



## A Centaur tutorial

Ian Jacobs, Laurence Rideau-Gallot

► **To cite this version:**

Ian Jacobs, Laurence Rideau-Gallot. A Centaur tutorial. [Technical Report] RT-0140, INRIA. 1992, pp.102. inria-00070028

**HAL Id: inria-00070028**

**<https://hal.inria.fr/inria-00070028>**

Submitted on 19 May 2006

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



UNITÉ DE RECHERCHE  
INRIA-SOPHIA ANTIPOLIS

Institut National  
de Recherche  
en Informatique  
et en Automatique

Sophia Antipolis  
B.P. 109  
06561 Valbonne Cedex  
France  
Tél.: 93 65 77 77

Rapports Techniques

N°141

*Programme 2*  
*Calcul symbolique, Programmation*  
*et Génie logiciel*

**A Centaur Tutorial**

Ian JACOBS  
Laurence RIDEAU

Août 1992

# A Centaur Tutorial

Ian Jacobs  
INRIA Rocquencourt

Laurence Rideau-Gallot  
INRIA Sophia-Antipolis

August 18, 1992

---

# A Centaur Tutorial

## Abstract

This paper presents the `CENTAUR` system through a tutorial describing the creation of an environment for a small language of mathematical expressions called `EXP`. With `CENTAUR`, the user may interactively generate programming language environments, including structured editors, debuggers, interpreters, and other tools. In this tutorial, all phases of language specification are covered: the design of the abstract and concrete syntax of `EXP` in `METAL` and `SDF`, the pretty printing rules in `PPML`, and the semantics of an `EXP` interpreter in `TYPOL`. The tools generated by `CENTAUR` based on these specifications are enhanced by a user interface built with `CENTAUR` graphic primitives.

## Une Introduction au système Centaur

### Résumé

Dans ce papier, nous présentons le système `CENTAUR` qui est un outil interactif de génération d'environnements de langages de programmation, c'est à dire des éditeurs structurés, des interprètes, et d'autres outils de mise au point. Nous décrivons la création d'un environnement pour un petit langage d'expressions mathématiques appelé `EXP`. Toutes les étapes de spécification du langage sont parcourues: la spécification de la syntaxe abstraite et de la syntaxe concrète de `EXP` en `METAL` et en `SDF`, les règles d'affichage en `PPML`, et la sémantique d'un évaluateur d'expressions en `TYPOL`. De plus, les outils générés sont intégrés dans une interface homme-machine construite à partir des primitives graphiques de `CENTAUR`.

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	The Exp environment . . . . .	1
1.1.1	Syntax and parsers . . . . .	1
1.1.2	Pretty printers . . . . .	1
1.1.3	Evaluator . . . . .	1
1.1.4	Graphic interface . . . . .	1
1.2	Resources . . . . .	1
1.3	The Centaur environment . . . . .	2
1.3.1	The ccredit . . . . .	2
1.3.2	The ctview . . . . .	2
1.3.3	Error handling . . . . .	3
1.3.4	Parsers . . . . .	3
1.4	Preparation . . . . .	3
1.4.1	The Exp directory . . . . .	3
1.4.2	Initialization . . . . .	5
1.4.3	Calling Centaur . . . . .	5
1.5	Designing a language . . . . .	6
1.5.1	Abstract syntax . . . . .	6
1.5.2	Modular design . . . . .	7
1.5.3	Metal or Sdf . . . . .	8
<b>2</b>	<b>The Metal definition</b>	<b>9</b>
2.1	Preparation . . . . .	9
2.2	The abstract syntax . . . . .	9
2.2.1	Edition . . . . .	11
2.2.2	Compiling the abstract syntax . . . . .	12
2.3	The concrete syntax . . . . .	13
2.3.1	Compiling the concrete syntax . . . . .	17
2.4	The parser . . . . .	18
2.4.1	The Buildfile . . . . .	18
2.4.2	The token file . . . . .	18
2.4.3	Starting the parser . . . . .	19
2.4.4	Metal and yacc . . . . .	19
2.4.5	Reading an Exp program . . . . .	21

