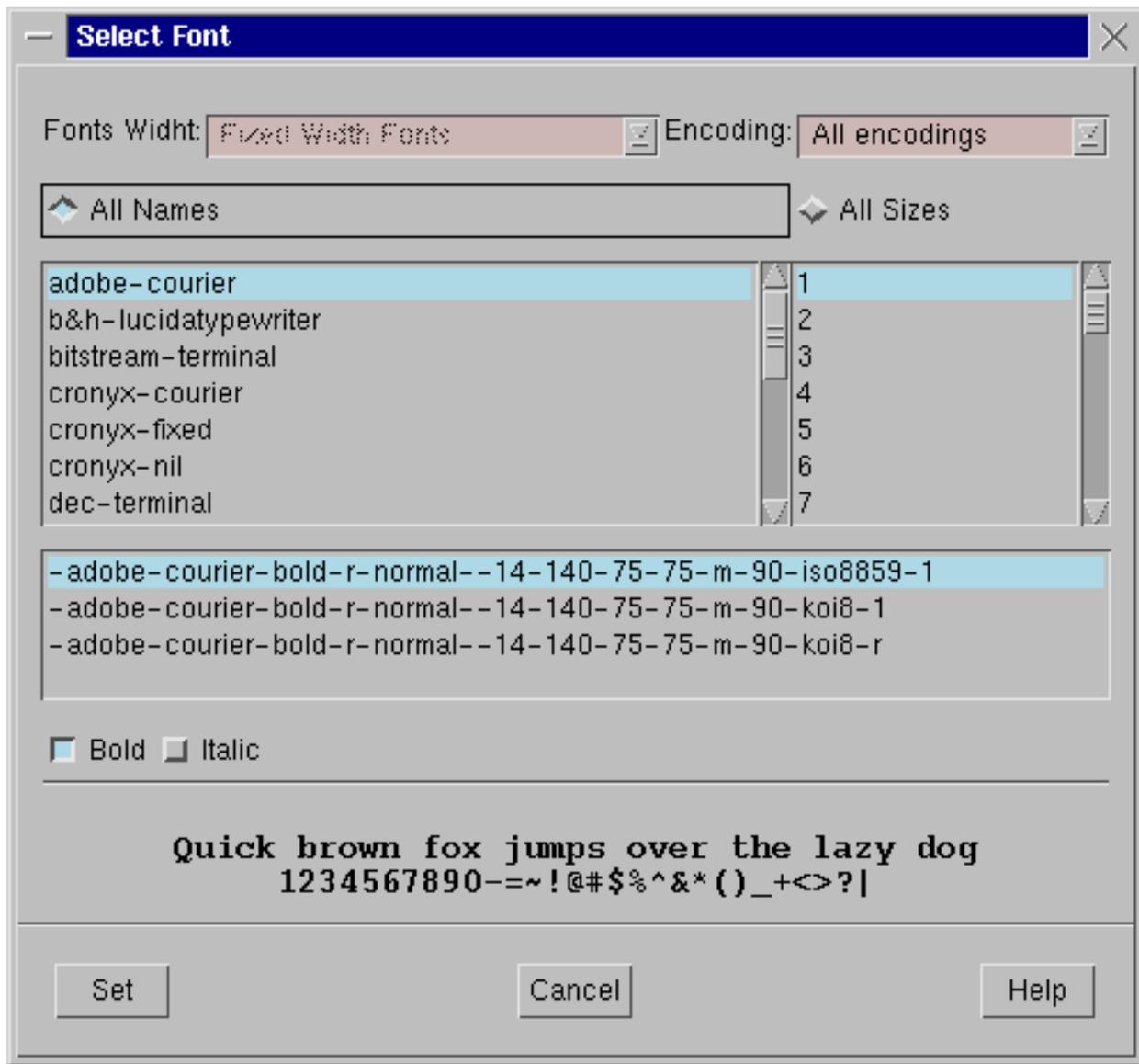
The left part of the dialog includes a group of six options that specify different contexts of a file you are working on: **comments**, **reserved words**, **symbols**, **constants**, **preprocessor directives** and the rest of the text.

If **Custom All** mode is selected, all types of text get their own fonts. This mode is turned off by default and all contexts except **Comments** get the same font.

To select font for the context, press button and select the font in the [Font_Browser](#) dialog.

In both dialogs after selecting of the fonts you can adjust the font size for all contexts by pressing up or down arrows on the left and right sides of the **All Fonts Size** label. Changing of the font sizes will be done depending on how much fonts similar to selected in the font browser are available in your system.

