OCaml at LexiFi

25 years of OCaml

Nicolás Ojeda Bär, nicolas.ojeda.bar@lexifi.com

FUN OCaml, September 15, 2025, Warsaw

LexiFi is a Paris-based software editor founded in 2000 specializing in the treatment of heterogeneous financial structured products.

Its software is fully written in OCaml. Its technology is based on an algebraic DSL allowing it to describe arbitrary contract payoffs in a generic way. This DSL allows applying a programming language-based approach to the treatment of such contracts.

Composing Contracts: An Adventure in Financial Engineering

Functional pearl

Simon Peyton Jones
Microsoft Research, Cambridge
simonpj@microsoft.com

Jean-Marc Eber
LexiFi Technologies, Paris
jeanmarc.eber@lexifi.com

Julian Seward
University of Glasgow
v-sewardj@microsoft.com

Abstract

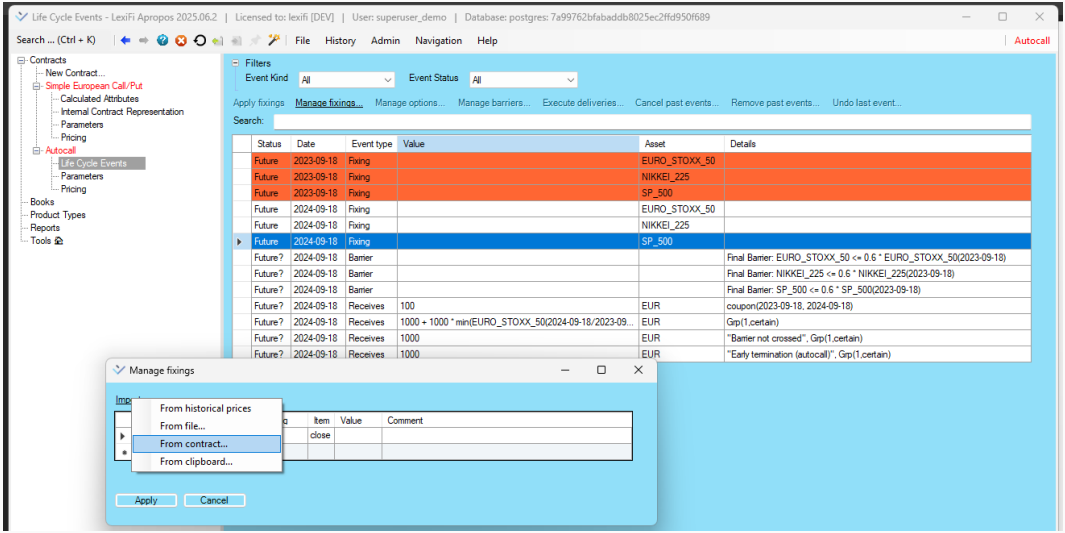
Financial and insurance contracts do not sound like promising territory for functional programming and formal semantics, but in fact we have discovered that insights from programming languages bear directly on the complex subject of describing and valuing a large class of contracts.

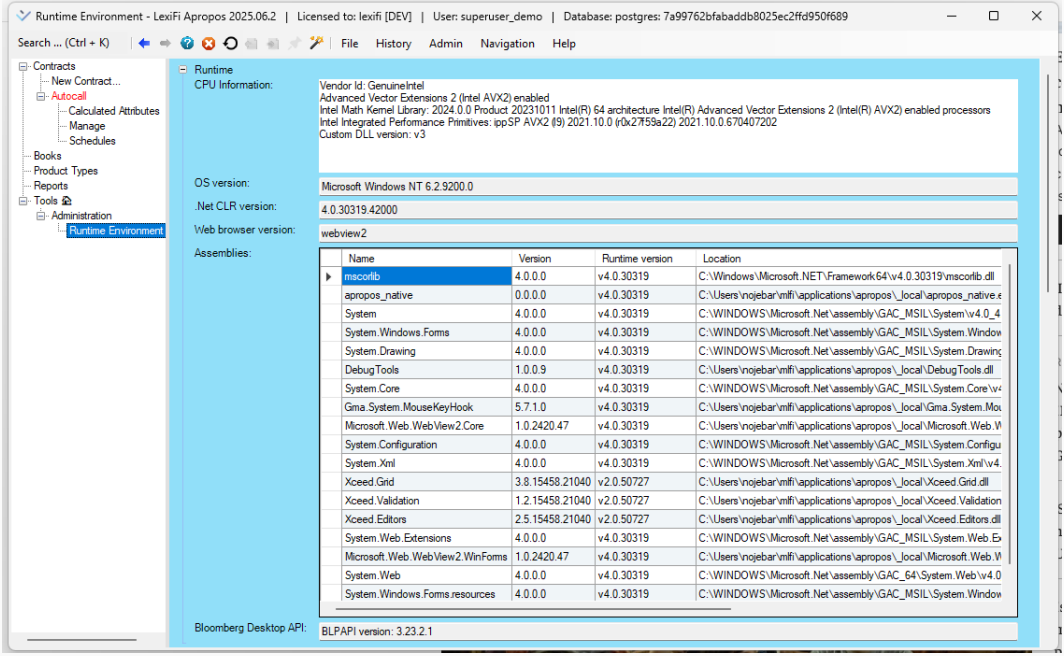
At this point, any red-blooded functional programmer should start to foam at the mouth, yelling "build a combinator library". And indeed, that turns out to be not only possible, but tremendously beneficial.

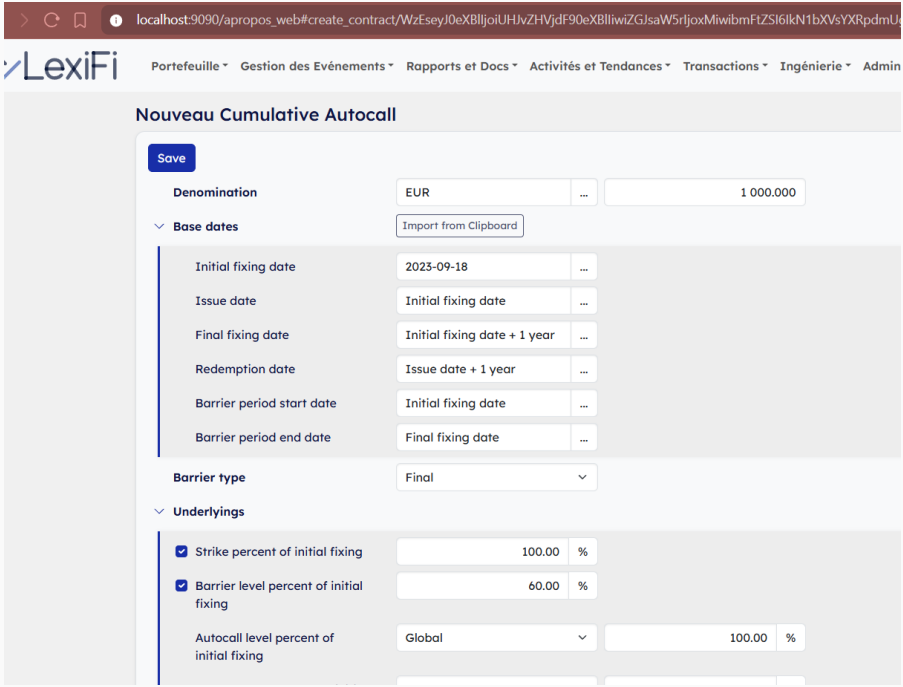
The finance industry has an enormous vocabulary of jargon for typical combinations of financial contracts (swaps, futures, caps, floors, swaptions, spreads, straddles, captions,

There is much left to do. We need to expand the set of contract combinators to describe a wider range of contracts; to expand the set of observables; to provide semantics for these new combinators; to write down and prove a range of theorems about contracts; to consider whether the notion of a "normal form" makes sense for contracts; to build a robust implementation; to exploit the dramatic simplifications that closed formulas make possible; to give a formal specification of the evolution of a contract during its life; and to validate all this in real financial settings. We have only just begun.

