ABL Code Analytics

Unique Tools for Deep Code Analysis and Inspection

Greg Shah
Golden Code Development

Thursday October 25, 2018

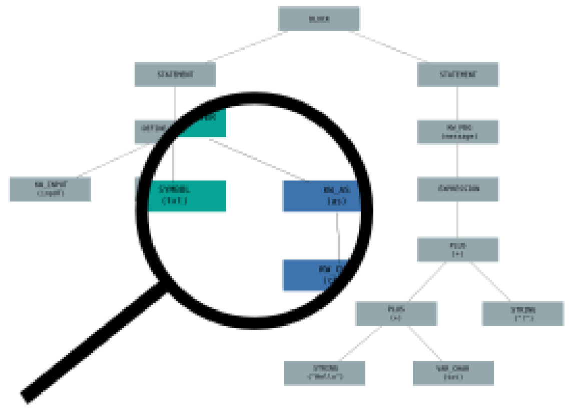
Agenda

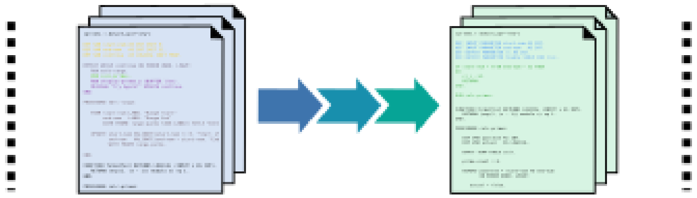
- Background
- Handling Flexibility and Scale
- Code Analytics
- Call Graph Analyzer
- Usage Tips
- How to Get Started
- Planned Improvements

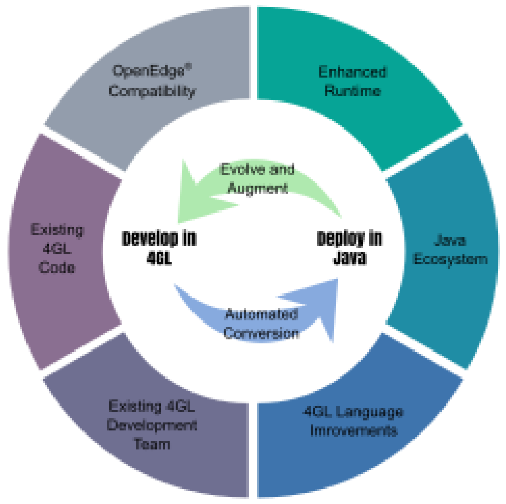
FWD analytics

Background

FWD is an **open source** toolset for **modernizing** 4GL applications.







Explore

Tools for reporting, statistics and analysis, which expose the inner workings of applications.

Transform

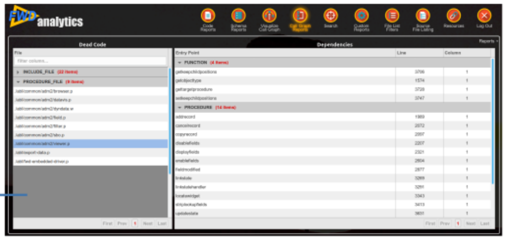
Fully automated transformation and modernization of entire applications without a manual rewrite.

Transcend

Compatible replacement for OpenEdge® which leverages an enhanced runtime, an upgraded 4GL language and Java to modernize and evolve applications.

FWD analytics Tools for reporting, statistics and analysis, which **expose the inner workings of an application**.

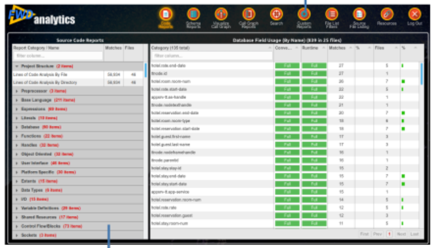
Search for arbitrary 4GL syntax to find exact answers to questions that were previously difficult or impossible to answer.



Call Graph Analysis determines program dependencies, missing code and explores all reachable code paths. Reports allow identification and removal of dead code reducing application size by 25% to 40%

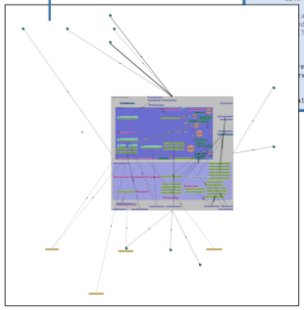
Explore

Build custom reports to meet specific needs.

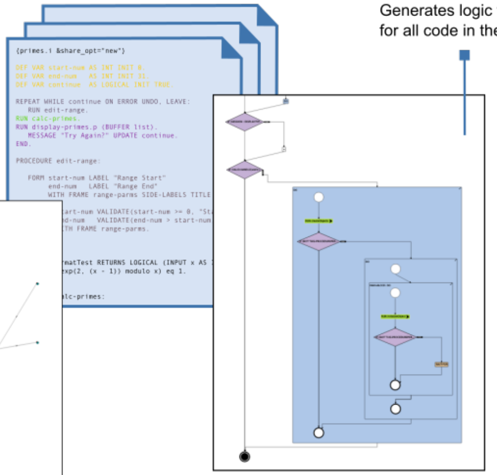


Built-in reports and statistics provide hundreds of predefined views of the code and schemata.

Visualize and interactively explore the call graph.



Generates logic flow charts for all code in the application.



```
{primes.i &share_opt="new"}

DEF VAR start-num AS INT INIT 0.
DEF VAR end-num   AS INT INIT 31.
DEF VAR continue  AS LOGICAL INIT TRUE.

REPEAT WHILE continue ON ERROR UNDO, LEAVE:
   RUN edit-range.
RUN calc-primes.
RUN display-primes.p (BUFFER list).
   MESSAGE "Try Again?" UPDATE continue.
END.

PROCEDURE edit-range:

   FORM start-num LABEL "Range Start"
        end-num   LABEL "Range End"
        WITH FRAME range-parms SIDE-LABELS TITLE

        tart-num VALIDATE(start-num >= 0, "St
        d-num    VALIDATE(end-num > start-num
        ITH FRAME range-parms.



        rmatTest RETURNS LOGICAL (INPUT x AS
        exp(2, (x - 1)) modulo x) eq 1.


        alc-primes:
```

Advantages

- Reduce development effort.
- Improve code quality.
- Deeply understand existing code.
- Compensate for missing documentation,
- Empower developers to more capably handle:
  - The most complex refactoring and modernization projects.
  - Making changes at scale, even with the largest of applications.

Why Use FWD Analytics?