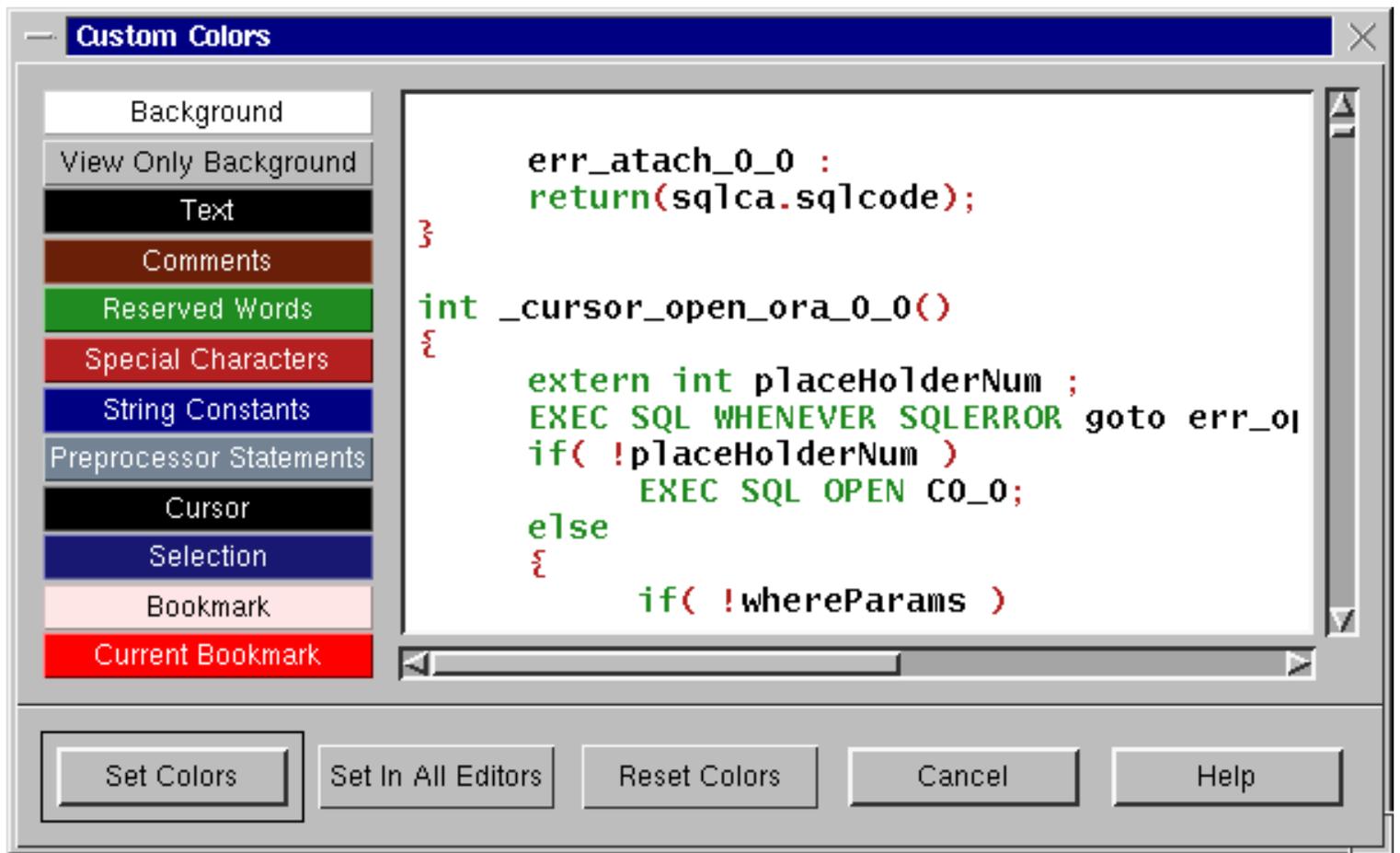
Font Browser



You can choose font face, size and encoding, font width type (fixed width or variable width), see how bold and/or italic variants look like. To continue setting of the selected font press the **Set** button to return to the previous dialog.

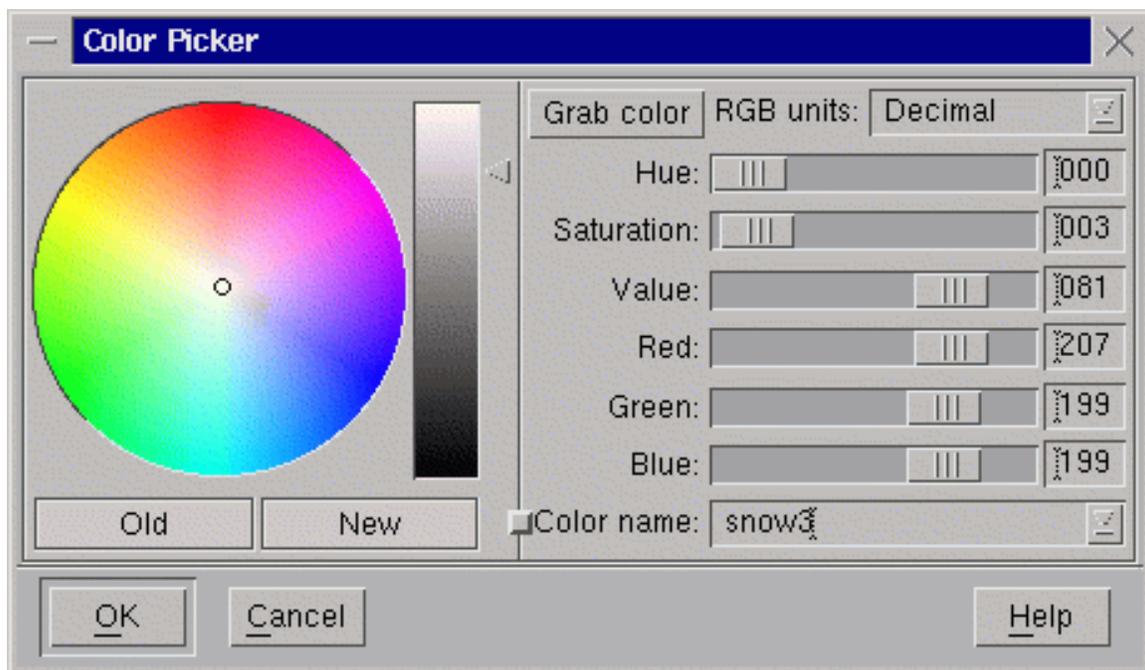
Colors

Pressing the **Set Colors** button displays the following dialog:



The dialog allows modification of colors for various language/SMED components.

Clicking on colored button with the name of color group brings up the color selection dialog:



You can grab any color from the screen, select color from color list (alphabetically sorted color names), or select

a color from the color circle. Hue, Saturation, Value, Red, Green and Blue values can also be modified by dragging their respective sliders or entering numbers, in various formats, in the entry fields on the right-hand side of the dialog.

Dialog Buttons

Set Fonts - applies the changes to the current editor window and closes the dialog.

Set Fonts in All Editors - applies the changes to all currently active editor sessions and closes the dialog.

Cancel - cancels all changes and closes the dialog.

Help - shows this Help window.

Other Options

The **Undo Depth** field controls the maximum undo depth.

A non-zero value placed into the **Auto Save Delay** field enables the automatic update of changes. All auto-saved changes are available in a file with the extension **.auto.bak*.

The **Update Functions Panel Interval** option sets the time in seconds between the last change in the editor and the function list refresh.

The **Brackets Flash Interval** option sets the frequency of matching brackets flashing in tenth of second.

The **Save Last Search/Replace Patterns** option sets the number of patterns that are saved in history.

A non-zero value of **Replace Tabs With Spaces** enables replacing of tabs with given number of spaces. If **Replace Leading Tabs Only** check-box is disabled - all tabs are replaced by given number of spaces.