- LexiFi was likely first software editor to use OCaml in an industrial setting and to license (sell) software written in OCaml.
- While an unorthodox choice of language to develop a commercial desktop application at the time, it turned out to be extremely good one in hindsight. It has allowed a small team of developers to punch above its weight.
- LexiFi applies OCaml outside of its traditional areas of application (static analysis, compilation, high-assurance software, etc).
- Historical note: factors that tilted the scale towards OCaml at the time were use of a standard build system (Make), the straightforward C FFI which allowed easy interoperability, a readable, compact and open-source compiler ($\rightarrow$ easy to modify).


Life Cycle Events - LexiFi Apropos 2025.06.2 | Licensed to: lexifi [DEV] | User: superuser_demo | Database: postgres: 7a99762bfabaddb8025ec2ffd950f689
Search ... (Ctrl + K)
File
History
Admin
Navigation
Help
Autocall
Contracts
New Contract...
Simple European Call/Put
Calculated Attributes
Internal Contract Representation
Parameters
Pricing
Autocall
Life Cycle Events
Parameters
Pricing
Books
Product Types
Reports
Tools
Filters
Event Kind
All
Event Status
All
Apply fixings
Manage fixings...
Manage options...
Manage barriers...
Execute deliveries...
Cancel past events...
Remove past events...
Undo last event...
Search:
Status
Date
Event type
Value
Asset
Details
Future 2023-09-18 Fixing EURO_STOXX_50
Future 2023-09-18 Fixing NIKKEI_225
Future 2023-09-18 Fixing SP_500
Future 2024-09-18 Fixing EURO_STOXX_50
Future 2024-09-18 Fixing NIKKEI_225
Future 2024-09-18 Fixing SP_500
Future? 2024-09-18 Barrier Final Barrier: EURO_STOXX_50 <= 0.6 * EURO_STOXX_50(2023-09-18)
Future? 2024-09-18 Barrier Final Barrier: NIKKEI_225 <= 0.6 * NIKKEI_225(2023-09-18)
Future? 2024-09-18 Barrier Final Barrier: SP_500 <= 0.6 * SP_500(2023-09-18)
Future? 2024-09-18 Receives 100 EUR coupon(2023-09-18, 2024-09-18)
Future? 2024-09-18 Receives 1000 + 1000 * min(EURO_STOXX_50(2024-09-18/2023-09... EUR Grp(1,certain)
Future? 2024-09-18 Receives 1000 EUR "Barrier not crossed", Grp(1,certain)
Future? 2024-09-18 Receives 1000 EUR "Early termination (autocall)", Grp(1,certain)
Manage fixings
From historical prices
From file...
From contract...
From clipboard...
Item
Value
Comment
close
Apply
Cancel


Runtime Environment - LexiFi Apropos 2025.06.2 | Licensed to: lexifi [DEV] | User: superuser_demo | Database: postgres: 7a99762bfabaddb8025ec2ffd950f689
Search ... (Ctrl + K)
File
History
Admin
Navigation
Help
Contracts
New Contract...
Autocall
Calculated Attributes
Manage
Schedules
Books
Product Types
Reports
Tools
Administration
Runtime Environment
Runtime
CPU Information:
Vendor Id: GenuineIntel
Advanced Vector Extensions 2 (Intel AVX2) enabled
Intel Math Kernel Library: 2024.0.0 Product 20231011 Intel(R) 64 architecture Intel(R) Advanced Vector Extensions 2 (Intel(R) AVX2) enabled processors
Intel Integrated Performance Primitives: ippSP AVX2 (l9) 2021.10.0 (r0x2f59a22) 2021.10.0.670407202
Custom DLL version: v3
OS version:
Microsoft Windows NT 6.2.9200.0
.Net CLR version:
4.0.30319.42000
Web browser version:
webview2
Assemblies:
Name
Version
Runtime version
Location
mscorlib
4.0.0.0
v4.0.30319
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\mscorlib.dll
apropos_native
0.0.0.0
v4.0.30319
C:\Users\nojebar\mlfi\applications\apropos_local\apropos_native.e
System
4.0.0.0
v4.0.30319
C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System\v4.0_4
System.Windows.Forms
4.0.0.0
v4.0.30319
C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Windo
System.Drawing
4.0.0.0
v4.0.30319
C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Drawin
DebugTools
1.0.0.9
v4.0.30319
C:\Users\nojebar\mlfi\applications\apropos_local\DebugTools.dll
System.Core
4.0.0.0
v4.0.30319
C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Core\v4
Gma.System.MouseKeyHook
5.7.1.0
v4.0.30319
C:\Users\nojebar\mlfi\applications\apropos_local\Gma.System.Mou
Microsoft.Web.WebView2.Core
1.0.2420.47
v4.0.30319
C:\Users\nojebar\mlfi\applications\apropos_local\Microsoft.Web.W
System.Configuration
4.0.0.0
v4.0.30319
C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Configu
System.Xml
4.0.0.0
v4.0.30319
C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Xml\v4.
Xceed.Grid
3.8.15458.21040
v2.0.50727
C:\Users\nojebar\mlfi\applications\apropos_local\Xceed.Grid.dll
Xceed.Validation
1.2.15458.21040
v2.0.50727
C:\Users\nojebar\mlfi\applications\apropos_local\Xceed.Validation
Xceed.Editors
2.5.15458.21040
v2.0.50727
C:\Users\nojebar\mlfi\applications\apropos_local\Xceed.Editors.dll
System.Web.Extensions
4.0.0.0
v4.0.30319
C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Web.Ex
Microsoft.Web.WebView2.WinForms
1.0.2420.47
v4.0.30319
C:\Users\nojebar\mlfi\applications\apropos_local\Microsoft.Web.W
System.Web
4.0.0.0
v4.0.30319
C:\WINDOWS\Microsoft.Net\assembly\GAC_64\System.Web\v4.0
System.Windows.Forms.resources
4.0.0.0
v4.0.30319
C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Windo
Bloomberg Desktop API:
BLPAPI version: 3.23.2.1


localhost:9090/apropos_web#create_contract/WzEseyJ0eXBlIjoiUHJvZHVjdF90eXBlIiwiZGJsaW5rIjoxMiwibmFtZSI6IkN1bXVsYXRpdmU
LexiFi
Portefeuille
Gestion des Evénements
Rapports et Docs
Activités et Tendances
Transactions
Ingénierie
Admin
Nouveau Cumulative Autocall
Save
Denomination
EUR
1 000.000
Base dates
Import from Clipboard
Initial fixing date
2023-09-18
Issue date
Initial fixing date
Final fixing date
Initial fixing date + 1 year
Redemption date
Issue date + 1 year
Barrier period start date
Initial fixing date
Barrier period end date
Final fixing date
Barrier type
Final
Underlyings
Strike percent of initial fixing
100.00
%
Barrier level percent of initial fixing
60.00
%
Autocall level percent of initial fixing
Global
100.00
%

