

ErIIDE

A modern GUI for erlang development

Eclipse

- An extensible platform for developing applications
- 100% Open Source
- 100% java
- created by IBM
- Many platforms
(Windows, Mac, Linux, Solaris)

Why Eclipse?

- Mature framework
- Large ecosystem of useful tools
- A picture is worth a thousand words

- Somewhat bloated and memory hungry.
Better with Java 6.
- Impedance mismatch with Erlang

ErlIDE

- A set of plugins for Eclipse
- Project started in 2000, restarted in 2002 and getting "real" from 2005.
- Mainly 2 developers, we'd like to see more!

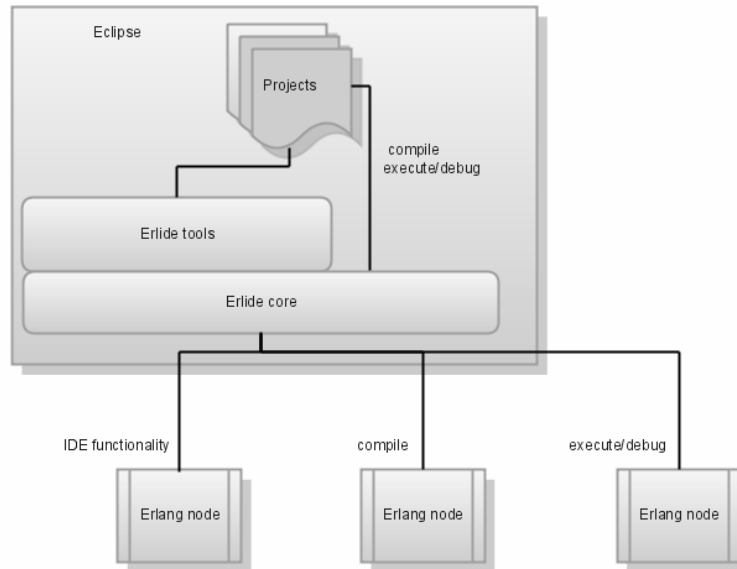
ErIIDE

- Originally 100% Java
- Code: 80% Java, 20% Erlang
- Functionality: 50% java, 50% Erlang

Architecture

- Eclipse workspace uses an Erlang backend to do the work
- Each project may be compiled by a separate backend
- Another backend is used to run the projects

Architecture



Architecture

- Erlang side provides services to Java side
 - RPC
 - event based
- Infrastructure to make communication as seamless as possible
 - implement Java interfaces in Erlang

ErIIDE and OTP

- Requires R11B-5 or later
- Special versions of jinterface, scanner, parser, syntax_tools, edoc, debugger

Features

- Editor
- Indentation of Erlang code
- Syntactic highlighting
- Bracket matching
- Automatic completion of erlang functions and records
- Selective display of functions and declarations (folding)
 - Indentation and code formatting
- Automatic indentation
- Formatting and pretty-printing

Features

- Hover
 - OTP documentation shown for external calls
 - Show declaration of macros and records
- Outline
 - Code outline of the structure of an Erlang module
 - Filtering of Erlang functions and declarations
 - Quick outline for selecting Erlang function or declaration

Features

- Navigation
 - Go to declaration of function, macro or record
- Project Import
 - Erlang-aware import of projects
- Creation wizards
 - Create Erlang projects
 - Create Erlang modules with code skeletons