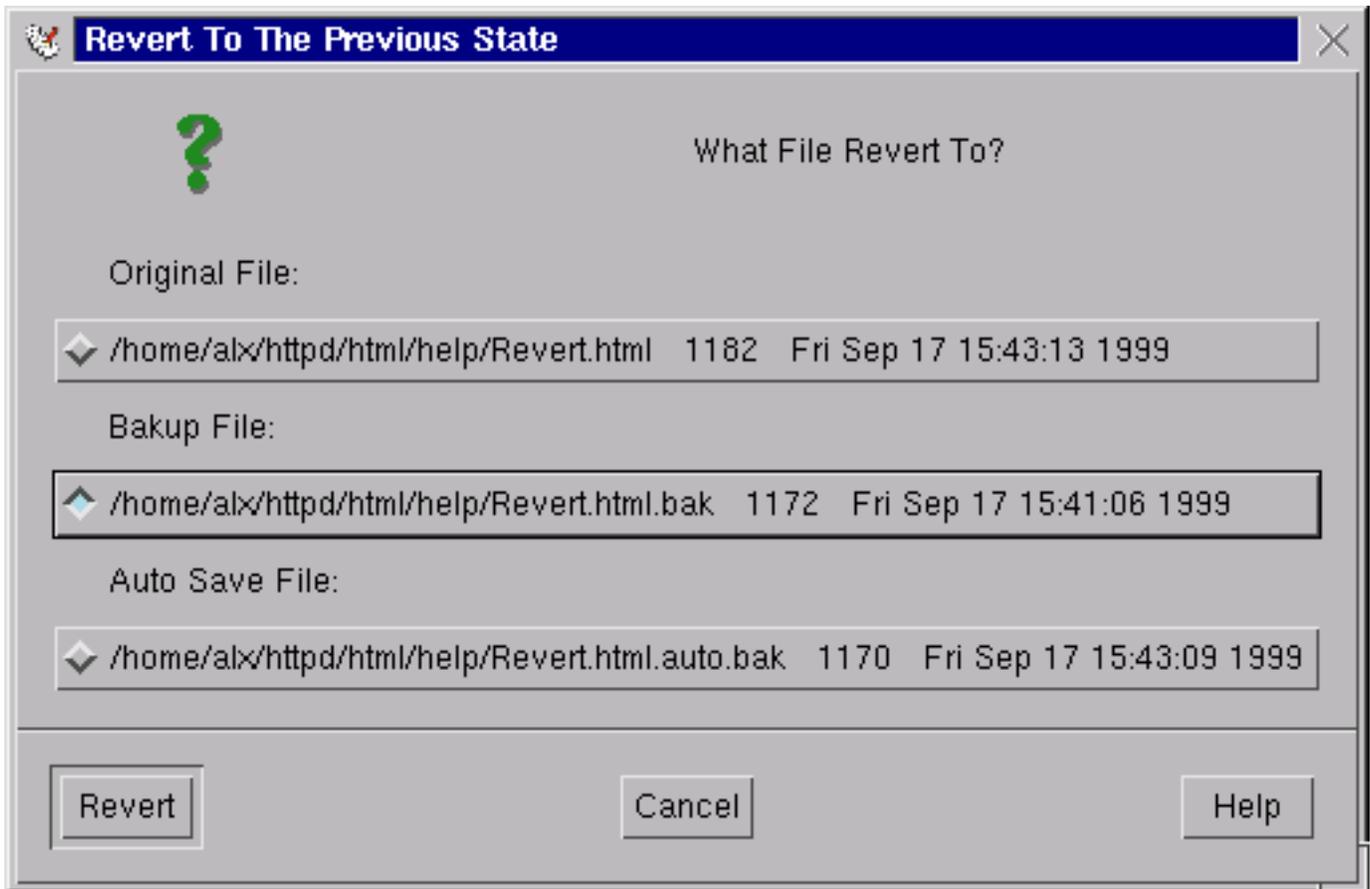
Saving Customization

The changes made in the Custom Resources dialog are immediately applied to the Editor Window. When modifications are complete, press the **Save Customization** button. This will save the modified resources to disk and apply them to all newly opened editor windows.

Pressing **Dismiss Dialog** will save the new customization only for the current editing session.

Revert to File Dialog

To return a file to its previously saved state use the **Revert to File** dialog. This dialog is located on the **File** menu in the [Editor Window](#). This dialog returns a file to a previously saved version:



- **Original File** - the copy of file that was loaded to the editor at the beginning of the editing session.
- **Backup File** - the backup copy of the file.
- **Auto Save File** - the most recent automatically saved copy of the file.

Dialog Buttons

Revert - returns a file to the selected version.

Cancel - cancels the operation.

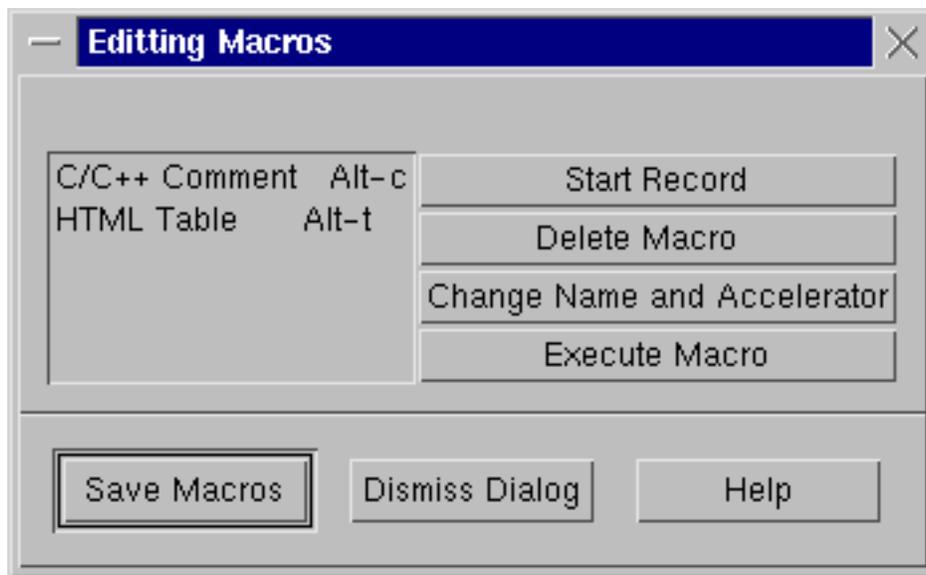
Help - shows this Help window.

Editor Macros Dialogs

- [Editing Macros Dialog](#)
- [Name and Accelerator Dialog](#)

Editing Macros Dialog

The **Editing Macros** dialog is used for [SMED](#) macro maintenance. Selecting **Macro** from the **Edit** menu in the [editor window](#) access it.



The dialog text field of the dialog displays the list of all the macros available in the editor.

Dialog Buttons

Start Record - starts recording a new macro.

Delete Macro - deletes the selected macro.

Change Name and Accelerator - opens [name and accelerator](#) dialog, which allows changing macro name and Shortcut key.

Execute Macro - playbacks the selected macro.

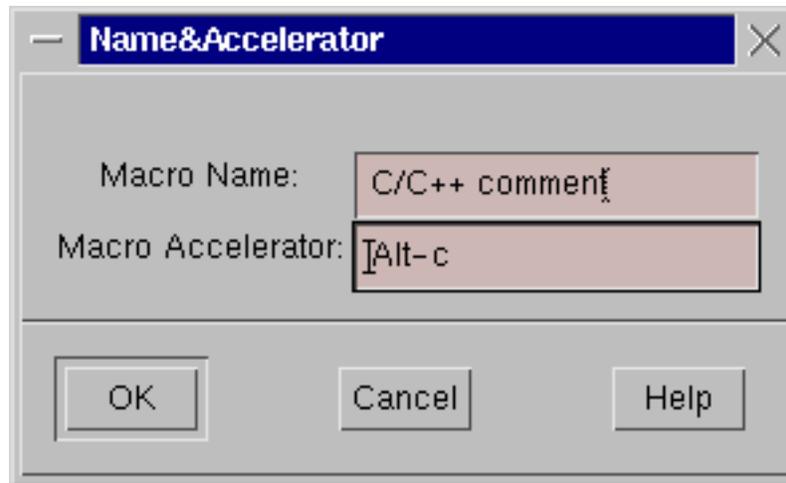
Save Macros - save changes and closes the dialog.

Dismiss Dialog - closes the dialog without saving the changes.

Help - shows this Help window.

Name and Accelerator Dialog

Selecting the Change Name and Accelerator button from the [Editing Macros](#) dialog accesses the name and Accelerator dialog. This dialog also appears when a new macro is recorded.



Macro Name - displays the name of the macro.