LexiFi

Portefeuille | Gestion des Evénements | Rapports et Docs | Activités et Tendances | Transactions | Ingénierie | Admin | Nouveau

Search contrac

TRS - 1 Year USD Phoenix Autocall Notes linked to Itau Unibanco Holding SA

Dashboard | Quant' Dashboard | Positions | Calendar | Payoff diagram | Simulation graphique | Documents | Trades | Quant' | Actions de Workflow

Simulation graphique

Settings | Scenario

Itau_Unibanco_Holding_SA | SOFR

Total value USD -7 190.88

Lifetime 0.98 years

A few words about myself.

I have been working at LexiFi for a little over ten years. Background in math, no professional programming experience before joining LexiFi. I learned OCaml as a hobby.

LexiFi turned out to be an excellent choice for someone interested in OCaml, owing to its long and unique history regarding OCaml. My own involvement with OCaml grew over time, and currently I help maintain the OCaml compiler, as well as the Dune build system and a few other projects. I am also a member of the executive committee of the OCaml Foundation.

Come talk to me if you want to chat about any of these subjects!

LexiFi and OCaml

LexiFi has played an active part in the the development of the OCaml compiler.

Before joining LexiFi, Alain Frisch, our CTO, spent a few years working in the Gallium team at INRIA (where OCaml was born).

LexiFi was a founding member of the Caml Consortium and the OCaml Foundation, and brought two of our technology clients to participate as well (SimCorp + Bloomberg). They also developed and grew their own OCaml development teams following our collaboration.

Some of the features where LexiFi has played a role (wholly or in part):

- Type-based disambiguation of constructors and record labels
- Inline records
- First-class modules
- Local exceptions
- Windows Unicode support
- Float unboxing optimizations
- RISC-V native-code backend
- Various warnings, warning mnemonics
- PPX (successor to Camlp4), and
- A lot of reviewing time, bug fixes, and standard library contributions

LexiFi is also involved in the community around OCaml. A number of tools that were developed internally have been open-sourced.

- native `dynlink` & `flexdll` (OCaml Windows linker)
- `sedlex` (Unicode lexer generator)
- `csml` (.NET/OCaml bridge)
- `ppx_tools` (prehistoric predecessor to `ppxlib`)
- `gen_js_api` (binding generator for the `js_of_ocaml` compiler)
- `landmarks` (time profiler)
- `webgl-plot` (WebGL-powered plotting engine)
- `dead_code_analyzer` (global dead code detection)
- `ocaml-vdom` (Elm architecture in OCaml)

The `gen_js_api` binding generator is used to build mainstream projects such as OCaml-LSP.

Development Environment

```
github.com/AlDanial/cloc v 2.06  T=13.38 s (354.7 files/s, 273904.6 lines/s)
```

Language	files	blank	comment	code
Text	472	4966	0	2011899
OCaml	2275	101389	74154	872586
HTML	452	28581	2127	199709
Markdown	321	8895	7	98630
CSV	38	22	0	49042
C#	91	4821	1930	35509
SVG	285	6	6	35134
C	64	2776	1876	25644
JSON	131	1	0	24430
SCSS	158	2025	1491	12656
XML	51	285	135	9372
CSS	31	1256	258	6083
Bourne Shell	141	1303	298	5856
Python	56	1502	503	5411
make	40	1111	379	4747
Java	16	371	142	3523
TeX	3	438	239	3239
PHP	3	107	0	2880
C/C++ Header	24	623	1825	2663
TypeScript	9	222	146	1978
JavaScript	14	430	396	1902
Groovy	22	24	135	829

- Mid-size monorepo: 900k LOC, 1500 modules
- Currently running OCaml 5 in production. Migration from OCaml 4 straightforward, but some issues around weak pointers and increased memory usage (issues being actively worked on).
- Target Windows, Linux and the Web. Developers work on Windows/WSL almost exclusively.
- Editors: Emacs/Vim/VS Code, in combination with OCaml-LSP/Merlin.

- Users of `ocp-indent`: trivial to integrate, works flawlessly, lets us maintain our existing formatting style. Tried OCamlFormat, but a few obstacles migrating an old codebase: opinioned styling, complications around merging changes between "old" and "new" branches, etc. But would like to switch to it eventually.
- Enforce policy of not wasting time discussing whitespace during review. Whitespace diffs are strongly discouraged ($\rightarrow$ harms ability to track changes over time, which is much more important than an uniform formatting style in a 25-year-old codebase).

- CI: tried Travis, AppVeyor, GH Actions. Completely stateless model of cloud CI providers inefficient and expensive in our experience. Currently: a few beefy dedicated VMs managed with Jenkins. Not fully stateless. We brought our budget down considerably, and a full CI run on Windows went from ~40m to ~10m (~8m on Linux).
- Keep binary cache of all our dependencies (compiler + third-party OCaml libraries), built in the CI and stored in S3 → quickly bootstrap a working environment in developer machines without having to compile everything from scratch. Very useful when switching branches (→ small compiler patch for relocatibility).

- Build system (in chronological order): Make, OMake, Dune.
- Were happy with OMake: easy to extend, etc, but moved on when Dune arrived because OMake was no longer actively developed. LexiFi funded Gerd Stolpmann to maintain OMake for some time.
- Dune provides a good out-of-the-box experience for a mostly-OCaml (+ C bindings) codebase (probably not the best solution if your codebase is more heterogeneous)
- Good things about Dune: correctness, composability ($\rightarrow$ trivial integration of libraries into monorepo), excellent Windows support.
- A few long-standing gripes about Dune: dependency overapproximation across libraries means a lot of unnecessary recompilation. Sluggish performance on Windows compared to Linux.
- Did some experiments with Ninja, but integration with Merlin remains an issue.

What about libraries?