<b>3</b>	<b>The Sdf definition</b>	<b>22</b>
3.1	Preparation . . . . .	22
3.2	The structure of an Sdf program . . . . .	22
3.3	Sorts . . . . .	23
3.4	Context-free Syntax . . . . .	23
3.4.1	Production rules . . . . .	23
3.4.2	Associativity . . . . .	24
3.5	Priorities . . . . .	25
3.6	Lexical Syntax . . . . .	26
3.7	Variables . . . . .	27
3.8	The complete Sdf definition . . . . .	27
3.9	Compiling the Sdf specification . . . . .	28
<b>4</b>	<b>Ppml</b>	<b>29</b>
4.1	Pretty printers . . . . .	29
4.2	Patterns . . . . .	29
4.2.1	Print depth . . . . .	31
4.3	Boxes . . . . .	32
4.4	The pretty printer manager . . . . .	35
4.5	The Ppml file . . . . .	35
4.6	A preliminary pretty printer for EXP . . . . .	36
4.7	Compiling Ppml . . . . .	38
4.7.1	Preparation . . . . .	38
4.7.2	The <code>Compile</code> button . . . . .	39
4.7.3	The <code>Exp-std.at</code> file . . . . .	39
4.7.4	Pretty printer modules . . . . .	42
4.8	A second pretty printer for Exp . . . . .	43
4.9	Color and font specifications . . . . .	50
4.9.1	The resource manager . . . . .	50
4.9.2	Ppml resource classes . . . . .	52
<b>5</b>	<b>Typol</b>	<b>55</b>
5.1	Typol and Prolog . . . . .	55
5.2	Where to put your semantic definition . . . . .	55
5.3	The Typol file . . . . .	55
5.4	Natural semantics and Typol . . . . .	56
5.5	A simple evaluator for Exp . . . . .	59
5.5.1	Communicating with Lisp . . . . .	61
5.5.2	Importing Prolog predicates . . . . .	62

---

5.6	Compiling Typol . . . . .	65
5.7	The Typol object . . . . .	65
5.8	Executing a TYPOL program . . . . .	67
5.8.1	Debugging a TYPOL program . . . . .	67
5.9	A complete evaluator for Exp . . . . .	67
5.9.1	The ASSIGNMENTS language . . . . .	67
5.9.2	Evaluating in an environment . . . . .	71
5.9.3	Recovering the environment . . . . .	73
5.10	Semantic oriented syntax definitions . . . . .	79
<b>6</b>	<b>An Exp environment</b> . . . . .	<b>83</b>
6.1	The environment module . . . . .	83
6.2	Setting and clearing the environment . . . . .	84
6.3	The minimal environment . . . . .	84
6.3.1	A popup menu . . . . .	85
6.3.2	The evaluation function . . . . .	86
6.4	Associating windows . . . . .	87
6.5	Mouse actions . . . . .	88
6.6	Evaluation . . . . .	90
6.7	Killing windows . . . . .	92
6.8	Clearing the environment . . . . .	95
6.9	Adding a selection . . . . .	96
6.10	The selection algorithm . . . . .	98
6.10.1	Recovering a selection path . . . . .	98
6.10.2	Calculating a variable name . . . . .	98
6.10.3	Tree pattern matching . . . . .	99
6.10.4	Setting the selection path . . . . .	99
6.10.5	The mouse functions . . . . .	100
6.10.6	User input . . . . .	100
6.10.7	Select resources . . . . .	101

## 1 Introduction

In this tutorial, we walk through the development of an environment for a language of mathematical expressions called `EXP`. We highlight important notions along the way and explain all the steps necessary for creating tools for `EXP`.

### 1.1 The `Exp` environment

#### 1.1.1 Syntax and parsers

We specify the syntax of `EXP` with both the `SDF` and `METAL` formalisms, which we then compile to produce an abstract syntax (formalism) definition, a parser, and tree building functions which the parser uses to construct `VTP` abstract syntax trees from syntactically correct programs. All parsers generated from `METAL` specifications are external processes that communicate with `CENTAUR`.

#### 1.1.2 Pretty printers

We specify the pretty printer with `PPML`, which we compile to produce an incremental tree formatter. We may create a compiled `Le-Lisp` module for this pretty printer specification to improve performance.

#### 1.1.3 Evaluator

We specify an evaluator for `EXP` in `TYPOL`, which is compiled into a Prolog program. The evaluator reads `EXP` abstract trees, and according to the semantics we prescribe in `TYPOL`, evaluates it, returning an abstract syntax tree that represents the result.