Macro Accelerator - displays the accelerator of the macro.

Dialog Buttons

OK - saves new attributes and closes the dialog.

Cancel - closes the dialog without saving the changes.

Help - shows this Help window.

To get more information about macros see [Macro Record/Playback](#) section.

C-Forge 'Drag and Drop' Overview

Many functions in C-Forge can be accomplished through the use of "Drag'n'Drop". Drag'n'Drop can be used on any number of objects by selecting them and simply dragging them, using the middle mouse button, to their destination. Here is summary of all Drag'n'Drop operations allowed in C-Forge.

Drag and Drop inside Project Desktop

Operations between different views:

From:	To:	Description:
DepTree	WrkArea	Reserve (if node is under Revision Control - perform a "check-out" with a lock) and place into working directory
WrkArea	DepTree	Replace (if node is under Revision Control - perform a "check-in")
Disk	DepTree (Target or Source)	Add Source to project
Disk	DepTree (Includes Folder)	Add Include to project
Disk	DepTree (Any Folder except Includes)	Add Target to project

Operations on Dependency Tree:

From:	To:	Action:	Keys:
Source	Target of this source, another source of this target	Rearrange	Btn2
Target	Target's folder, another target of this folder	Rearrange	Btn2
Target, Folder	Any another folder, except parent folder	Move	Shift+Btn2
Source, Target	Any other target	Link	Ctrl+Shift+Btn2
Folder	Any folder	Link (Mirror)	Ctrl+Shift+Btn2

Note: To unlink linked (mirrored) folders, move one of them to parent folder of other..

Operations with drop sites:

From:	What:	Drop site:	Description:
DepTree, WrkArea	Editable node under RCS	Version Control System	View RCS information
DepTree, WrkArea	Editable node NOT under RCS	Version Control System	Place into RCS
WrkArea	Source node	Trash	Discard changes
DepTree	Any node	Trash	Remove nodes from tree
Disk	Any node	Trash	Remove nodes from disk
All	Any	Touch	Touches dropped nodes
DepTree	Any editable node	Edit	Check out nodes and place to editor
WrkArea, Disk	Any	Edit	Edit nodes
WrkArea, DepTree	Source	Build	Build parent of current node
DepTree	Target	Build	Build current target
DepTree	Folder	Build	Build all targets of current folder
Any	All	Info	Show information about current node

Drag and Drop with Diff Tool drop sites

From:	Node:	Description:
--------------	--------------	---------------------

Project Desktop (DepTree, WrkArea)	Any Editable Node	Set this node to compare
Project Desktop (Disk)	Any Node	Set this node to compare
File Selection	Any Node	Set this node to compare
Version Tool, Revisions Tree	Any revision item	Set this revision to compare
Any Text Field		Set this text as filename to compare

Drag and Drop inside Version Tool

From:	To:	Description:
Revision Tree Item	Another Revision Tree Item	Show differences between versions usind Diff Tool

Last modified: Monday, 14-May-2001 07:03:19 EDT

Help/Man Browser

C-Forge Help/Man Browser allows you to access internal help system and contents of system manual pages.

Internal help

Internal help can be called from any place of C-Forge on pressing **F1** key or selecting the appropriate topic from **Help** menu.

Toolbar

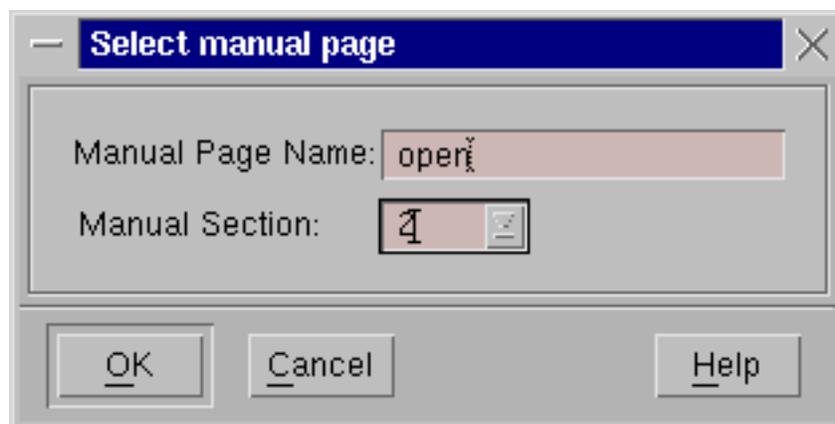
Toolbar is useful for quick navigation through help history contents.



- **Backward** - returns to previous help page in history list
- **Forward** - moves forward to next page in history list
- **Top help page** - goes to the top help page
- **Previous page** - goes to previous (on context) help page
- **Next page** - goes to next (on context) help page

System manual pages

To view the contents of on-line manual page press **Alt+M** in Help/Man Browser window or select text in [Editor Window](#) and select **Manual Page for Symbol** from popup menu.



Manual Page Name - the name of page to view

Manual Page Number - the manual section to lookup for **Name** first

Last modified: Wednesday, 11-Jul-2001 07:07:29 EDT

Frequently Used Interface Components

The Code Forge User Interface makes use of a wide range of standard Windows components including menus, buttons and tool bars. Below is a detailed description of each of these components and what they do.

- [Window Menu](#)
 - [Client Control Panel](#)
- [Regular Expression Filters](#)
- [Trees](#)
- [Text Input Dialog](#)
- [Title Bar](#)

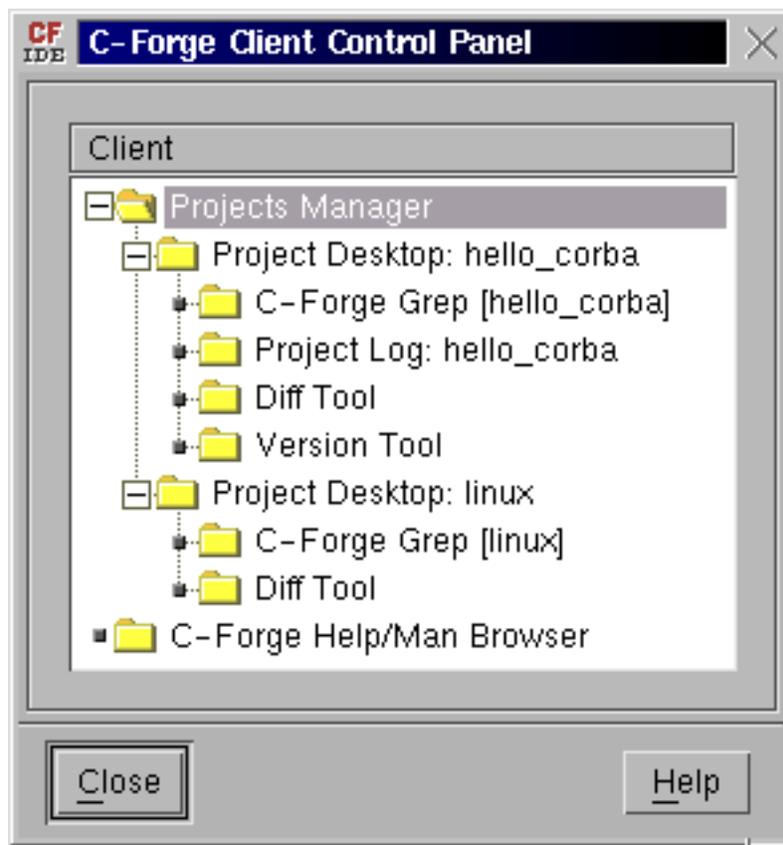
Window Menu

The **Window** menu is found on many of the top-level windows. It displays **Control panel** item and all the active top-level Code Forge windows.

If some dependent windows were opened from a top-level window, an arrow follows the name of the top-level window in the **Window** menu. Selecting such an item opens a submenu that displays all the dependent windows.

Selecting the name of a window brings the corresponding window to top.

Selecting **Control panel** item opens the Client Control Panel.



The Client Control Panel displays a list of all the current Code Forge clients and their dependencies. Double-clicking on the node will bring the corresponding window to top. Clicking the right mouse button on the node opens a pop-up menu with two items:

Open - brings the corresponding window to top.

Close - closes the corresponding window.

Regular Expression Filters

Windows that contain lists will also contain a regular expression Filter Text Field that is used to filter the list items.



The filter pattern is entered using UNIX [regular expressions](#). For example "*" will match all the files, "*.c" will match all the files with the extension ".c". You can also select an expression from the drop down portion of the combo-box. To filter the list, press the **Filter** button found in the dialog or press **Enter**. The list will be re-scanned to match the entered expression. To remove the filter, delete the regular expression and press **Enter**.

Trees

Trees display a hierarchical list of objects, each of which consists of a label and an optional bitmap. As a rule two different bitmaps are used for opened and closed tree nodes. The trees are typically used to display the files and directories on a disk, the targets and their dependencies, or any other kind of information that might usefully be displayed as a hierarchy.

The nodes of a tree can be opened or closed by clicking them with the left mouse button or using the keyboard:

- **Enter** - expand/collapse
- **Space** - expand/collapse
- **Ctrl+Up** - collapse the expanded node or collapse the parent of the collapsed node
- **Ctrl+Down** - expand.

Folders

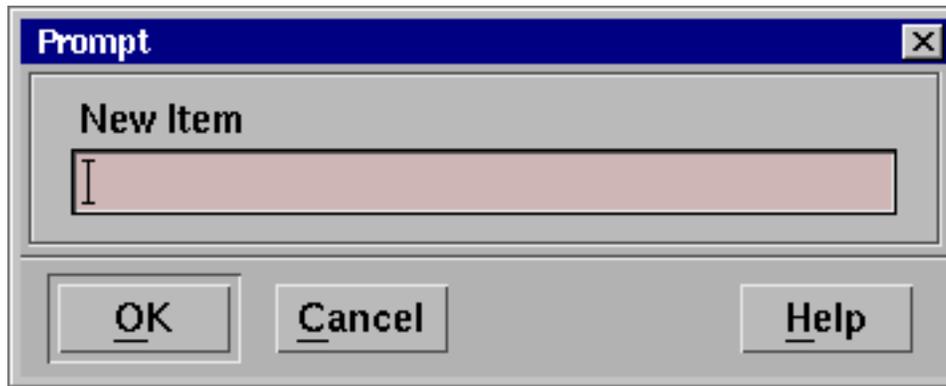
Folders panels display a set of named objects, such as buffers in an [Editor Window](#), actions to make/execute/debug in the [Node Properties](#) dialog, etc. A row of tabs at the top of the panel represents the list of folders in the "stack". Clicking on a tab with the mouse, brings the corresponding folder to the top.

Standard Accelerator Keys:

- **Ctrl+1..9** - jumps to the tab with given number
- **Ctrl+0** - shows the pop-up list menu, which can also be launched by clicking with the right mouse button on the tab
- **Ctrl+Alt+Tab (or Ctrl+ +)** - jumps to next tab
- **Ctrl+Alt+Shift+Tab(or Ctrl+ -)** - jumps to previous tab

Text Input Dialog

The Text Input Dialog appears when Code Forge is asking for input.



The dialog displays a prompt in a dialog box and waits for the user to input text and click the **OK** button. Clicking **Cancel** will cancel the input. Clicking **Help** will show corresponding Help window.

Title Bar

Code Forge displays the properties of some objects in a table format. The title bar of such tables can be used to change the appearance of the table and to sort the values.

Clicking the left mouse button, on a horizontal resizing column sash, and dragging it left or right to modify the width of a column in the table. To move the column, click on the column title, and drag it with the middle mouse button to a new location. Most of the values in the table can be sorted by clicking on the column headings. Clicking the column heading again will reverse the sort order. Any tree branches can be collapsed/expanded via Ctrl+Up/Ctrl+Down keys.

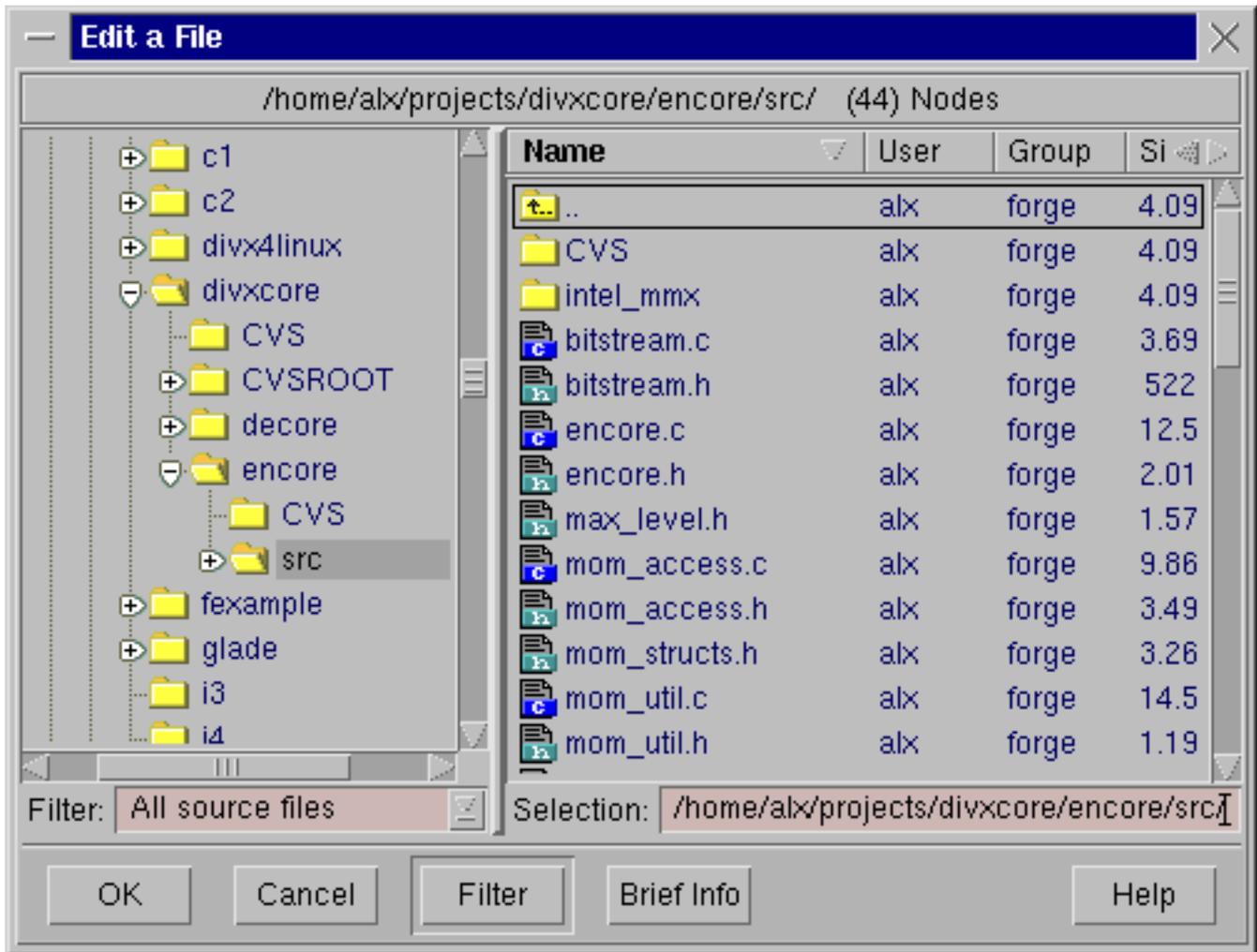
Last modified: Tuesday, 07-Aug-2001 09:03:10 EDT