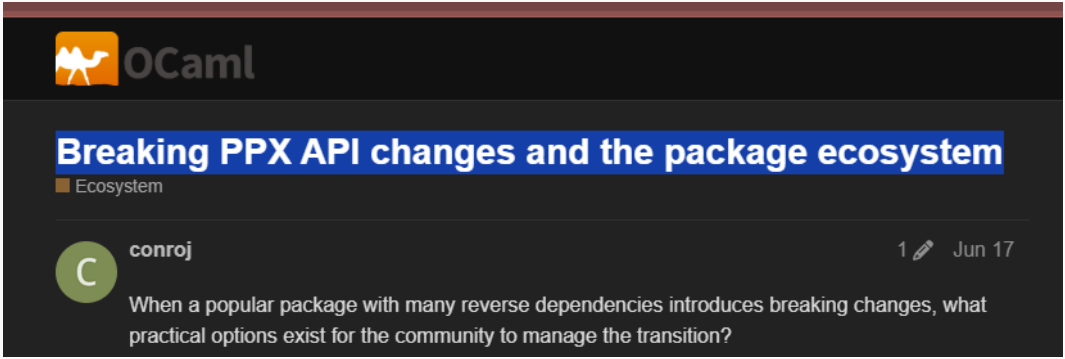
- Integrating a library means (potentially a lot of) code where bugs or security issues can lurk. This means that a serious auditing process is required before integrating third-party code into our codebase. Also licensensing issues.
- Strong preference for small, well tested and focused libraries that solve a precise problem, rather than "frameworks" that try to solve many problems and that require the user into a specific programming style or choices.
- Full list of OCaml libraries that we currently link: `camlpdf`, `ocaml-sha`, `ocurl`, `postgresql`, `sqlite`, `xmlm`, `camlzip`, `cmarkit`, `ocaml-dtoa`, `uutf`, `uucp`, `qrc`, `cryptokit`.
- A few large non-OCaml libraries: Intel MKL, LibXL, WebView2 (= Chromium), specialized UI widgets, etc.

We do not use OPAM. Historically, OPAM came into the picture in 2011 and first-class Windows support was introduced only recently. But more seriously, a package manager like OPAM addresses the needs of a industrial environment only tangentially:

- Having access to lots of libraries $\rightarrow$ dubious usefulness: we only integrate libraries once in a while, small libraries that we can easily integrate directly in our monorepo (with licenses which are compatible with our business model).
- Dune solves the problem of easily getting access to libraries for us: just drop the library repo in the tree and use it.
- A powerful constraint solver is less useful when using libraries with few to no dependencies.
- Automatically updating libraries $\rightarrow$ bad
- Lack of reproducibility when doing `opam update` $\rightarrow$ bad
- Dedicated repositories, etc... are possible, but then you end up using a small percentage of OPAM, and we already had ad-hoc infrastructure in place.

Very light use of PPX.

- `landmarks`, our time profiler, uses PPX for instrumentation, and
- `gen_js_api` has a PPX mode to generate JS bindings "inline".
- PPX is fragile and sensitive to upstream changes, degrades compilation time
- Compiling a file "by hand" harder if using PPX
- One really useful PPX is `ppx_deriving`, but we have Runtime Types (later)



What about parallelism?

- Processor-based parallelism, communicating using message-passing over sockets, using `Marshal` for serialization.
- Works well, but inefficient memory use (not easy to share heap data).
- We do not use Lwt/Async/etc or any other concurrency protocol. Hard to integrate to an existing codebase (need to insert explicit yield points, monadic code "infects" everything). Direct-style concurrency as in OCaml 5 helps here, but explicit yield points still need to be inserted.
- Investigating switching to OCaml 5 parallelism but task is gargantuan due to the vast amount of global state. Also concerns about performance when increasing the number of domains.

Language Use

- Heavy users of "advanced" features of the language. Objects to implement mixins, .NET bindings, etc. GADTs for type-safe manipulation of Runtime Types. First-class modules to implement "dynamic factories" (implemented first at LexiFi and later upstreamed; if existentials had existed, maybe would have proceeded differently), type-based disambiguation, labelled/optional arguments, inline records, etc.
- When in doubt, avoid fancy types (polymorphic variants, first-class modules, objects, GADTs, etc.). Most bugs are logical and cannot be caught by the type system anyway. Fancy types make the code harder to understand and modify.
- Option and result types at library boundaries, exceptions for control flow and error handling within implementations.
- Immutability whenever possible, but without fetishizing it.

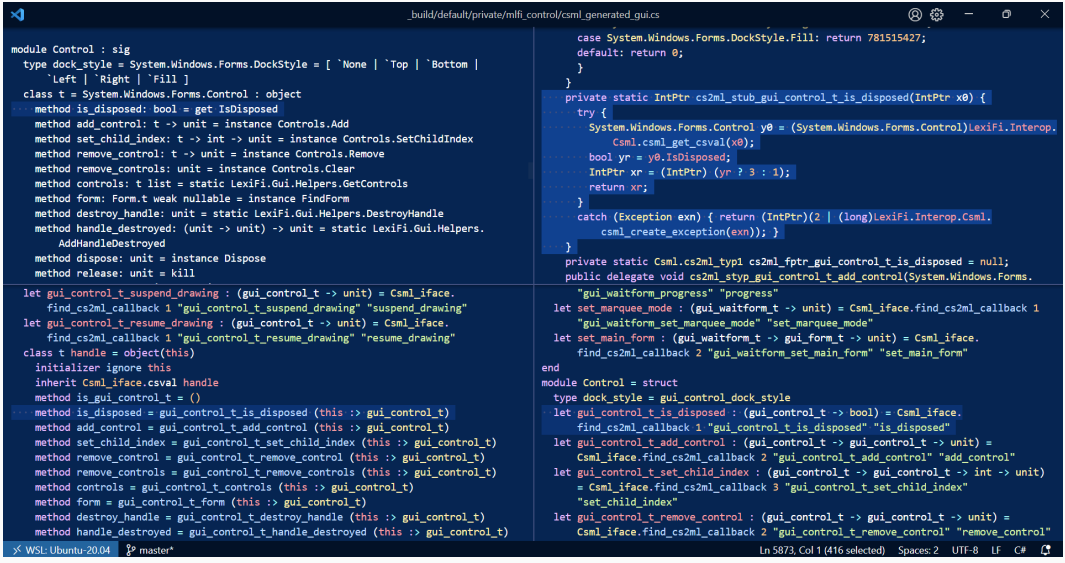
- Code readability is paramount. Anything that harms readability should be looked at critically. Infix operators, monads and monadic operators, and other such features are generally discouraged.
- Debugging: most bugs in ordinary OCaml code are logical errors. The quote of Brian W. Kernighan and Rob Pike is relevant here: "As personal choice, we tend not to use debuggers [...] we find stepping through a program less productive than thinking harder and adding output statements and self-checking code at critical places."
- Having a way to print arbitrary values is very useful, and lacking in vanilla OCaml. We solve it using Runtime Types (later).

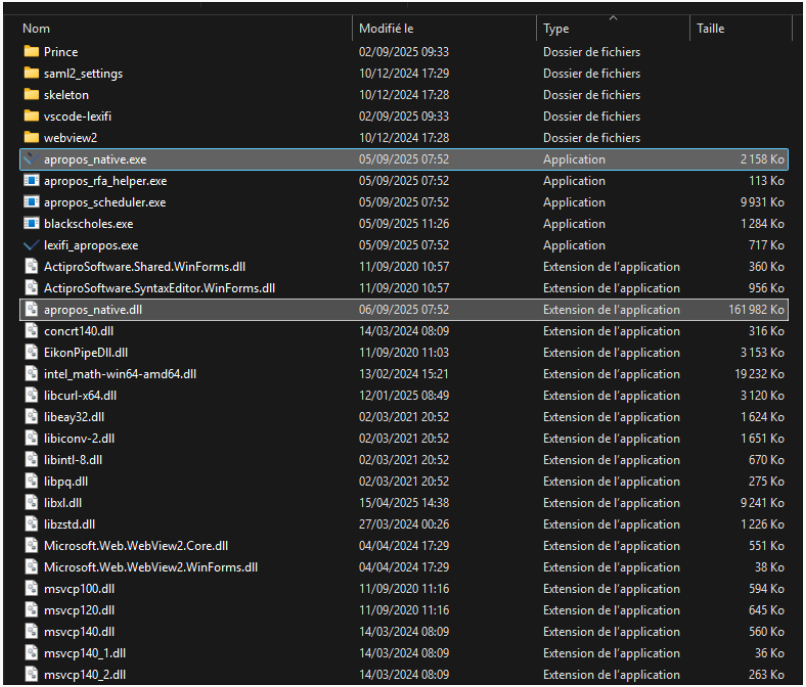
- Floats are boxed, but compiler can elide boxing of intermediate results in FORTRAN-style code (i.e. loops instead of recursive functions). Numerical code is often clearer when written this way.
- Numerical code written in OCaml + a large but uniform set of primitives written in C. Vectorized primitives used from the Intel MKL.
- `no-flat-float-array` mode: `float array` is boxed, use `floatarray` instead.
- We do not use F_λ. Performance-critical code small subset of all code. Manual optimization using the standard backend provides sufficient leverage. Automatically optimizing the rest of the code not worth it currently ($\rightarrow$ bigger object size, increased compilation time, gap between source and generated code).
- Manually written C bindings. In practice, not hard to do. Very few bugs/issues so far with this approach. Have not felt the need for eg `ctypes` so far.

Deployment

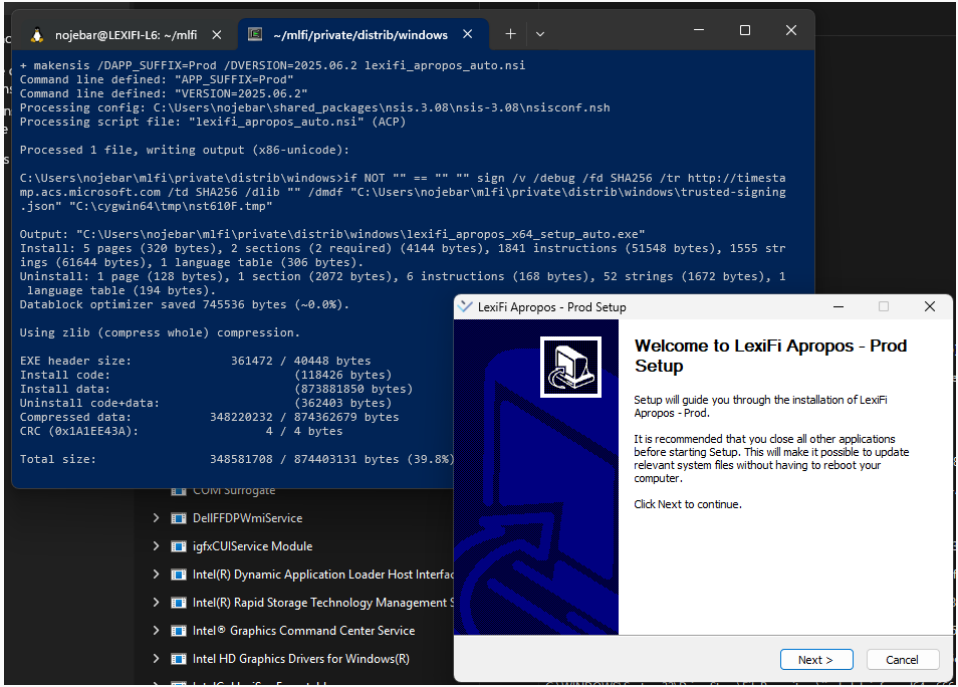
Deploying on Windows.

- Two production compilers on Windows: Microsoft's C compiler (MSVC) and GCC (via the `mingw-w64` toolchain). Cross-platform support in the standard and the `unix` library is excellent.
- Compile applicative code + runtime to a shared library: `-output-complete-obj`.
- A small .NET driver starts up the necessary services and ends by jumping into the OCaml runtime.
- Deliver via NSIS installer (open-source scriptable Windows installer creator). The installation script is less than 100 LOC, touched less than once a year.
- Application signing on Windows via cloud API very accessible ($<$ 10 EUR/month).
- Use the CSML to bridge .NET and OCaml and to generate bindings to call .NET libraries. Winform bindings on top of that to build native GUI. CSML currently dormant, as all new development is happening on the Web.


_build/default/private/mlfi_control/csml_generated_gui.cs
module Control : sig
type dock_style = System.Windows.Forms.DockStyle = [`None | `Top | `Bottom |
`Left | `Right | `Fill]
class t = System.Windows.Forms.Control : object
method is_disposed: bool = get IsDisposed
method add_control: t -> unit = instance Controls.Add
method set_child_index: t -> int -> unit = instance Controls.SetChildIndex
method remove_control: t -> unit = instance Controls.Remove
method remove_controls: unit = instance Controls.Clear
method controls: t list = static LexiFi.Gui.Helpers.GetControls
method form: Form.t weak nullable = instance FindForm
method destroy_handle: unit = static LexiFi.Gui.Helpers.DestroyHandle
method handle_destroyed: (unit -> unit) -> unit = static LexiFi.Gui.Helpers.
AddHandleDestroyed
method dispose: unit = instance Dispose
method release: unit = kill
case System.Windows.Forms.DockStyle.Fill: return 781515427;
default: return 0;
}
}
private static IntPtr cs2ml_stub_gui_control_t_is_disposed(IntPtr x0) {
try {
System.Windows.Forms.Control y0 = (System.Windows.Forms.Control)LexiFi.Interop.
Csml.csml_get_csval(x0);
bool yr = y0.IsDisposed;
IntPtr xr = (IntPtr) (yr ? 3 : 1);
return xr;
}
catch (Exception exn) { return (IntPtr)(2 | (long)LexiFi.Interop.Csml.
csml_create_exception(exn)); }
}
private static Csml.cs2ml_typ1 cs2ml_fptr_gui_control_t_is_disposed = null;
public delegate void cs2ml_styp_gui_control_t_add_control(System.Windows.Forms.
let gui_control_t_suspend_drawing : (gui_control_t -> unit) = Csml_iface.
find_cs2ml_callback 1 "gui_control_t_suspend_drawing" "suspend_drawing"
let gui_control_t_resume_drawing : (gui_control_t -> unit) = Csml_iface.
find_cs2ml_callback 1 "gui_control_t_resume_drawing" "resume_drawing"
class t handle = object(this)
initializer ignore this
inherit Csml_iface.csval handle
method is_gui_control_t = ()
method is_disposed = gui_control_t_is_disposed (this :> gui_control_t)
method add_control = gui_control_t_add_control (this :> gui_control_t)
method set_child_index = gui_control_t_set_child_index (this :> gui_control_t)
method remove_control = gui_control_t_remove_control (this :> gui_control_t)
method remove_controls = gui_control_t_remove_controls (this :> gui_control_t)
method controls = gui_control_t_controls (this :> gui_control_t)
method form = gui_control_t_form (this :> gui_control_t)
method destroy_handle = gui_control_t_destroy_handle (this :> gui_control_t)
method handle_destroyed = gui_control_t_handle_destroyed (this :> gui_control_t)
"gui_waitform_progress" "progress"
let set_marquee_mode : (gui_waitform_t -> unit) = Csml_iface.find_cs2ml_callback 1
"gui_waitform_set_marquee_mode" "set_marquee_mode"
let set_main_form : (gui_waitform_t -> gui_form_t -> unit) = Csml_iface.
find_cs2ml_callback 2 "gui_waitform_set_main_form" "set_main_form"
end
module Control = struct
type dock_style = gui_control_dock_style
let gui_control_t_is_disposed : (gui_control_t -> bool) = Csml_iface.
find_cs2ml_callback 1 "gui_control_t_is_disposed" "is_disposed"
let gui_control_t_add_control : (gui_control_t -> gui_control_t -> unit) =
Csml_iface.find_cs2ml_callback 2 "gui_control_t_add_control" "add_control"
let gui_control_t_set_child_index : (gui_control_t -> gui_control_t -> int -> unit)
= Csml_iface.find_cs2ml_callback 3 "gui_control_t_set_child_index"
"set_child_index"
let gui_control_t_remove_control : (gui_control_t -> gui_control_t -> unit) =
Csml_iface.find_cs2ml_callback 2 "gui_control_t_remove_control" "remove_control"
WSL: Ubuntu-20.04
master*
Ln 5873, Col 1 (416 selected)
Spaces: 2
UTF-8
LF
C#



Nom	Modifié le	Type	Taille
Prince	02/09/2025 09:33	Dossier de fichiers	
saml2_settings	10/12/2024 17:29	Dossier de fichiers	
skeleton	10/12/2024 17:28	Dossier de fichiers	
vscode-lexifi	02/09/2025 09:33	Dossier de fichiers	
webview2	10/12/2024 17:28	Dossier de fichiers	
apropos_native.exe	05/09/2025 07:52	Application	2 158 Ko
apropos_rfa_helper.exe	05/09/2025 07:52	Application	113 Ko
apropos_scheduler.exe	05/09/2025 07:52	Application	9 931 Ko
blackscholes.exe	05/09/2025 11:26	Application	1 284 Ko
lexifi_apropos.exe	05/09/2025 07:52	Application	717 Ko
ActiproSoftware.Shared.WinForms.dll	11/09/2020 10:57	Extension de l'application	360 Ko
ActiproSoftware.SyntaxEditor.WinForms.dll	11/09/2020 10:57	Extension de l'application	956 Ko
apropos_native.dll	06/09/2025 07:52	Extension de l'application	161 982 Ko
concrt140.dll	14/03/2024 08:09	Extension de l'application	316 Ko
EikonPipeDll.dll	11/09/2020 11:03	Extension de l'application	3 153 Ko
intel_math-win64-amd64.dll	13/02/2024 15:21	Extension de l'application	19 232 Ko
libcurl-x64.dll	12/01/2025 08:49	Extension de l'application	3 120 Ko
libeay32.dll	02/03/2021 20:52	Extension de l'application	1 624 Ko
libiconv-2.dll	02/03/2021 20:52	Extension de l'application	1 651 Ko
libintl-8.dll	02/03/2021 20:52	Extension de l'application	670 Ko
libpq.dll	02/03/2021 20:52	Extension de l'application	275 Ko
libxl.dll	15/04/2025 14:38	Extension de l'application	9 241 Ko
libzstd.dll	27/03/2024 00:26	Extension de l'application	1 226 Ko
Microsoft.Web.WebView2.Core.dll	04/04/2024 17:29	Extension de l'application	551 Ko
Microsoft.Web.WebView2.WinForms.dll	04/04/2024 17:29	Extension de l'application	38 Ko
msvcp100.dll	11/09/2020 11:16	Extension de l'application	594 Ko
msvcp120.dll	11/09/2020 11:16	Extension de l'application	645 Ko
msvcp140.dll	14/03/2024 08:09	Extension de l'application	560 Ko
msvcp140_1.dll	14/03/2024 08:09	Extension de l'application	36 Ko
msvcp140_2.dll	14/03/2024 08:09	Extension de l'application	263 Ko


nojebar@LEXIFI-L6: ~/mlfi
~/mlfi/private/distrib/windows
+ makensis /DAPP_SUFFIX=Prod /DVERSION=2025.06.2 lexifi_apropos_auto.nsi
Command line defined: "APP_SUFFIX=Prod"
Command line defined: "VERSION=2025.06.2"
Processing config: C:\Users\nojebar\shared_packages\nsis.3.08\nsis-3.08\nsisconf.nsh
Processing script file: "lexifi_apropos_auto.nsi" (ACP)
Processed 1 file, writing output (x86-unicode):
C:\Users\nojebar\mlfi\private\distrib\windows>if NOT "" == "" "" sign /v /debug /fd SHA256 /tr http://timesta
mp.acs.microsoft.com /td SHA256 /dlib "" /dmdf "C:\Users\nojebar\mlfi\private\distrib\windows\trusted-signing
.json" "C:\cygwin64\tmp\nst610F.tmp"
Output: "C:\Users\nojebar\mlfi\private\distrib\windows\lexifi_apropos_x64_setup_auto.exe"
Install: 5 pages (320 bytes), 2 sections (2 required) (4144 bytes), 1841 instructions (51548 bytes), 1555 str
ings (61644 bytes), 1 language table (306 bytes).
Uninstall: 1 page (128 bytes), 1 section (2072 bytes), 6 instructions (168 bytes), 52 strings (1672 bytes), 1
language table (194 bytes).
Datablock optimizer saved 745536 bytes (~0.0%).
Using zlib (compress whole) compression.
EXE header size: 361472 / 40448 bytes
Install code: (118426 bytes)
Install data: (873881850 bytes)
Uninstall code+data: (362403 bytes)
Compressed data: 348220232 / 874362679 bytes
CRC (0x1A1EE43A): 4 / 4 bytes
Total size: 348581708 / 874403131 bytes (39.8%)
COM Surrogate
DellFFDPWmiService
igfxCUIService Module
Intel(R) Dynamic Application Loader Host Interfac
Intel(R) Rapid Storage Technology Management S
Intel® Graphics Command Center Service
Intel HD Graphics Drivers for Windows(R)
LexiFi Apropos - Prod Setup
Welcome to LexiFi Apropos - Prod Setup
Setup will guide you through the installation of LexiFi Apropos - Prod.
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.
Click Next to continue.
Next >
Cancel

Deploying on Linux, the Cloud and the Web.

- For institutional clients, standard deployment involves a Web frontend, a server-side running under Docker on AWS EC2 or standalone virtual machine. Build Docker image in CI, upload to S3, download from deployment host. On-premise hosting by the clients is also supported.
- For some clients ("platforms") the web server exposes only HTTP APIs (no frontend) that they can use to build their own products.
- Interfacing with AWS for deployment tasks is done mostly using Python scripts (+ static types!), which is quite good at this task. Initially using OCaml, but covering all needed APIs was a lot of work. We did not find a "turnkey" alternative by the community.
- On the Web: we use the excellent `js_of_ocaml` compiler, combined with our own (open-source) `gen_js_api` to generate API bindings and `ocaml-vdom` to write client-side applications.


_build/default/private/thirdparty/ocaml-vdom/lib/js_browser.ml
end
module Document: sig
type t
val t_of_js: Ojs.t -> t
val t_to_js: t -> Ojs.t
val create_element: t -> string -> Element.t [@@js.call]
val create_element_ns: t -> string -> string -> Element.t [@@js.call "createElementNS"]
val create_text_node: t -> string -> Element.t [@@js.call]
val create_event: t -> string -> Event.t [@@js.call]
val create_range: t -> Range.t [@@js.call]
val get_element_by_id: t -> string -> Element.t option [@@js.call]
val get_elements_by_class_name: t -> string -> Element.t array [@@js.call]
val get_elements_by_tag_name: t -> string -> Element.t array [@@js.call]
val body: t -> Element.t [@@js.get]
val document_element: t -> Element.t [@@js.get]
val active_element: t -> Element.t [@@js.get]
val cookie: t -> string [@@js.get]
val set_cookie: t -> string -> unit [@@js.set]
val set_title: t -> string -> unit [@@js.set]
val open_: t -> ?mime_type:string -> ?history_mode:string -> unit -> unit [@@js.call "open"]
val write: t -> string -> unit [@@js.call]
val writeln: t -> string -> unit [@@js.call]
val close: t -> unit [@@js.call]
ignore (Ojs.call (t_to_js x474) "removeAllRanges" [||])
end
module Document =
struct
type t = Ojs.t
let rec t_of_js : Ojs.t -> t = fun (x476 : Ojs.t) -> x476
and t_to_js : t -> Ojs.t = fun (x475 : Ojs.t) -> x475
let (create_element : t -> string -> Element.t) =
fun (x478 : t) ->
fun (x477 : string) ->
Element.t_of_js
(Ojs.call (t_to_js x478) "createElement"
[|(Ojs.string_to_js x477)|])
let (create_element_ns : t -> string -> string -> Element.t) =
fun (x481 : t) ->
fun (x479 : string) ->
fun (x480 : string) ->
Element.t_of_js
(Ojs.call (t_to_js x481) "createElementNS"
[|(Ojs.string_to_js x479);(Ojs.string_to_js x480)|])
let (create_text_node : t -> string -> Element.t) =
fun (x483 : t) ->
fun (x482 : string) ->
Element.t_of_js
(Ojs.call (t_to_js x483) "createTextNode"
[|(Ojs.string_to_js x482)|])
let (create_event : t -> string -> Event.t) =
fun (x485 : t) ->
fun (x484 : string) ->
Event.t_of_js
(Ojs.call (t_to_js x485) "createEvent"
[|(Ojs.string_to_js x484)|])
WSL: Ubuntu-20.04
master*
Ln 1975, Col 1 (56 selected)
Spaces: 2
UTF-8
LF
OCaml

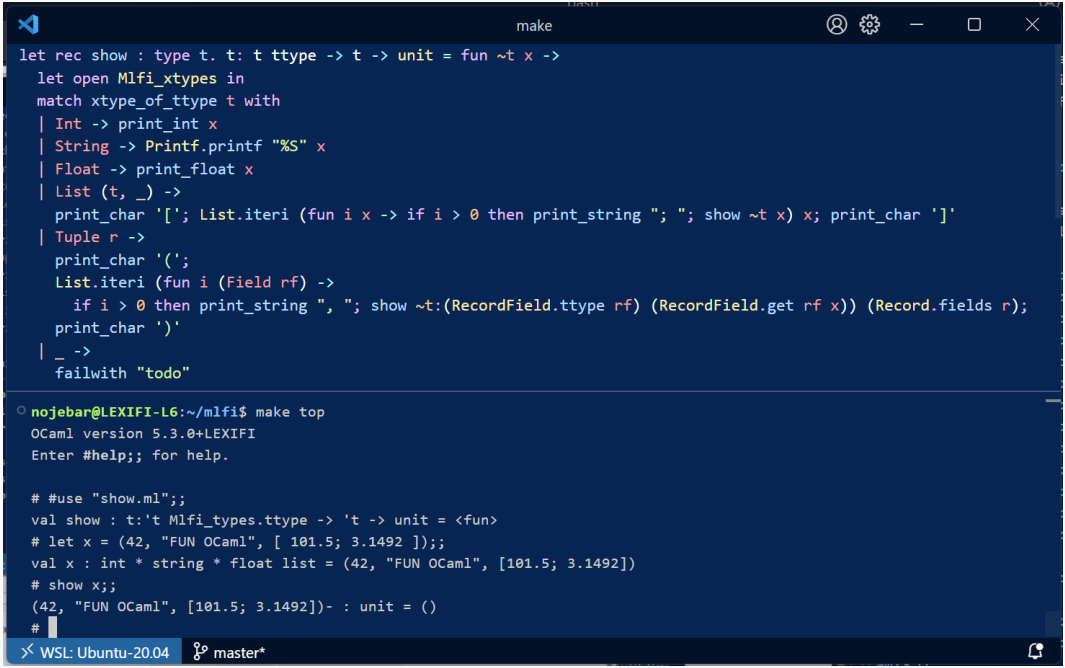
Source code deployments.

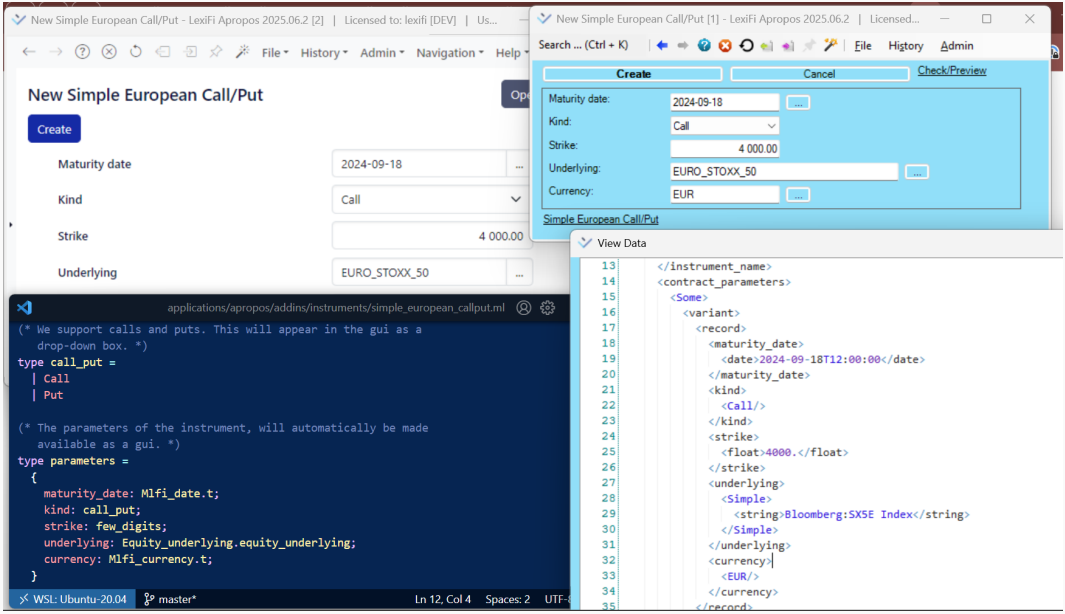
- Some large technology clients licensed LexiFi's technology in source code form. These companies sometimes started internal OCaml dev teams from scratch following collaboration with LexiFi (eg Bloomberg), and their use of OCaml grew considerably over time.
- For early version of one such project, we shipped our code in static library form, which is not an easy feat to replicate with other languages.
- Other technology clients access LexiFi's technology via a standalone command-line tool. The tool can be used directly or via thin, machine-generated, C# or Java APIs.

The Secret Sauce

LexiFi maintains a fork of the OCaml compiler since the beginning (a 25-year-old fork!), extended with a form of type reflection. This small extension is a key building block of LexiFi's technology. We would absolutely not be able to be as productive without it.

The LexiFi compiler included a number of other extensions over the years. Some of them were eventually upstreamed into the official compiler (eg first-class modules), while others ended up being removed. In order to simplify maintenance and increase interoperability with the larger OCaml ecosystem, our strategy for a number of years now has been to reduce the diff with the official compiler as much as possible. In particular, we no longer extend/modify OCaml's syntax.


make
let rec show : type t. t: t ttype -> t -> unit = fun ~t x ->
let open Mlfi_xtypes in
match xtype_of_ttype t with
| Int -> print_int x
| String -> Printf.printf "%S" x
| Float -> print_float x
| List (t, _) ->
print_char '['; List.iteri (fun i x -> if i > 0 then print_string "; "; show ~t x) x; print_char ']'
| Tuple r ->
print_char '(';
List.iteri (fun i (Field rf) ->
if i > 0 then print_string ", "; show ~t:(RecordField.ttype rf) (RecordField.get rf x)) (Record.fields r);
print_char ')'
| _ ->
failwith "todo"
nojebar@LEXIFI-L6:~/mlfi$ make top
OCaml version 5.3.0+LEXIFI
Enter #help;; for help.
#use "show.ml";;
val show : t:'t Mlfi_types.ttype -> 't -> unit = <fun>
let x = (42, "FUN OCaml", [101.5; 3.1492]);;
val x : int * string * float list = (42, "FUN OCaml", [101.5; 3.1492])
show x;;
(42, "FUN OCaml", [101.5; 3.1492])- : unit = ()
#
WSL: Ubuntu-20.04
master*


New Simple European Call/Put - LexiFi Apropos 2025.06.2 [2] | Licensed to: lexifi [DEV] | Us...
File History Admin Navigation Help
New Simple European Call/Put
Create
Maturity date 2024-09-18
Kind Call
Strike 4 000.00
Underlying EURO_STOXX_50
New Simple European Call/Put [1] - LexiFi Apropos 2025.06.2 | Licensed...
Search ... (Ctrl + K)
File History Admin
Create
Cancel
Check/Preview
Maturity date: 2024-09-18
Kind: Call
Strike: 4 000.00
Underlying: EURO_STOXX_50
Currency: EUR
Simple European Call/Put
applications/apropos/addins/instruments/simple_european_callput.ml
(* We support calls and puts. This will appear in the gui as a
drop-down box. *)
type call_put =
| Call
| Put
(* The parameters of the instrument, will automatically be made
available as a gui. *)
type parameters =
{
maturity_date: Mlfi_date.t;
kind: call_put;
strike: few_digits;
underlying: Equity_underlying.equity_underlying;
currency: Mlfi_currency.t;
}
WSL: Ubuntu-20.04
master*
Ln 12, Col 4
Spaces: 2
UTF-
View Data
</instrument_name>
<contract_parameters>
<Some>
<variant>
<record>
<maturity_date>
<date>2024-09-18T12:00:00</date>
</maturity_date>
<kind>
<Call/>
</kind>
<strike>
<float>4000.</float>
</strike>
<underlying>
<Simple>
<string>Bloomberg:SX5E Index</string>
</Simple>
</underlying>
<currency>
<EUR/>
</currency>
</record>

The `~t` argument is synthesized by the compiler. This mechanism allows using ordinary OCaml code to write type-deriving functions:

- generic printing routine
- codecs to- and from- all kinds of formats (JSON/XML/etc)
- deriving GUIs from type definitions
- deriving SQL schemas, queries from type definitions
- and more

Unlike `ppx_deriving`, the integration with the typecker makes it very ergonomic (the safety of OCaml with the usability of Python?). New "deriving" functionality is as difficult as writing an ordinary function. Developers can start using it moments after learning it.

If you want to learn more about this, come to tomorrow's workshop!

Human Aspects

- Hiring good OCaml programmers takes a bit of patience but is doable. The language seems to attract good quality candidates: high competency/candidate ratio.
- By comparison, recently we tried to hire a Python/DevOps programmer and had the opposite problem: lots of candidates, but low competency/candidate ratio.
- Quants: our hires typically have only a general software engineering background, and no knowledge of OCaml when they arrive. They are able to learn it and become productive quickly.
- We have a good experience retaining hires: very low turnover, most of developers seem happy to write OCaml for a living (a good working atmosphere and varied set of tasks helps a lot as well).
- Mostly hire people who learned functional programming in University. Most hires are junior or mid-level developers, who evolve and grow while working at LexiFi.

Conclusion

OCaml is not just for static analyzers, compilers, etc... It is possible, and pleasant, to use OCaml to build ordinary desktop applications, client/server software, web services, etc... and to do so commercially.

- A simple language with a simple cost model, where it is easy to track how much time and space is used.
- A compiler that produces efficient code that looks like the source code, with only predictable optimizations.
- A low-latency garbage collector
- Excellent stability of the language and standard library (backwards compatibility).
- Same language for the "back" and the "front" (thanks to `js_of_ocaml`) is a productivity boost, as it allows easy sharing of code between both ends.

Thanks! Interested?
https://www.lexifi.com/careers