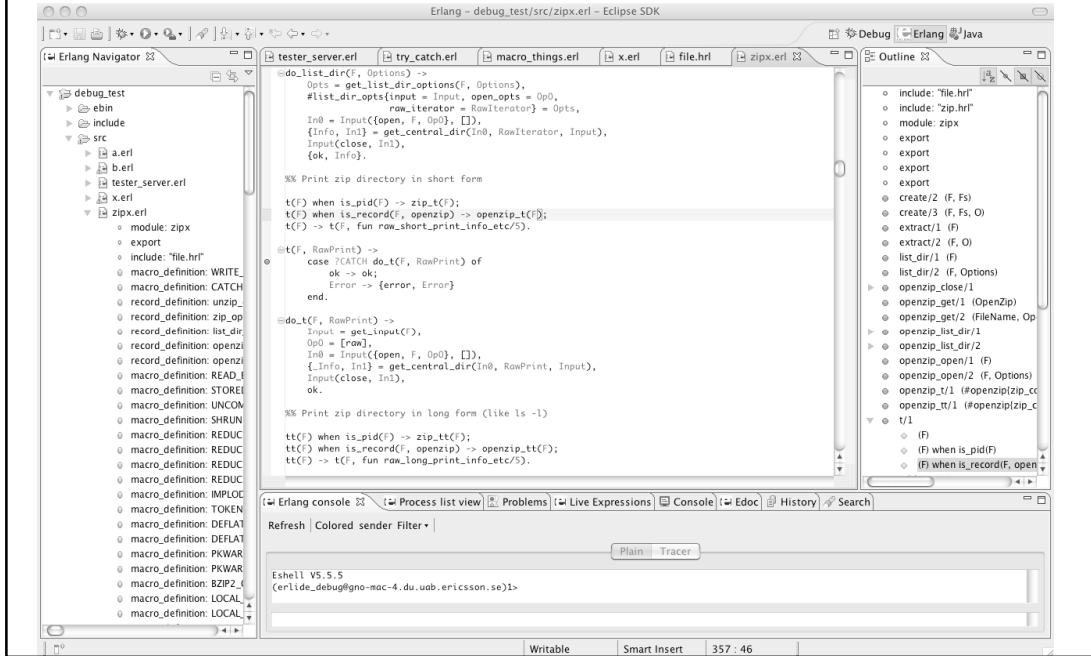
Features

- Debugger
 - Erlang debugger within eclipse debugger framework
 - Breakpoints, single-stepping
 - Inspection and modification of local variables
 - Distributed debugger, debug on multiple nodes

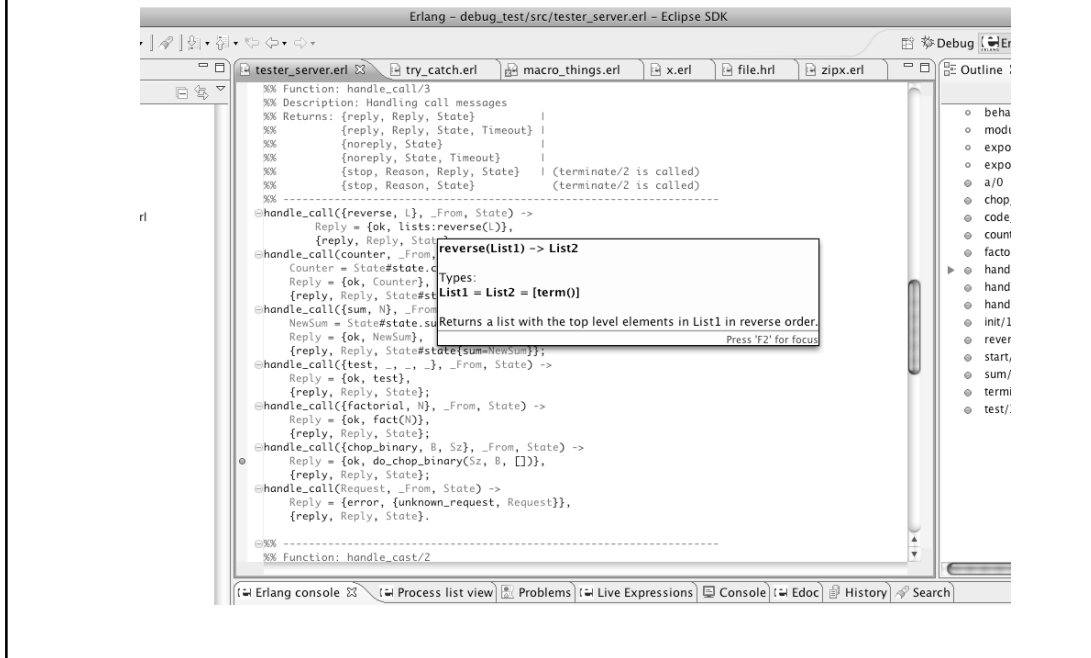
Features

- Erlang runtime handling
 - Provision for different runtimes, local or remote
 - Develop on one runtime, test and debug on another
- Warnings, errors, TO-DO-markers
 - Show errors and warnings in code and in problems view
 - Automatically mark and list comments with TODO and FIXME
 - Erlang log printouts with link to code line

Demonstration



Demonstration



Demonstration

The screenshot shows an Erlang IDE with a code editor on the left and a function tooltip on the right. The code in the editor includes:

```

%% Returns: {ok, NewState}
%%
code_change(_OldVsn, State, _Extra) ->
{ok, State}.

%% Internal functions
%%
fact(0) ->
1;
fact(N) when is_integer(N) ->
M = N*fact(N-1),
M.

do_chop_binary(Sz, Bin, Acc) ->
case Bin of
<<:Sz/binary, Rest/binary>> ->
do_chop_binary(Sz, Rest, [B | Acc]);
<<>> ->
lists:reverse(Acc);
_ ->
end.

lists:flatten/2
lists:flat_length/1
lists:flatlength/1
lists:flatmap/2
lists:foldr/3
lists:flatmap/3
lists:foldl/4
lists:foldr/4
lists:filter/3
lists:foreach/3
lists:flatten/1

```

The tooltip for `foldr(Fun, Acc0, List) -> Acc1` contains the following information:

- Types: `Fun = fun(Elem, AccIn) -> AccOut`, `Elem = term()`, `Acc0 = Acc1 = AccIn = AccOut = term()`, `List = [term()]`
- Like `foldl/3`, but the list is traversed from right to left.
- For example:
 - `> P = fun(A, AccIn) -> io:format("~p ", [A]), AccIn end.`
 - `#Fun<erl_eval.12.2225172>`
 - `> lists:foldl(P, void, [1,2,3]).`
 - `1 2 3 void`

Demonstration

The screenshot shows the Eclipse IDE with an Erlang project. The code in the editor includes:

```

tt(F) when is_pid(F) -> zip_tt(F);
tt(F) when is_record(F, openzip) -> openzip_tt(F);
tt(F) -> t(F, fun raw_long_print_info_etc/5).

%% option utils
get_unzip_opt([], Opts) ->
Opts;
get_unzip_opt([verbose | Rest], Opts) ->
get_unzip_opt(Rest, Opts#unzip_opts{feedback = fun verbose_unzip/1});
get_unzip_opt([cooked | Rest], Opts) ->
get_unzip_opt(Rest, Opts#unzip_opts{cooked = true});
get_unzip_opt([unzip | Rest], Opts) ->
get_unzip_opt(Rest, Opts#unzip_opts{unzip = true});
get_unzip_opt([memo | Rest], Opts) ->
get_unzip_opt(Rest, Opts#unzip_opts{memo = true});
get_unzip_opt([cwd | Rest], Opts) ->
get_unzip_opt(Rest, Opts#unzip_opts{cwd = true});
get_unzip_opt([filter | Rest], Opts) ->
Filter = fun_an(Rest, Opts),
get_unzip_opt(Rest, Opts#unzip_opts{filter = Filter});
get_unzip_opt([files | Rest], Opts) ->
FileInList = fun_an(Rest, Opts),
Filter = fun_an(Rest, Opts),
get_unzip_opt(Rest, Opts#unzip_opts{files = FileInList, filter = Filter});
get_unzip_opt([keep | Rest], Opts) ->
Keep = fun_an(Rest, Opts),
Filter = fun_an(Rest, Opts),
get_unzip_opt(Rest, Opts#unzip_opts{keep = Keep, filter = Filter});
get_unzip_opt([unknown | _Rest], _Opts) ->
throw({bad_option, Unknown}).

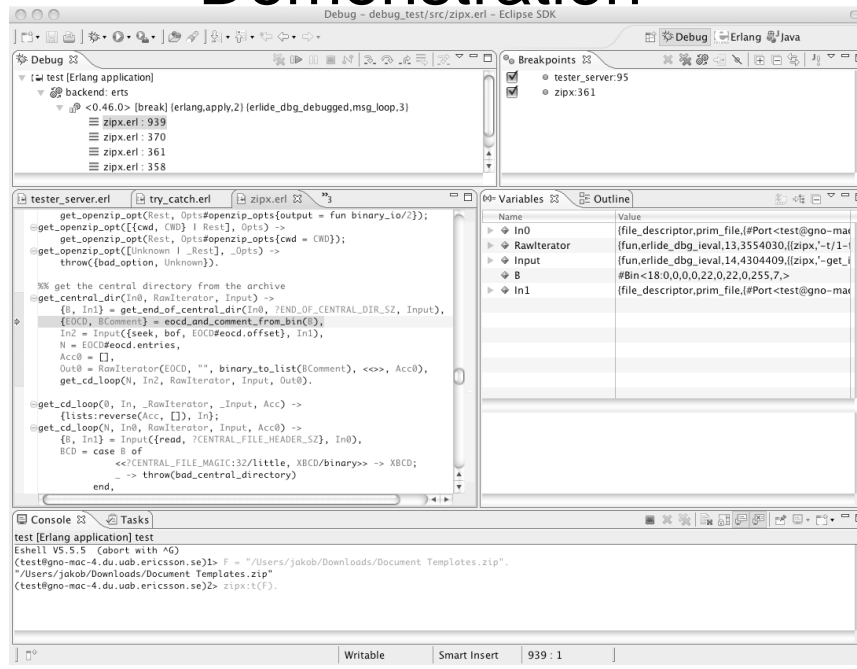
get_list_dir_opt([], Opts) ->
Opts;
get_list_dir_opt([cooked | Rest], #list_dir_opts{open_opts = OpO} = Opts) ->
get_list_dir_opt(Rest, Opts#list_dir_opts{open_opts = OpO -- [raw]});

```

The tooltip for `do_unzip/2 (F, Options)` contains the following information:

- record_definition: `unzip_opts`
- unzip/1 (F)
- unzip/2 (F, Options)
- do_unzip/2 (F, Options)
 - get_unzip_opt/2
 - [(cooked | Rest), #unzip_opts{open_opts = OpO} = OpO]
 - verbose_unzip/1 (FN)
 - get_unzip_options/2 (F, Options)
 - get_z_files/5
 - [(#zip_file{offset = Offset}) = ZipFile | Rest], Z, InO, #unzip_opts{open_opts = OpO}

Demonstration



Some future features

- Semantic highlighting in editor (function calls, variable usage, etc)
- Show documentation of user functions
- Erlang-aware search
- Refactoring
- xref-based indexing and searching
- Tracing viewer
- Syntax error correction
- What tools do you need?

Resources

- <http://erlide.sourceforge.net>
- <http://erlide.sourceforge.net/update>

- <http://eclipse.org>
- <http://erlang.org>