File Selection Dialog

- [Overview](#)
- [Current Directory Name Label](#)
- [Directory List](#)
- [File List](#)
 - [Brief Info Mode](#)
 - [Full Info Mode](#)
 - [Full Info Mode Title Bar](#)
- [Filter Text Field](#)
- [Selection Text Field](#)
- [Action Area Buttons](#)

Overview

The File Selection Dialog displays a directory tree allowing the selection of one or more files.



The dialog is composed of six parts:

- The **Directory Name Label**,
- **Directory List**,
- **File List**,
- **Filter Text Field**,
- **Selection Text Field**, and
- **Action Area Buttons**.

Current Directory Name Label

The Current Directory Name Label displays the full path of the current directory and the number of nodes in the File List. It shows "Scanning..." while the current directory is being scanned.

Directory List

The Directory List shows a view of the directory tree. Selecting a directory node, by pressing **Enter** or

clicking on it with the left mouse button, displays the contents of the directory in the **File List**. The first time a directory node is opened Code Forge scans the contents of the directory. While the directory is being scanned the word "Scanning..." appears in the Directory Name Label.

File List

The File List displays all files that are found in the current directory node. Selecting a file node, by clicking the **OK** button, will close the dialog returning the selected node to the calling command. The File Selection dialog files can be viewed in two formats **Brief** or **Full** depending on the status of the **Brief/Full** Info button.

Brief Info Mode

Pressing the **Brief Info** button causes files to appear as columns of large icons that correspond to file types. If the file name is longer than 14 characters, the system will cut off the remaining ones and add a '~' symbol to the end.

Full Info Mode

When the **Full Info** mode is selected the File List is displayed as a table with a Title Bar.

The files and directories are shown as corresponding small icons along with the

- item name,
- user and group name,
- item size,
- date and time of the last modification,
- permission information and
- if this item is linked to some other element, the name of that element is also listed.

Full Info Mode Title Bar

The **Full Info Mode Title Bar** can be used to change the appearance of the table. The width of a column, in the table, can be changed by grabbing the horizontal resizing column sash using the left mouse button and dragging it left or right. To move a column, grab the column title, and drag it with the middle mouse button to a new location.

Filter Text Field

The Filter Text Field is used to filter the File List - the Directory List can not be filtered. The filter pattern is entered using UNIX [regular expressions](#). For example "*" will match all the files, "*.c" will match all the files with the extension ".c". To filter the File List, press the **Filter** button found in the Action Area or press the <Enter> key. The directory will be re-scanned to match the expression.

Selection Text Field

The Selection Text Field displays the name of the node selected in the File List along with its full path, or the name entered from the keyboard. Press the **OK** button, in the Action Area or the <Enter> key, to select a file and close the dialog.

Action Area Buttons

- **OK** - selects the item that is chosen in the Selection Text Field and closes the File Selection Dialog.
- **Cancel** - closes the File Selection Dialog without selecting anything.
- **Filter** - clears the File List and re-scans the current directory.
- **Help** - shows Help topic for the File Selection Dialog.

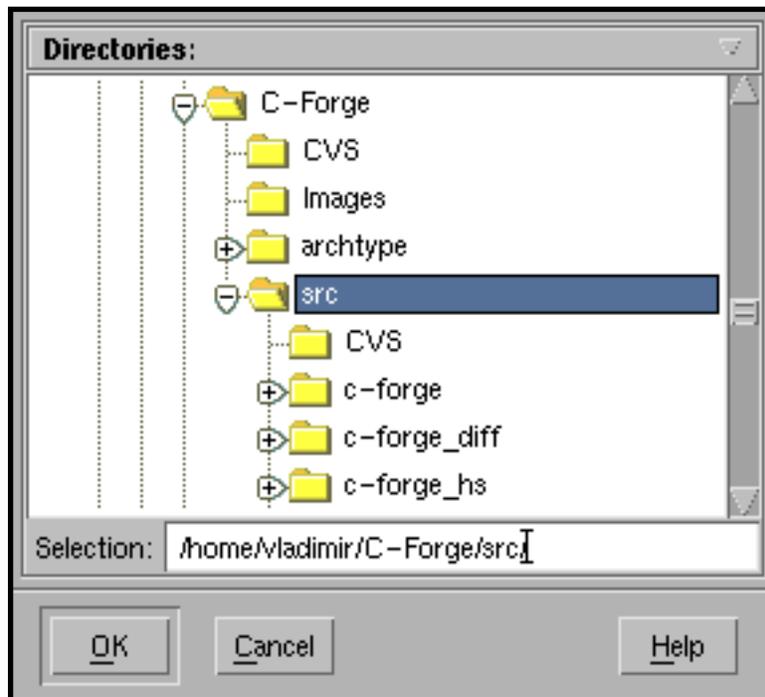
Last modified: Monday, 14-May-2001 07:03:19 EDT

Path Selection Dialog

- [Overview](#)
- [Directory List](#)
- [Selection Text Field](#)
- [Action Area Buttons](#)

Overview

The Path Selection Dialog displays a directory tree allowing the selection of directories.



The dialog is composed of three parts:

- **Directory List,**
- **Selection Text Field,** and
- **Action Area Buttons.**

Directory List

The Directory List shows a view of the directory tree. Selecting a directory node, by pressing **Enter** or clicking on it with the left mouse button, displays the contents of the directory just under the name of the

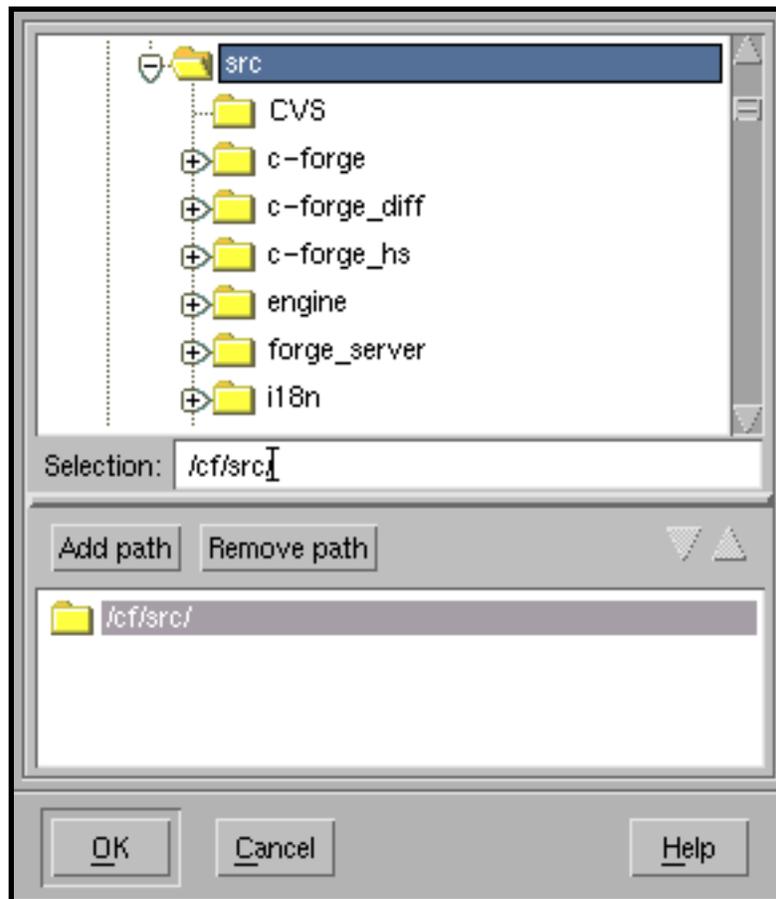
directory.

Selection Text Field

The Selection Text Field displays the name of the node selected in the Directory List along with its full path, or the name entered from the keyboard. Press the **OK** button, in the Action Area or the **<Enter>** key, to select a directory and close the dialog.

Edit Paths Dialog

In some cases you need to select multiple paths for operation (for example in the [Tools options](#) dialog for initializing "Include paths" or in the [Search/Replace](#) tool for custom path selection).



In this dialog you can friendly add new paths and remove already selected paths using "Add path" and "Remove path" buttons. You can also change the positions of the paths using arrows next to those buttons.

Action Area Buttons

- **OK** - selects the item that is chosen in the Selection Text Field and closes the Path Selection Dialog
 - **Cancel** - closes the Path Selection Dialog without selecting anything
 - **Help** - shows Help topic for the Path Selection Dialog
-

Last modified: Friday, 15-Mar-2002 11:58:38 EST

Mouse Reference

- [Mouse button 1](#)
- [Mouse button 2](#)
- [Mouse button 3](#)

Most of C-Forge windows allow using a mouse to work with objects.

Mouse Button 1 (Select)

Mouse button 1 (the left button, by default) is used to select objects.

To select a single object click its icon with Mouse button 1.

To select several successive objects in a list, select the first element in the list to be selected, hold down the **Shift** key and click on the last element in the list to be selected.

To select several objects on the desktop, place the mouse pointer in the background area, press and hold Mouse button 1, and then drag a rubber band box around multiple icons to select all the icons within the box.

To select several isolated objects, hold down the **Ctrl** key and click on another icon to add it to the current selection. A second **Ctrl** clicking on a selected object removes it from a group of selected objects.

Mouse button 1 can be also used to select text blocks.

Mouse Button 2 (Drag)

Mouse button 2 (the middle button, by default) can be used to drag and drop objects. For the mouse with two buttons the simultaneous use on both buttons simulates the middle button.

C-Forge supports 3 drag and drop modes:

- **Moving objects** - hold mouse button 2 down while dragging an icon. Release the button to drop.
- **Copy objects** - press and hold down the **Ctrl** key and drag using mouse button 2.
- **Create a link** - press and hold down the **Shift+Ctrl** keys and drag using mouse button 2.

By default, clicking by mouse button 2 transfers (copies) the selected text

Mouse Button 3 (Pop-up)

Mouse button 3 (the right button, by default) opens pop-up menu if it is supported in the current window.

Last modified: Monday, 14-May-2001 07:03:19 EDT

GNU Make Automatic Variables

GNU Make automatic variables are special feature of **make** utility. These variables have values computed each time a rule is executed, based on the target and dependencies of the rule.

Example: Suppose you are writing a pattern rule to compile a **.c** source file into a **.o** object file: how do you write the **cc** command so that it operates on the right source file name? You cannot write the name in the command, because the name is different each time the implicit rule is applied. So, you would use **\$\$@** for the object file name and **\$\$<** for the source file name.

Table of automatic variables

\$\$@ - the file name of the target of the rule. If the target is an archive member, then **\$\$@** is the name of the archive file. In a pattern rule that has multiple targets, **\$\$@** is the name of whichever target caused the rules commands to be executed.

\$\$% - the target member name, when the target is an archive member. For example, if the target is **foo.a(bar.o)** then **\$\$%** is **bar.o** and **\$\$@** is **foo.a**. **\$\$%** is empty when the target is not an archive member.

\$\$< - the name of the first dependency. If the target got its commands from an implicit rule, this will be the first dependency added by the implicit rule.

\$\$? - the names of all the dependencies that are newer than the target, with spaces between them. For dependencies which are archive members, only the member named is used.

\$\$^ - the names of all the dependencies, with spaces between them. For dependencies, which are archive members, only the member named is used. A target has only one dependency for each file it depends upon, no matter how many times each file is listed as a dependency. So if you list a dependency more than once for a target, the value of **\$\$^** contains just one copy of the name.

\$\$+ - this is like **\$\$^**, but dependencies listed more than once are duplicated in the order they were listed in the makefile. This is primarily useful for use in linking commands where it is meaningful to repeat library file names in a particular order.

\$\$* - the stem with which an implicit rule matches. If the target is **dir/a.foo.b** and the target pattern is **a.%.b** then the stem is **dir/foo**. The stem is useful for constructing names of related files.

In a static pattern rule, the stem is part of the file name that matches the **%** in the target pattern.

In an explicit rule, there is no stem; so `$$*` cannot be determined in that way. Instead, if the target name ends with a recognized suffix, `$$*` it is set to the target name minus the suffix. For example, if the target name is `foo.c`, then `$$*` is set to `foo`, since `.c` is a suffix. GNU `make` does this bizarre thing only for compatibility with other implementations of `make`. You should generally avoid using `$$*` except in implicit rules or static pattern rules.

If the target name in an explicit rule does not end with a recognized suffix, `$$*` is set to the empty string for that rule.

`$$?` is useful even in explicit rules when you wish to operate on only the dependencies that have changed. For example, suppose that an archive named `lib` is supposed to contain copies of several object files. This rule copies just the changed object files into the archive:

```
lib: foo.o bar.o lose.o win.o
ar r lib $$?
```

Of the variables listed above, four have values that are single file names, and two have values that are lists of file names. These six have variants that get just the files directory name or just the file name within the directory. The variant variables names are formed by appending `D` or `F`, respectively. These variants are semi-obsolete in GNU `make` since the functions `dir` and `notdir` can be used to get a similar effect. Note, however, that the `F` variants all omit the trailing slash which always appears in the output of the `dir` function.

Table of the variants

`$(@D)` - the directory part of the file name of the target, with the trailing slash removed. If the value of `$$@` is `dir/foo.o` then `$(@D)` is `dir`. This value is `.` if `$$@` does not contain a slash.

`$(@F)` - the file-within-directory part of the file name of the target. If the value of `$$@` is `dir/foo.o` then `$(@F)` is `foo.o`. `$(@F)` is equivalent to `$(notdir $$@)`.

`$(*D)`

`$(*F)` - the directory part and the file-within-directory part of the stem; `dir` and `foo` in this example.

`$(%D)`

`$(%F)` - the directory part and the file-within-directory part of the target archive member name. This makes sense only for archive member targets of the form `ARCHIVE(MEMBER)` and is useful only when `MEMBER` may contain a directory name.

`$(<D)`

`$(<F)` - the directory part and the file-within-directory part of the first dependency.

\$(^D)

\$(^F) - lists of the directory parts and the file-within-directory parts of all dependencies.

\$(?D)

\$(?F) - lists of the directory parts and the file-within-directory parts of all dependencies that are newer than the target.

Note that we use a special stylistic convention when we talk about these automatic variables; we write "the value of \$<", rather than "the variable <" as we would write for ordinary variables such as **objects** and **CFLAGS**. We think this convention looks more natural in this special case. Please do not assume it has any deep significance; \$< refers to the variable named < just as **\$(CFLAGS)** refers to the variable named **CFLAGS**. You could just as well use **\$(<)** in place of \$<.

Last modified: Monday, 14-May-2001 07:03:19 EDT

GNU Regular Expressions

GNU regular expressions allow you to specify patterns for filters and search strings.

The GNU regular expression facilities are like those of most Unix editors, but more powerful:

* - specifies a repetition of the preceding expression 0 or more times.

+ is like *, but specifies repetition of the preceding expression 1 or more times.

? is like *, but matches at most one repetition of the preceding expression.

| specifies an alternative. Two regular expressions A and B with | in between form an expression that matches anything that either A or B will match. Thus, *foo | bar* matches either *foo* or *bar*, but no other string.

| applies to the largest possible surrounding expressions. Only a surrounding (...) grouping can limit the grouping power of |.

Full backtracking capability exists when multiple |'s are used.

(...) are a grouping construct that serves three purposes:

- To enclose a set of | alternatives for other operations.

Thus, *(foo|bar)x* matches either *foox* or *barx*.

- To enclose a complicated expression for * to operate on.

Thus, *ba(na)** matches *bananana*, etc., with any number of *na*'s (zero or more).

- To mark a matched substring for future reference.

This is not a consequence of the idea of a parenthetical grouping; it is a separate feature that happens to be assigned as a second meaning of the same \(...\) construct because there is no conflict in practice between the two meanings. The following is an explanation of this feature.

\digit

After the end of a (...) construct, the matcher remembers the beginning and end of the text matched by that construct. Then, later on in the regular expression, you can use \ followed by a digit to mean, match the same text matched this time by the (...) construct. The first nine (...) constructs that appear in a regular expression are assigned numbers 1 through 9 in order of their beginnings. \1 through \9 can be used to refer to the text matched by the corresponding (...) construct.

For example, (.*)\1 matches any string that is composed of two identical halves. The (.*?) matches the first half, which can be anything, but the \1 that follows must match the exact text.

^

Matches the empty string, but only if it is at the beginning of the buffer.

\$

Matches the empty string, but only if it is at the end of the buffer.

\b

Matches the empty string, but only if it is at the beginning or end of a word. Thus,

\bfoo\b matches any occurrence of *foo* as a separate word.

\ball\s\|\)\b matches *ball* or *balls* as a separate word.

\B

Matches the empty string, provided it is not at the beginning or end of a word.

?

Matches the empty string, provided it is at the beginning of a word.

Matches the empty string, provided it is at the end of a word.

\w

Matches any word-constituent character. The editor syntax table determines which characters these are.

\W

Matches any character that is not a word-constituent.

`\s<code>`

Matches any character whose syntax is `<code>`. `<code>` is a letter that represents a syntax code: thus, `w` for word constituent, `-` for whitespace, `(` for open-parenthesis, etc.

Thus, `\s(` matches any character with open-parenthesis syntax.

`S<code>`

Matches any character whose syntax is not `<code>`.

Last modified: Tuesday, 19-Jun-2001 08:19:46 EDT

C-Forge command-line parameters

Command-line parameters for starting IDE

C-Forge IDE starts with the script 'cforge', you can use following options for this script when you starting the IDE:

- **-help**
shows short help message which describes main command-line options
- **-version**
shows the version number and build date of the C-Forge IDE
- **-display <dpy>**
using this option you can specify the X server to use, overriding the environment variable **DISPLAY**
- **-project <file>**
using this option you can specify project file (.mak) to load automatically after starting C-Forge.
- **-logo**
shows splash logo screen on startup
- **-no-logo**
do not shows splash logo screen on startup
- **-install**
installs a private color map (used for pseudo-color visuals)
- **-no-install**
always uses default color map
- **-fn **
sets default font list
- **-text_font **
sets font for the edit text area
- **-project_font **
sets font for the project dependency tree

- **-forge_server <port>**
sets UDP port number for forge server
- **-editor_server <port>**
sets TCP port number for editor server
- **-win_style**
to set presentation like Win95 (white background)
- **-std_style**
to set default presentation (gray background)

Note: You can also use default X Toolkit command line arguments.

Command-line parameters for starting Editor

C-Forge supports execution of the [Smart Editor](#) from the command line using the 'clsmed' command.

Usage: clsmed [options] <files>

where <files> is in the form: [+line_num] file [[+line_num] file]

- **line_num** - jump to line with specified number
- **file** - source file name (which may not exist)

Used options:

- **-o**
try to restore file order like in the previous SMED session
- **-v**
opens all files in the view only mode
- **-w**
opens files in separate windows
- **-N**
uses native C-Forge editor (default)
- **-L language**

sets language module for file, for example: 'C', 'C++', 'html'

- **-C**
uses CRiSP editor instead of default
- **-t tag_file**
uses specified tag file (generated by ctags or etags), you can also specify more than once -t option

Last modified: Tuesday, 19-Jun-2001 08:20:14 EDT