- Reduce development effort.
- Improve code quality.
- Deeply understand and explore existing code.
- Compensate for missing documentation.
- Empower developers to more capably handle:
  - The most complex refactoring, transformation and modernization problems; AND
  - Making changes at scale, even with the largest of applications.

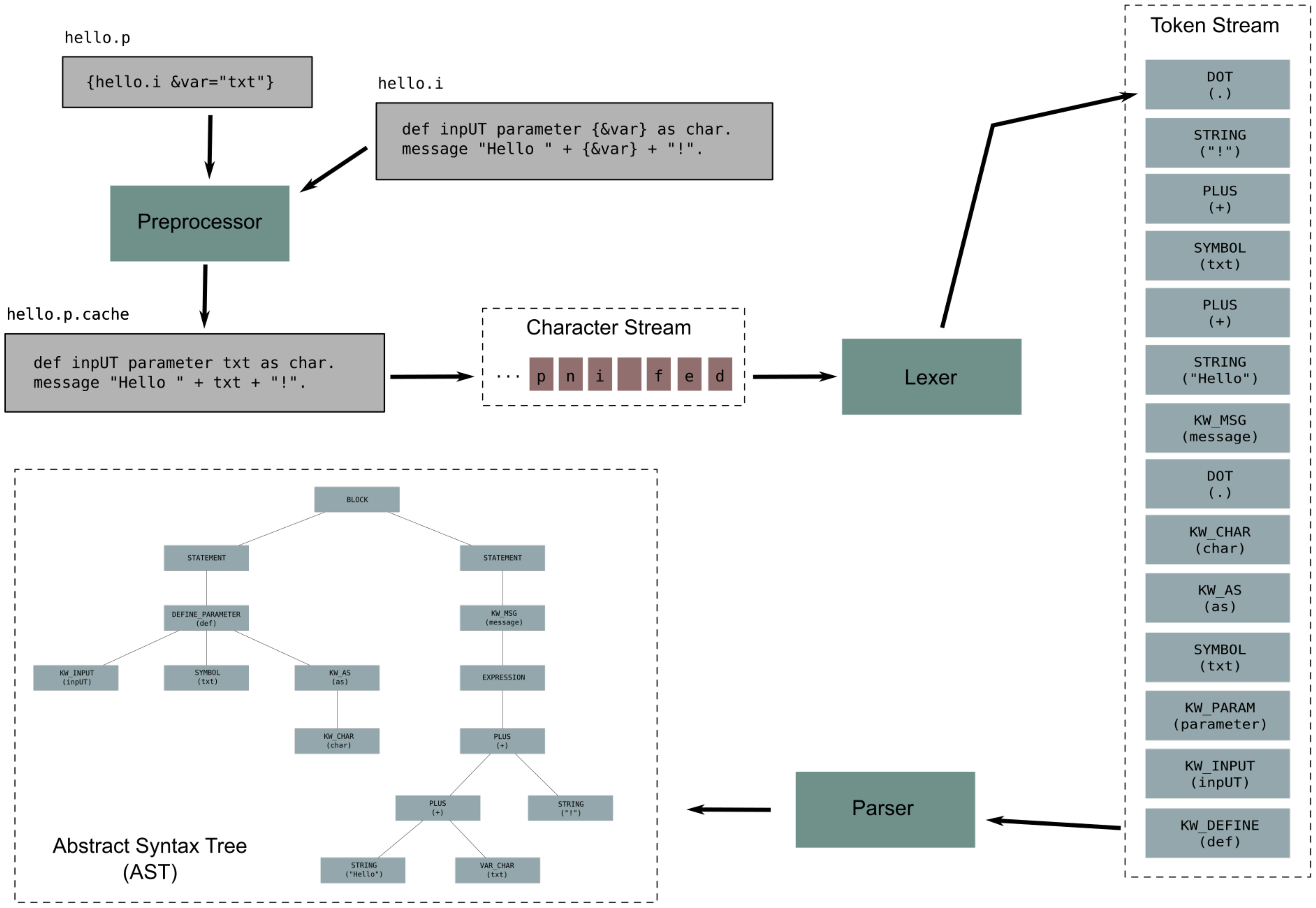
FWD analytics

Handling Flexibility and Scale

Your Source Is Not Helping

- Programmatic analysis of an application needs to be aware of the ABL language syntax.
- Your source code is text. That text is non-regular and ambiguous.
  - different text, same meaning (non-regular code)
  - same text different meaning (ambiguous code)
- The more flexible the language's grammar, the larger the set of possible valid constructs that can be written by the programmer.
  - Short Term: marginal time savings when writing new code.
  - Long Term: Increased costs of reading, maintenance, debugging, support and refactoring.
- To enable proper analysis of code, we must transform the text into a data structure that represents the purest form of the code.
- ASTs represent the code's language syntax without syntactic sugar. The result is regular and unambiguous.

File -> Char -> Token -> Tree



hello.p

```
{hello.i &var="txt"}
```

hello.i

```
def inpUT parameter {&var} as char.
message "Hello " + {&var} + "!".
```

hello.p.cache

```
def inpUT parameter txt as char.
message "Hello " + txt + "!".
```

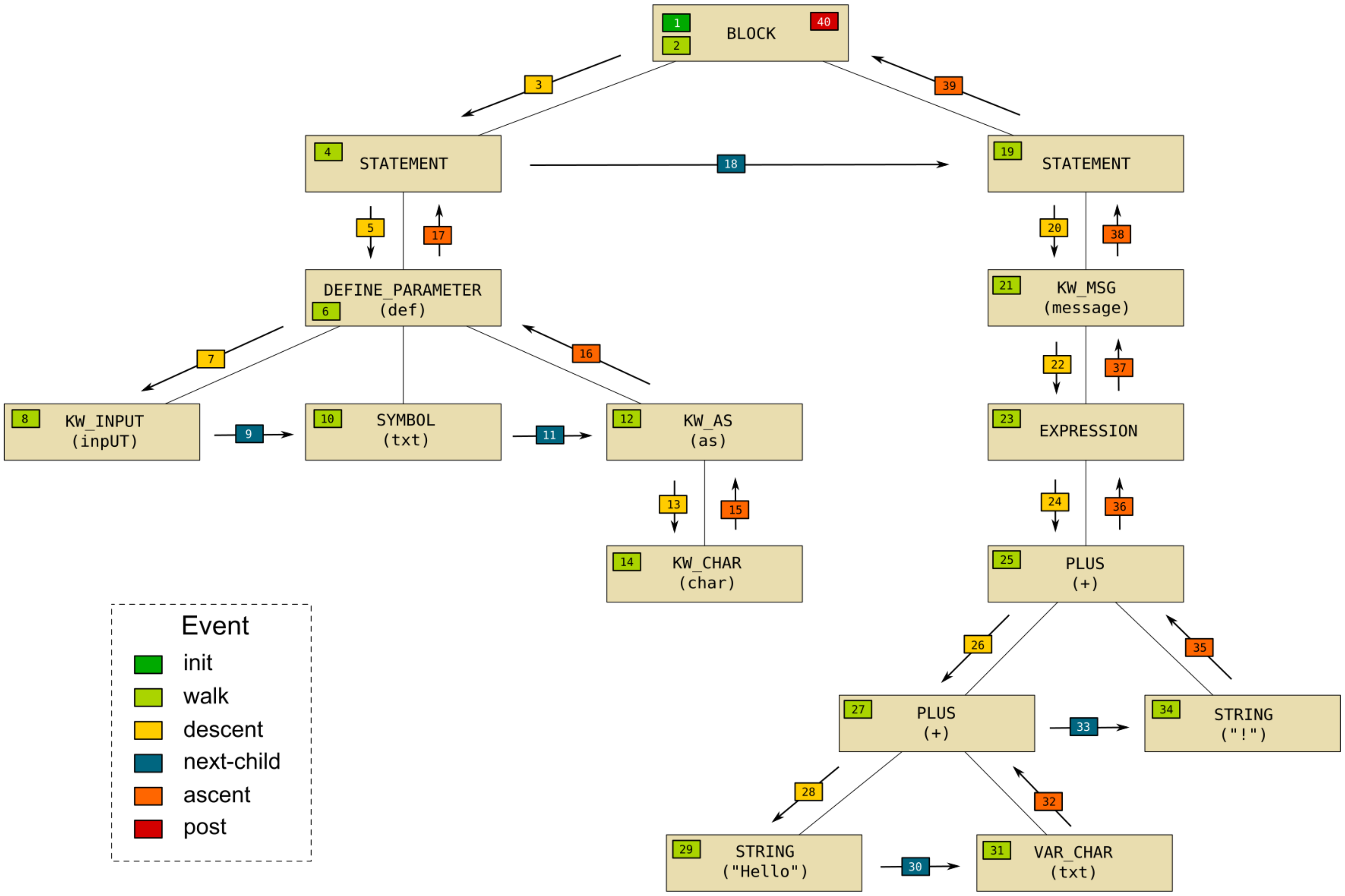
Scale Multiplies Issues

- Spread that syntax flexibility across tens of thousands of files and millions of lines of code.
- How do you find specific patterns?
- How do you even know if you have found all matches?
- Brute force is not enough.
- **Scale makes hard problems impossible (or at least impractical).**
- Automation is the only practical solution.

TRee Processing Language (TRPL)

- FWD provides tools to parse an entire application.
- Each source file and each schema file (.df) will be represented as an AST.
- TRPL is the analysis and transformation toolset in FWD which can operate on the entire set of ASTs as a batch.
- When you process trees, it is commonly called a tree walk.
- TRPL includes an engine that handles the tree walking for programs written in the TRPL language.

TRee Processing Language (TRPL) Event Model



AST Design for Transformation

- At parse time, there is a great deal of knowledge about the code. Encoding that knowledge into the tree makes downstream work easier.
- Resolving data types of each expression component is very important. This allows downstream code to calculate the type of each subexpression or expression in the application.
- By tracking resources by scope and creating linkages between the references and the definition, it becomes easier to work with these resources later.
- Structuring the tree is important. This can make it easier to walk the tree, match patterns and transform.
  - Multiple nodes can be rewritten as a single unambiguous node (e.g. KW_DEFINE KW_PARAMETER can be written as DEFINE_PARAMETER).
  - Artificial nodes can be inserted to group multiple related nodes.
- Calculated values and context-specific information are stored in the associated nodes as annotations.
- The ASTs created by FWD were designed with these issues (and others) in mind.

Report Generation

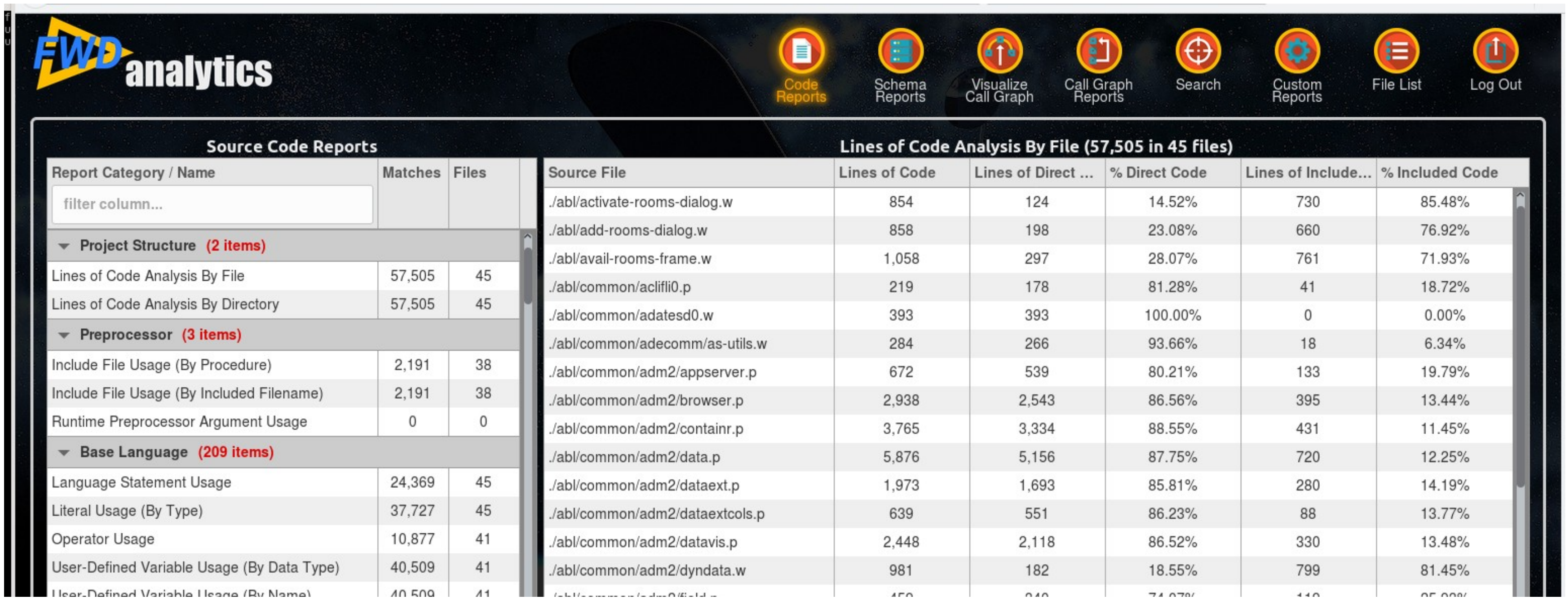
- After the entire application has been parsed, we can run the report generation step.
- This is a non-interactive process that runs a set of pre-defined TRPL programs to calculate a few hundred reports.
- This can take minutes for a small project or hours for a large project.
- Both the parsing and the report generation can be scripted and used in CI or build servers.
- After the reports are generated, they can be accessed via an interactive web interface.

FWD analytics

Code Analytics

Reports

- List of predefined reports on left
- Currently viewed report on right
- Most reports are a set of mutually exclusive categories
- Summary statistics for the report at the top
- Individual categories have their own statistics
- Filter and sort columns using the column header
- Click on a row in the current report to see the exact list of matches
- Pagination controls at the bottom



Source Code Reports

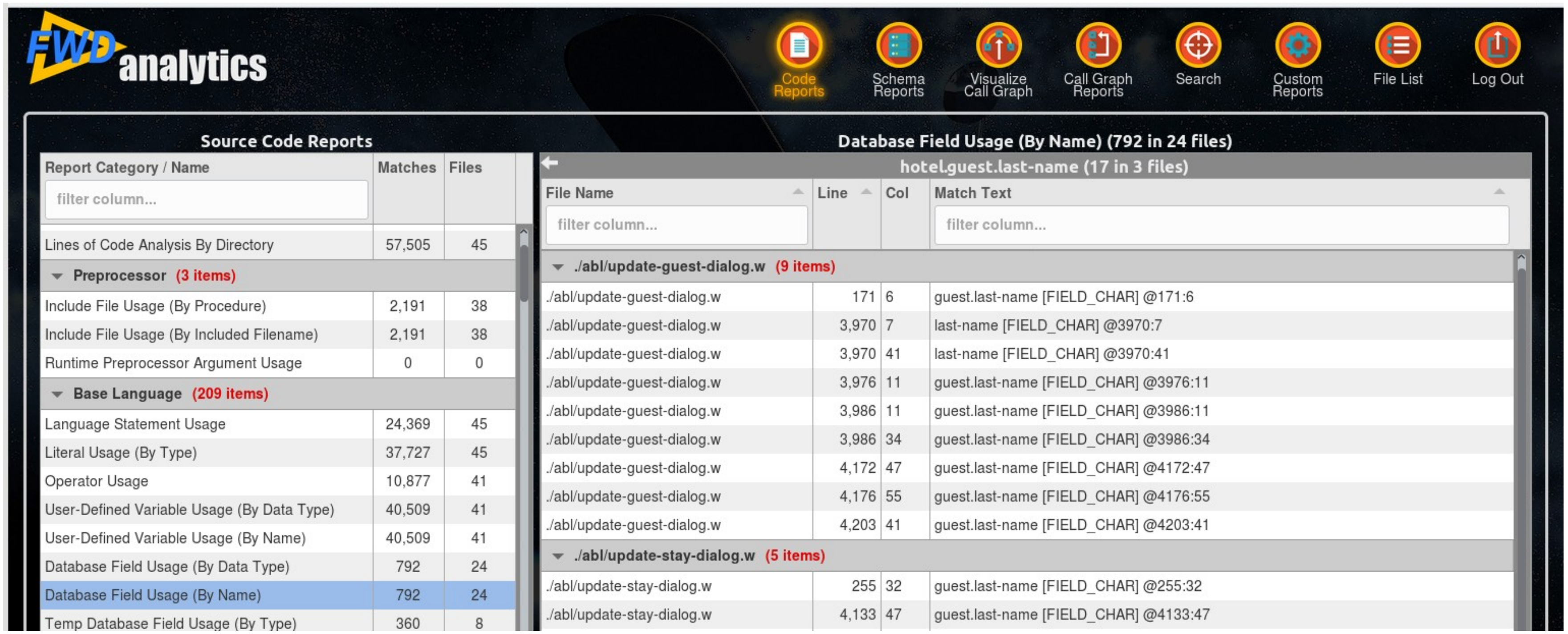
Report Category / Name	Matches	Files
filter column...		
Project Structure (2 items)		
Lines of Code Analysis By File	57,505	45
Lines of Code Analysis By Directory	57,505	45
Preprocessor (3 items)		
Include File Usage (By Procedure)	2,191	38
Include File Usage (By Included Filename)	2,191	38
Runtime Preprocessor Argument Usage	0	0
Base Language (209 items)		
Language Statement Usage	24,369	45
Literal Usage (By Type)	37,727	45
Operator Usage	10,877	41
User-Defined Variable Usage (By Data Type)	40,509	41
User-Defined Variable Usage (By Name)	40,509	41

Lines of Code Analysis By File (57,505 in 45 files)

Source File	Lines of Code	Lines of Direct ...	% Direct Code	Lines of Include...	% Included Code
./abl/activate-rooms-dialog.w	854	124	14.52%	730	85.48%
./abl/add-rooms-dialog.w	858	198	23.08%	660	76.92%
./abl/avail-rooms-frame.w	1,058	297	28.07%	761	71.93%
./abl/common/aclifli0.p	219	178	81.28%	41	18.72%
./abl/common/adatesd0.w	393	393	100.00%	0	0.00%
./abl/common/adecomm/as-utils.w	284	266	93.66%	18	6.34%
./abl/common/adm2/appserver.p	672	539	80.21%	133	19.79%
./abl/common/adm2/browser.p	2,938	2,543	86.56%	395	13.44%
./abl/common/adm2/containr.p	3,765	3,334	88.55%	431	11.45%
./abl/common/adm2/data.p	5,876	5,156	87.75%	720	12.25%
./abl/common/adm2/dataext.p	1,973	1,693	85.81%	280	14.19%
./abl/common/adm2/dataextcols.p	639	551	86.23%	88	13.77%
./abl/common/adm2/datavis.p	2,448	2,118	86.52%	330	13.48%
./abl/common/adm2/dyndata.w	981	182	18.55%	799	81.45%
./abl/common/adm2/field.p	459	340	74.07%	119	25.93%

Category Details

- List of predefined reports on left
- Exact list of matches for the selected category on the right
- Grouped by the file in which they appear
- Category statistics at the top
- Each match has line/column numbers in the "cache" file (fully preprocessed file)
- Filter, sort and pagination controls
- Click on a row of a specific match to go to the source view at that exact location



Source Code Reports

Report Category / Name	Matches	Files
filter column...		
Lines of Code Analysis By Directory	57,505	45
Preprocessor (3 items)		
Include File Usage (By Procedure)	2,191	38
Include File Usage (By Included Filename)	2,191	38
Runtime Preprocessor Argument Usage	0	0
Base Language (209 items)		
Language Statement Usage	24,369	45
Literal Usage (By Type)	37,727	45
Operator Usage	10,877	41
User-Defined Variable Usage (By Data Type)	40,509	41
User-Defined Variable Usage (By Name)	40,509	41
Database Field Usage (By Data Type)	792	24
Database Field Usage (By Name)	792	24
Temp Database Field Usage (By Type)	360	8

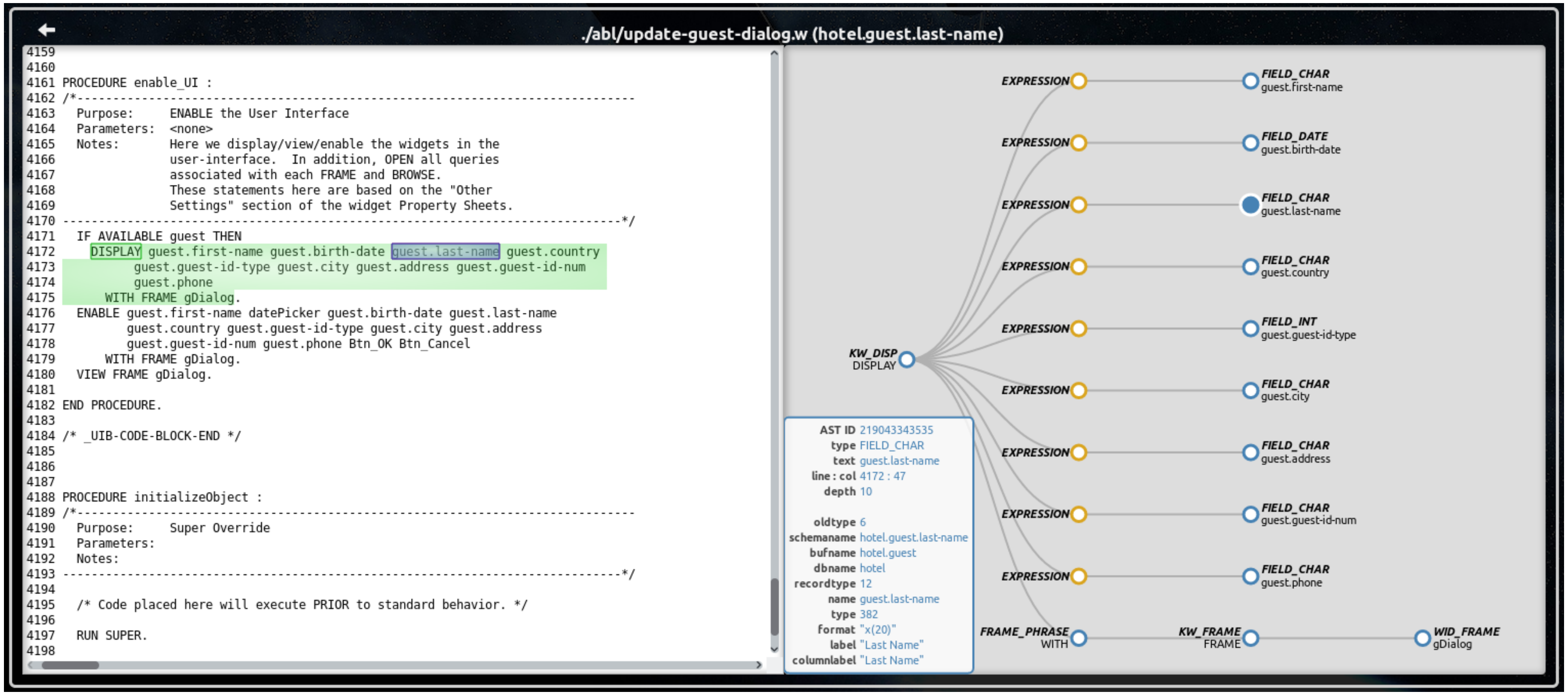
Database Field Usage (By Name) (792 in 24 files)

hotel.guest.last-name (17 in 3 files)

File Name	Line	Col	Match Text
filter column...			filter column...
./abl/update-guest-dialog.w (9 items)			
./abl/update-guest-dialog.w	171	6	guest.last-name [FIELD_CHAR] @171:6
./abl/update-guest-dialog.w	3,970	7	last-name [FIELD_CHAR] @3970:7
./abl/update-guest-dialog.w	3,970	41	last-name [FIELD_CHAR] @3970:41
./abl/update-guest-dialog.w	3,976	11	guest.last-name [FIELD_CHAR] @3976:11
./abl/update-guest-dialog.w	3,986	11	guest.last-name [FIELD_CHAR] @3986:11
./abl/update-guest-dialog.w	3,986	34	guest.last-name [FIELD_CHAR] @3986:34
./abl/update-guest-dialog.w	4,172	47	guest.last-name [FIELD_CHAR] @4172:47
./abl/update-guest-dialog.w	4,176	55	guest.last-name [FIELD_CHAR] @4176:55
./abl/update-guest-dialog.w	4,203	41	guest.last-name [FIELD_CHAR] @4203:41
./abl/update-stay-dialog.w (5 items)			
./abl/update-stay-dialog.w	255	32	guest.last-name [FIELD_CHAR] @255:32
./abl/update-stay-dialog.w	4,133	47	guest.last-name [FIELD_CHAR] @4133:47

Source/AST View

- Fully preprocessed file on left with the match selected in pink.
- Current selection in the AST on the right.
- Source and AST views are linked, a selection on either side is highlighted and made visible on the other side.
- Hover mouse over an AST node to get details.
- Shift-click on the "root" node of the subtree to traverse up the tree.
- Ctrl-click on a child node to traverse down the tree.



./abl/update-guest-dialog.w (hotel.guest.last-name)

```
4159
4160
4161 PROCEDURE enable_UI :
4162 /*------------------------------------------------------------------------------
4163   Purpose:     ENABLE the User Interface
4164   Parameters:  <none>
4165   Notes:       Here we display/view/enable the widgets in the
4166                user-interface.  In addition, OPEN all queries
4167                associated with each FRAME and BROWSE.
4168                These statements here are based on the "Other
4169                Settings" section of the widget Property Sheets.
4170 ------------------------------------------------------------------------------*/
4171   IF AVAILABLE guest THEN
4172     DISPLAY guest.first-name guest.birth-date guest.last-name guest.country
4173           guest.guest-id-type guest.city guest.address guest.guest-id-num
4174           guest.phone
4175       WITH FRAME gDialog.
4176   ENABLE guest.first-name datePicker guest.birth-date guest.last-name
4177          guest.country guest.guest-id-type guest.city guest.address
4178          guest.guest-id-num guest.phone Btn_OK Btn_Cancel
4179       WITH FRAME gDialog.
4180   VIEW FRAME gDialog.
4181
4182 END PROCEDURE.
4183
4184 /* _UIB-CODE-BLOCK-END */
4185
4186
4187
4188 PROCEDURE initializeObject :
4189 /*------------------------------------------------------------------------------
4190   Purpose:     Super Override
4191   Parameters:
4192   Notes:
4193 ------------------------------------------------------------------------------*/
4194
4195   /* Code placed here will execute PRIOR to standard behavior. */
4196
4197   RUN SUPER.
4198
```

AST ID	219043343535
type	FIELD_CHAR
text	guest.last-name
line : col	4172 : 47
depth	10
oldtype	6
schemaname	hotel.guest.last-name
bufname	hotel.guest
dbname	hotel
recordtype	12
name	guest.last-name
type	382
format	"x(20)"
label	"Last Name"
columnlabel	"Last Name"

Search

- If grep (regex searching) was fully aware of ABL syntax it would still not be as good as this.
- Write expressions or arbitrary complexity that match based on the full richness of the AST.
- The TRPL engine does the tree walk, you just specify exactly what you want to match.
- The TRPL expression syntax has many features that make it easier to process AST concepts, including the knowledge of the current AST node being visited.
- Code that cannot be implemented in a single expression can be put into a callable TRPL function and accessed from expressions.
- All AST nodes and other data being accessed are actually Java objects. You can call Java instance methods (no statics or generics at this time) on these objects and you can pass those same objects to Java methods or to TRPL functions.
- TRPL has a wide range of advanced AST processing features that can be leveraged.

Search: Field References

All references to guest.last-name:

```
type == prog.field_char and
getNoteString("schemaname").equals("hotel.guest.last-
name")
```

Assignments to guest.last-name:

```
type == prog.field_char and
getNoteString("schemaname").equals("hotel.guest.last-
name") and parent.type == prog.assign and childIndex == 0
```

Search: Buffers That Hide Buffers

Version 1:

```
type == prog.define_buffer and
this.getChildAt(0).text.toLowerCase() ==
this.getChildAt(1).getChildAt(0).text.toLowerCase()
```

Version 2:

```
parent.type == prog.kw_for and parent.parent.type ==
prog.define_buffer and
text.equalsIgnoreCase(parent.prevSibling.text)
```

Version 3:

```
upPath("DEFINE_BUFFER/KW_FOR") and
text.equalsIgnoreCase(parent.prevSibling.text)
```

Search: FIND and NO-ERROR

- All FIND statements (62 matches):

```
type == prog.kw_find
```

- FIND statements **without** NO-ERROR (26 matches)

```
type == prog.kw_find and not

this.descendant(2, prog.kw_no_error)
```

- FIND statements **with** NO-ERROR (36 matches)

```
type == prog.kw_find and
downPath("RECORD_PHRASE/KW_NO_ERROR")
```

Custom Reports

- Practice first with Custom Search
- Refine output with Custom Reports
  - Multiplex expressions to define “buckets”
  - Specify “dump” text preferences
- Persist the report definitions you find useful
- Organize by category and title
- Planned: Edit and Delete of custom reports

Custom Reports Example

- Title:

  FIND without NO-ERROR (by Buffer Name)

- Condition:

  ```
  type == prog.kw_find and parent.type == prog.statement and
  not this.descendant(2, prog.kw_no_error)
  ```

- Multiplex Expression:

  ```
  this.getImmediateChild(prog.record_phrase,
  null).getChildAt(0).getAnnotation("schemaname")
  ```

- Category:

  Database

FWD analytics

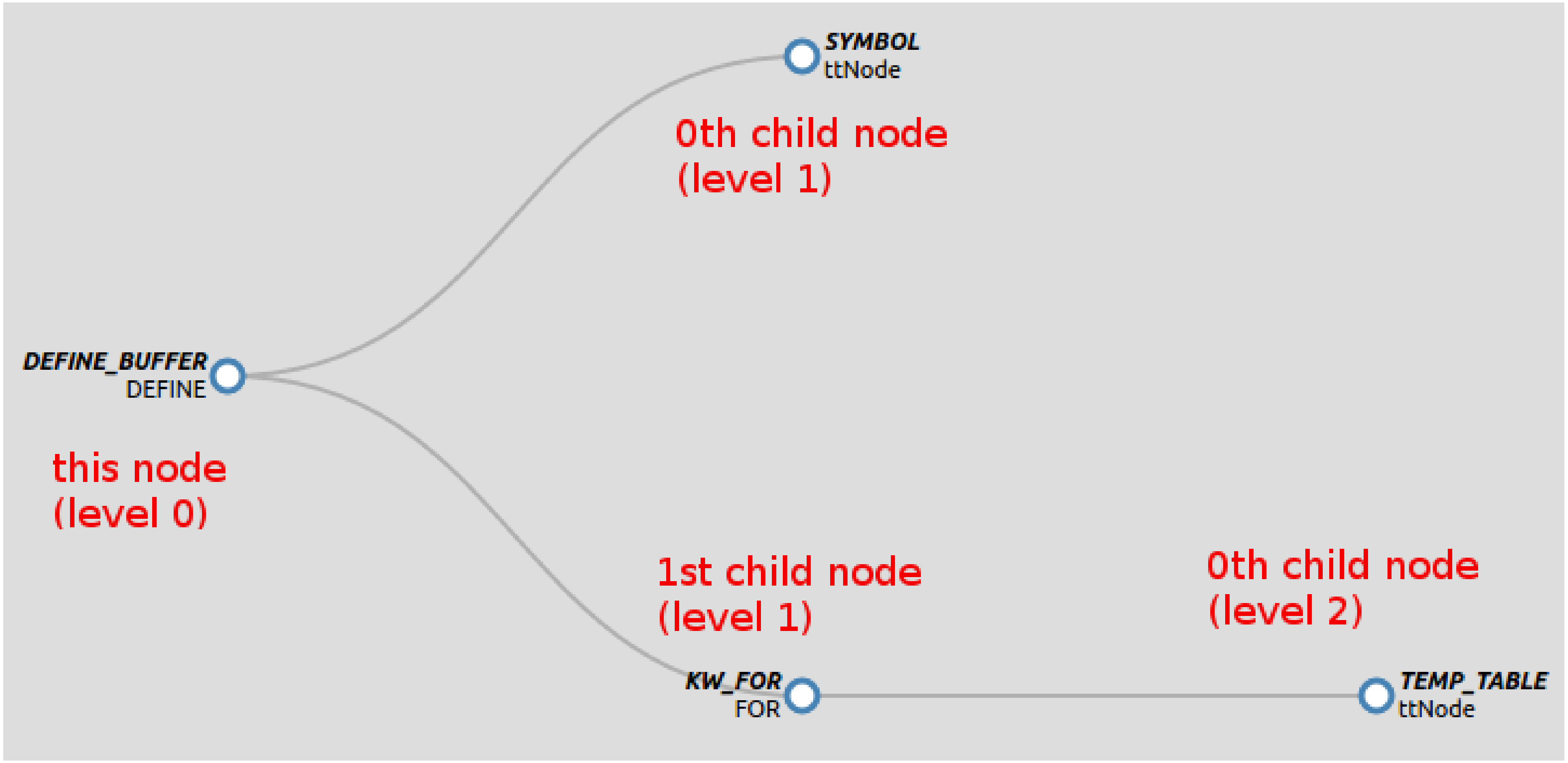
Usage Tips

Writing a Search Expression

- Look at the AST structure that corresponds to the code you are trying to match.
    - Write a code snippet and parse it, then view it in the source/AST view.
    - Use the predefined reports to find locations that already exist.
- Decide which node is the best situated. Usually this is about finding the node that is most “centrally” located.
- All the context for the expression is written from that node’s “perspective”.
- Use the token type first, to roughly match a set of possible nodes.
- Refine this to get an exact match by addng use of tree structure, annotations and text.

Look at the AST

- Tree visualization of DEFINE BUFFER



Don’t Fight the Tree!

- Let the structure of the AST solve the problem for you.
- TRPL will walk the tree for you.
- Your expression is being executed at each possible location in the entire application.
- It is a “callback” model with the events determined by the tree structure.
- The tree structure is the pure form of the language syntax as represented in your code.
- Matching on the tree is matching on the syntax.
- If you are finding yourself doing something “unnatural”, ask: how can the tree structure help me?

FWD analytics

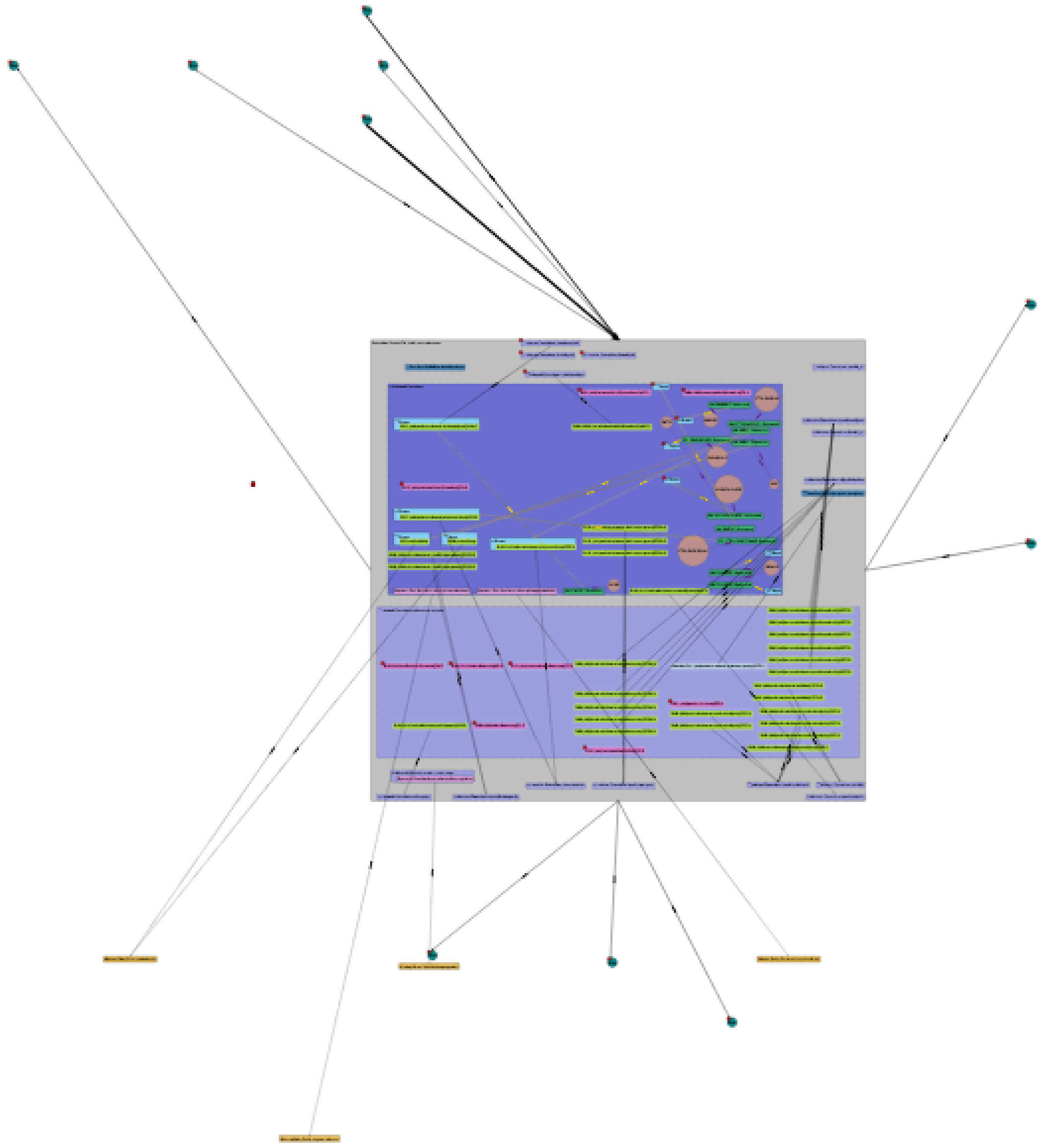
Call Graph Analyzer

Call Graph Analyzer

- Uses a graph database.
- Creates a “vertex” for every callable code block (e.g. function or internal procedure) in the application.
- Creates a “vertex” for every call-site (location that invokes one or more code blocks, e.g. RUN statement).
- Creates an “edge” between the call sites and the code blocks.
- Traversing from the a root entry point list (which you provide), we can walk the entire call graph of your application.
- This can be used to answer questions that are otherwise difficult or impossible to answer.

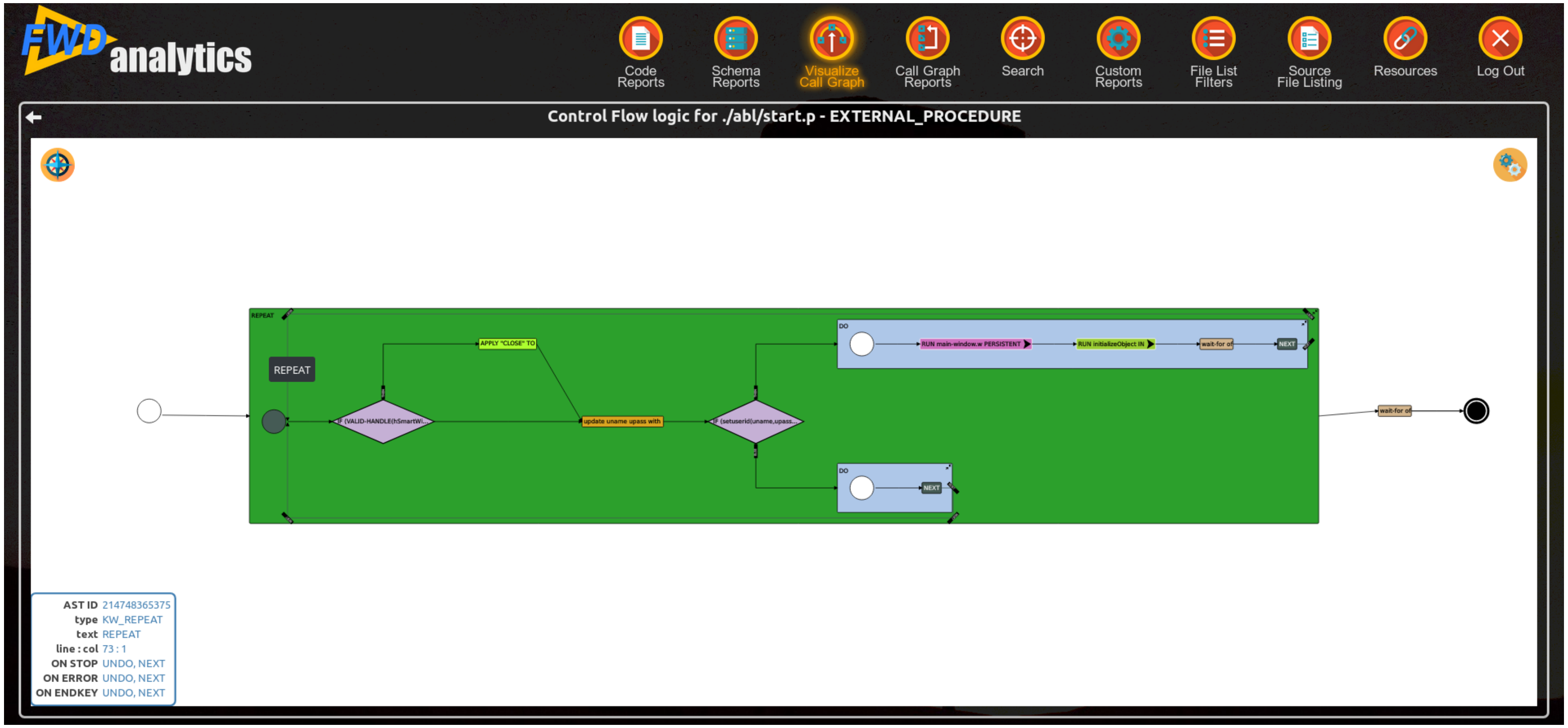
Call Graph Visualization

- Live model of the call tree using a "force directed graph".
- User can load the graph from arbitrary locations.
- Traverse to "More" links with SHIFT-click (load just that node) or CTRL-click (add node to current graph snippet).
- Use this to explore the application.
- Useful to identify macro patterns that would be hard to see by reading source code.
- Zoom with mouse wheel, pan with drag on background.
- Still in very active development, this is an early version.
- Drag nodes to move them around. Hover to see details.
- SHIFT-click on AST nodes to go to the source/AST view.



Logic Flow Charts

- Live model of the logic flow chart for every callable block of code in the application.
- Accessed via the “Y” flow icon in the call graph visualization.
- Traverse to called code from call sites such as RUN.
- This is a form of documentation.
- Useful to identify macro patterns that would be hard to see by reading source code.
- Zoom with mouse wheel, pan with drag on background.
- Drag nodes to move them around. Hover to see details.
- SHIFT-click on AST nodes to go to the source/AST view.



AST ID	214748365375
type	KW_REPEAT
text	REPEAT
line : col	73 : 1
ON STOP	UNDO, NEXT
ON ERROR	UNDO, NEXT
ON ENDKEY	UNDO, NEXT

Call Graph Reports

- Ambiguous Call Sites
  - Caused by indirect calling conventions and runtime determination of call targets.
  - To complete the graph, you provide hints to tell the call graph analyzer how to traverse these.
  - Iterative process to define hints, run the analyzer, review the latest ambiguous listing, provide hints… until there are no further ambigous locations.
- Dead Code
  - In our experience, 25% to 40% of every non-trivial application of a certain age (10+ years) is dead code.
  - Once your graph is complete, this is an accurate list of the code you can delete.
  - Delete the code and put it through testing to confirm that the graph hints were correct.
- Missing Call Targets
- External Dependencies

FWD analytics

How to Get Started

How to Get Started

- Download and install FWD.
- Download one of the sample template projects (there is one for ChUI and one for GUI).
- Follow the “Getting Started” instructions to get the template project installed and configured for your application code, including placing your code and schemata into the template project.
- Run the `ant report_server` target.
- Start the report server.
- Access the server at port 9443 via a browser.
- Full details of this process and all documentation: **https://proj.goldencode.com/projects/p2j/wiki/Code_Analytics**

FWD analytics

Planned Improvements

Planned Improvements

- Add more built-in call-graph analysis and reports. One example: identifying all locations that use a specific NEW SHARED variable (and the inverse).
- Move our existing transformation rules that calculate important properties to an early enough location that it can be integrated into reporting. This would include things like buffer scoping, frame scoping, index selection, transaction/block properties and more.
- Duplicate Code Identification. We can identify arbitrary code matches across the entire application using a bottom-up fingerprinting approach for each unique sub-tree in the application. By using fuzzy logic, we can match code that is the same whether it was cut and pasted or just independently coded the same way. Using these fingerprints we can turn duplicated code into common code.
- Improved TRPL syntax and structure, source level debugging.

FWD

Find Us On the Web!

- www.beyondabl.com
- facebook.com/beyondabl
- twitter.com/beyondabl
- plus.google.com/+beyondabl
- linkedin.com/company/fwd-project
- youtube.com/channel/UCk3pga7EKxAQVOV_CiYOR7g