#### 1.1.4 Graphic interface

We construct a graphic interface that allows us to trigger the evaluator with the mouse and display the results in a special window.

### 1.2 Resources

`CENTAUR` now features a *resource manager* by which users provide values for most system parameters, such as formalism locations, pretty printer colors and fonts, and `ctedit` mouse event lists. The resource manager consults the *current database*, which contains a list of resource specifications. Each specification assigns a value to an object parameter, called an object *property*.

A resource specification is a pair that assigns a value to a pattern. Each pattern represents the path leading to an object in the system hierarchy. The last word in the pattern is a property



of the object for which we may furnish a value, such as the background color used by a pretty printer or the location of a formalism. Patterns are modeled after those used by the X window system.<sup>1</sup>

We specify resources in *database files* that CENTAUR reads and stores in the current database. At the beginning of a session, CENTAUR automatically loads the file named `$HOME/.centaur.rdb` into the current database. This file contains a specification that “declares” new user languages, and possible other specifications that direct CENTAUR to separate database files. We organize database files hierarchically, mirroring the organization of system objects. Each database file contains resource information pertinent to a given object, and then directs us to database files that concern subobjects.

Except for the `.centaur.rdb` file, database files are loaded on demand only. Each time we modify a database file, however, we reread specifications into the current database by clicking on the button `Reset Resources`. This alone does not suffice to update our working environment, however, since tools in the environment only consult the database when they require parameter values. Thus, after resetting the database, we must also reset pertinent tools by hand so that they read fresh database values.

If at any time you wish to print the contents of the current database, type the line (`print-database`) in the CENTAUR main window. We discuss resource specifications and construct databases throughout the tutorial.

### 1.3 The Centaur environment

Aside from the specification languages, CENTAUR provides several generic tools that facilitate the development of an environment prototype.

#### 1.3.1 The *ctedit*

The *ctedit* is a structured editor which ensures syntactic correctness when you edit a program written in a language L. The *ctedit* allows only tree surgery that obeys a language’s abstract syntax definition.

#### 1.3.2 The *ctview*

The *ctview* is a viewing tool which allows us to manipulate programs through a graphic interface. Clicking on the `Editor` button of the CENTAUR main menu opens a new *ctview* which contains an empty *ctedit*. Each *ctview* menu bar has the following pulldown menus:

---

<sup>1</sup>Long resource specifications may be split on to several lines. Each line (except the last) should end with a backslash followed only by a newline. Due to a bug in X11R4 (corrected in X11R5), continued lines must not begin with white space.

- *File* : This menu allows you to read and parse programs from external files into the cedit, and write programs.
- *Display* : This menu allows you to modify certain pretty printing parameters (e.g., print level, page width), change pretty printers, and refresh the cedit contents.
- *Edit* : Allows you to perform generic edition: cut, copy, paste, or send selected subexpressions to a separate Gnu emacs editor for unstructured edition.
- *Selections* : Once you have read a program into a ctview, you may select program fragments by clicking and dragging with the mouse. This menu allows you to fine tune the presentation of selected material. You may make a selection visible, shrink it (set its local pretty print level to zero), expand it (the opposite), or unselect it.

When we read a program written in L into a ctview, it becomes specific to the language L. The cedit obeys L's syntax definition. If we have described a graphical environment for L, the ctview is transformed by the addition of the environment's buttons, menus, mouse behavior, etc.

### 1.3.3 Error handling

Language environments may use a generic mechanism for displaying error messages. For example, errors incurred when type checking a METAL, PPML, or TYPOL program appear in a special window. Clicking on an error message and then the `Show Error` button selects and highlights the error in the ctview containing the program.

Syntax errors encountered while parsing programs appear in the CENTAUR window. Syntax errors prevent the construction of an abstract syntax tree, so no tree appears in the editor.

### 1.3.4 Parsers

CENTAUR starts external parsers for any known language on demand, according to the system resources. These parsers continue to run even when you exit CENTAUR so that you may use them for another session.

## 1.4 Preparation

Before attacking the design of the EXP environment, we must make a few preparations.

### 1.4.1 The Exp directory

Although you may create a language directory anywhere you like, we suggest creating a directory: