Program Database Toolkit (PDT)

Developers of high-performance scientific applications and frameworks expect tools to provide complete analysis capabilities for the advanced constructs of object-oriented languages, such as templates and namespaces. We developed an analysis infrastructure that provides this support, the

Program Database Toolkit

Based on the latest EDG Front End, the Program Database Toolkit consists of the following components: the C/C++ IL Analyzer and DUCTAPE.

EDG Front End

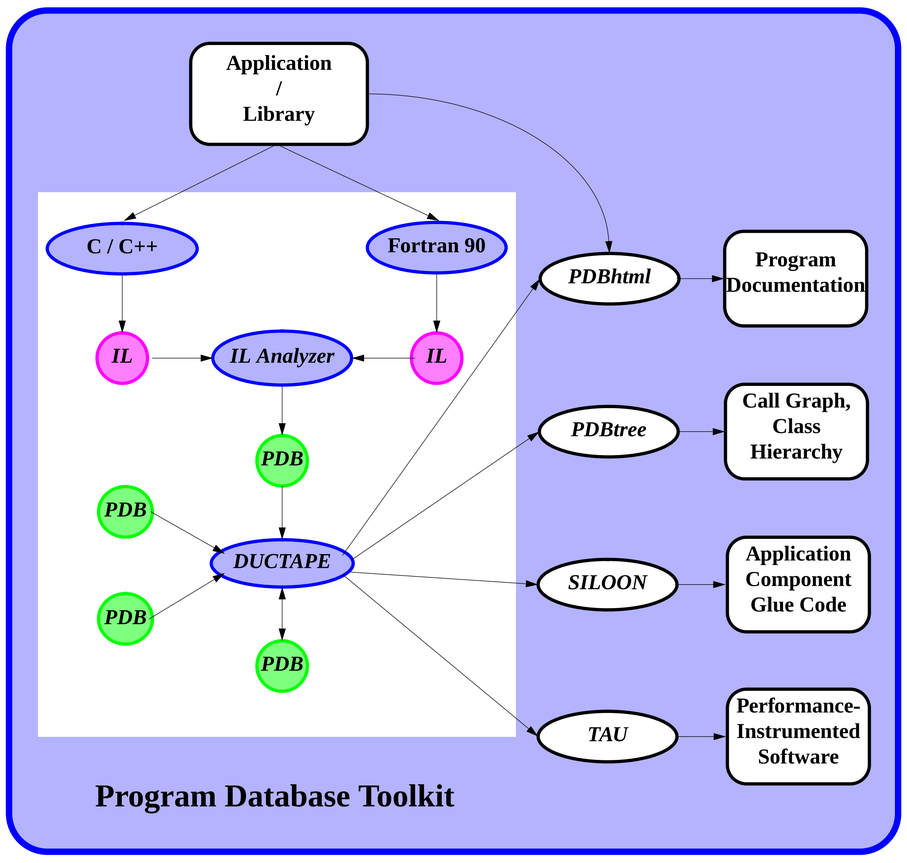
The Edison Design Group (EDG) Front End is a parser that is nearly up-to-date with the C++ standard. The Front End parses a source file, and creates an intermediate-language (IL) tree.

IL Analyzer

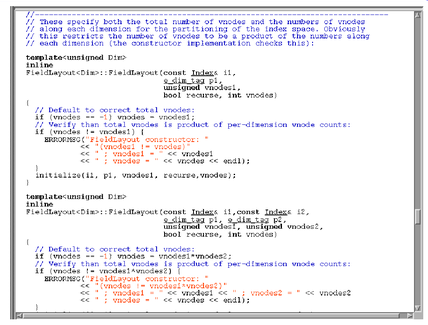
The IL Analyzer processes the intermediate language tree, and creates another file. This file contains the high-level "interface" of the original source. It consists of item descriptions that characterize functions and classes, including template instantiations, as well as other types, source files, namespaces, templates, and macros. This file is in "program database" (PDB) format, and can be easily and efficiently read by a programming or a scripting language.

DUCTAPE

DUCTAPE (C++ program Database Utilities and Conversion Tools APplication Environment) provides a C++ library that enables applications to access PDB files.



PDBhtml Output



The Program Database Toolkit is used in the development of applications. PDT enables static analysis, generation of documentation and "glue" code, and source-to-source translation.

Static Analysis and Documentation Generation

Four DUCTAPE applications have been developed:
pdbconv converts PDB files to a more readable format,
pdbmerge merges PDB files from separate compilations,
pdbtree prints file inclusion, class hierarchy, and call graph trees, and
pdbhtml "htmlizes" C++ source.

Code Generation

For SILOON (Scripting Interface Languages for Object-Oriented Numerics), PDT assists in generating "glue" code. This automates access to C++ routines from programs written in scripting languages.

Source-to-Source Translation

The TAU (Tuning and Analysis Utilities) Instrumentor uses PDT to instrument C++ programs for TAU profiling and tracing instrumentation. It traverses the PDB list of functions and templates, and inserts the TAU profiling macros in the source. The programs are then recompiled and linked with the TAU library to generate profile data files during execution.

Source File

```
void sort_5elements(int *arr) {
    quicksort(arr, 0, 4);
}

int select_kth_largest(int k, int *S, int n) {
    ...
    n_M = ceil(n, 5);
    for (i=0; i< n_M; i++) {
    sort_5elements(&S[5*i]);
    }

    S1 = (int *) malloc ((3*n/4) * sizeof(int));
    S3 = (int *) malloc ((3*n/4) * sizeof(int));

    for (i=0; i< n_M; i++)
    M[i] = S[middle(i)];
    m = select_kth_largest(ceil(n_M,2), M, n_M);

    ...
}

void interchange(int *a, int *b) {
    ...
    temp = *a;  *a = *b;  *b = temp;
}

void quicksort(int *arr, int m, int n) {
    ...
    if (m < n) {
        i = m;  j = n+1;  k = arr[m];
        do {
            do { i = i + 1; } while (arr[i] > k);
            do { j = j - 1; } while (arr[j] < k);
            if (i < j) interchange(&arr[i], &arr[j]);
        } while (i < j);
        interchange(&arr[m], &arr[j]);
        quicksort(arr, m, j - 1);
        quicksort(arr, j + 1, n);
    }
}
```

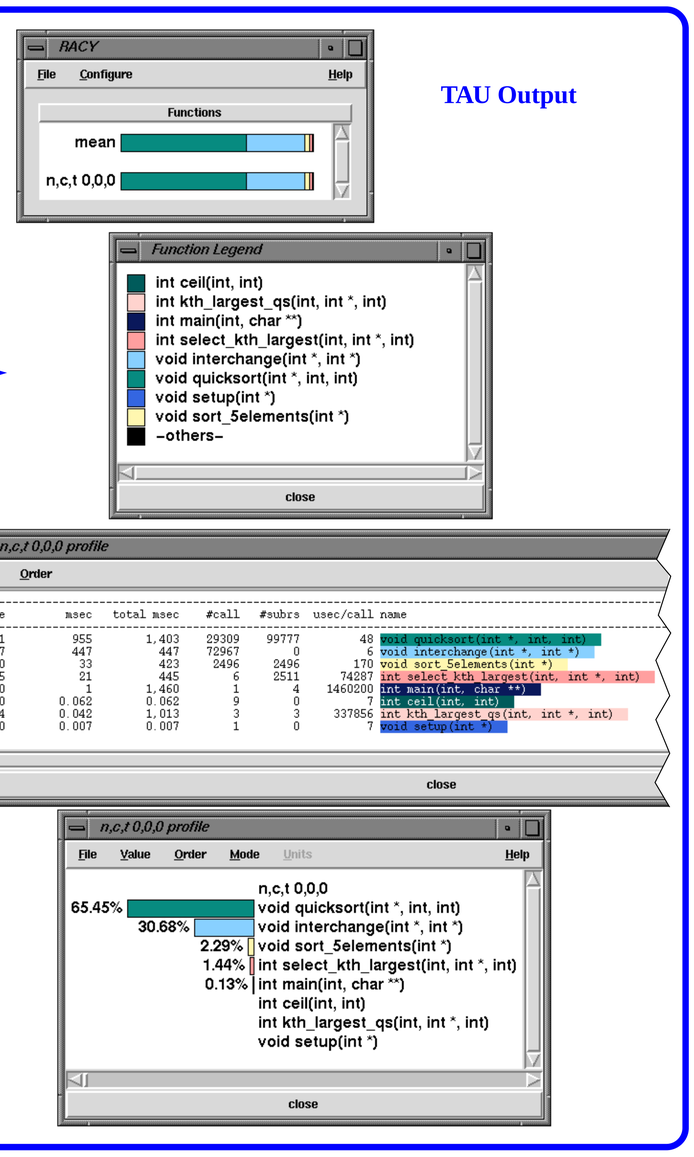
PDB File

```
<PDB 1.0>

so#15 /usr/include/stdlib.h

so#17 decl.h

so#18 klargest.cpp
sinc so#15
sinc so#17

ro#4 sort_5elements
rloc so#18 237 6
rsig ty#551
rlink C++
rstore NA
rvirt no
rcall ro#12 no so#18 239 3
rpos so#18 237 1 so#18 237 29 so#18 238 1
   so#18 241 1

ro#8 select_kth_largest
rloc so#18 31 5
rsig ty#546
rlink C++
rstore NA
rvirt no
rcall ro#2 no so#18 57 12
rcall ro#3 no so#18 70 9
rcall ro#4 no so#18 74 5
rcall ro#5 no so#18 81 16
rcall ro#5 no so#18 82 16
rcall ro#6 no so#18 85 5
rcall ro#7 no so#18 86 5
rcall ro#3 no so#18 90 26
rcall ro#8 no so#18 90 7
rcall ro#9 no so#18 108 5
rcall ro#8 no so#18 109 12
rcall ro#9 no so#18 110 5
rcall ro#9 no so#18 119 7
rcall ro#9 no so#18 120 7
rcall ro#9 no so#18 127 7
rcall ro#8 no so#18 128 14
rcall ro#9 no so#18 129 7
rpos so#18 31 1 so#18 31 44 so#18 32 1
   so#18 133 1

ro#10 interchange
rloc so#18 142 6
rsig ty#547
rlink C++
rstore NA
rvirt no
rpos so#18 142 1 so#18 142 32 so#18 143 1
   so#18 149 1

ro#12 quicksort
rloc so#18 189 6
rsig ty#549
rlink C++
rstore NA
rvirt no
rcall ro#10 no so#18 223 9
rcall ro#10 no so#18 225 5
rcall ro#12 no so#18 226 5
rcall ro#12 no so#18 227 5
rpos so#18 189 1 so#18 189 38 so#18 190 1
   so#18 229 1

ty#546 int (int, int *, int)
ykind func
yrett ty#1
yargt ty#1
yargt ty#3
yargt ty#1

ty#547 void (int *, int *)
ykind func
yrett ty#7
yargt ty#3
yargt ty#3

ty#551 void (int *)
ykind func
yrett ty#7
yargt ty#3

ty#549 void (int *, int, int)
ykind func
yrett ty#7
yargt ty#3
yargt ty#1
```

Instrumented Source File

```
void sort_5elements(int *arr) {
    TAU_PROFILE ("void sort_5elements(int *)",
        " ", TAU_USER);
    quicksort(arr, 0, 4);
}

int select_kth_largest(int k, int *S, int n) {
    TAU_PROFILE("int select_kth_largest(int, int *,
int)",
        " ", TAU_USER);
    ...
    n_M = ceil(n, 5);
    for (i=0; i< n_M; i++) {
    sort_5elements(&S[5*i]);
    }

    S1 = (int *) malloc ((3*n/4) * sizeof(int));
    S3 = (int *) malloc ((3*n/4) * sizeof(int));

    for (i=0; i< n_M; i++)
    M[i] = S[middle(i)];
    m = select_kth_largest(ceil(n_M,2), M, n_M);
    ...
}

void interchange(int *a, int *b) {
    TAU_PROFILE ("void interchange(int *, int *)",
        " ", TAU_USER);
    ...
    temp = *a;  *a = *b;  *b = temp;
}

void quicksort(int *arr, int m, int n) {
    TAU_PROFILE("void quicksort(int *, int, int)",
        " ", TAU_USER);
    ...
    if (m < n) {
        i = m;  j = n+1;  k = arr[m];
        do {
            do { i = i + 1; } while (arr[i] > k);
            do { j = j - 1; } while (arr[j] < k);
            if (i < j) interchange(&arr[i], &arr[j]);
        } while (i < j);
        interchange(&arr[m], &arr[j]);
        quicksort(arr, m, j - 1);
        quicksort(arr, j + 1, n);
    }
}
```

TAU Output



ROUTINE Item

ro#[routineID]	<name of routine>	
rloc	<fileID> <line> <column>	# location
[		
rclass	<classID>	# parent class
racs	<pub | prot | priv>	# access
OR		
rnspace	<namespaceID>	# parent namespace
]		
rsig	<typeID>	# signature
rlink	<no | internal | C++ | C>	# linkage
rstore	<ext | stat | auto | NA | asm>	# storage class
rstatic	<boolean>	# is static function?
rkind	<ctor | dtor | conv | op>	# special kind
rvirt	<no | virt | pure>	# virtuality
rcrvo	<boolean>	# is covariant return virtual # override?
rinline	<boolean>	# is inline?
rcgen	<boolean>	# is compiler generated?
rexpl	<boolean>	# is explicit ctor?
rtempl	<templateID or boolean>	# ID if template instance; # boolean if specialized
rspecl	<boolean>	# is specialized?
rcall [...]	<routineID> <no | virt> <fileID> <line> <column>	# callees
rpos	<start of return type> <last token before "{"> <"{"> <"}">	

Program Database (PDB) Format

all ITEMs	source position, parent class, access
HEADER	<PDB 1.0>
FILES	header files included by source file
ROUTINES	signature, characteristics, functions called, template from which instantiated
TYPES	various characteristics, depending on type
CLASSES	characteristics, direct parent classes, friend classes and functions, member functions, information on other members, template from which instantiated
NAMESPACES	members of namespace or alias
TEMPLATES	text of template
MACROS	text of macro

Kathleen Lindlan, Allen D. Malony,
Jan Cuny, Sameer Shende
Department of Computer and Information Science
University of Oregon, Eugene, OR 97403
{klindlan, cuny, malony, sameer}@cs.uoregon.edu

Bernd Mohr
Research Centre Juelich, ZAM
D-52425 Juelich, Germany
b.mohr@fz-juelich.de

Peter Beckman
Advanced Computing Laboratory, LANL
Los Alamos, NM 87545
beckman@acl.lanl.gov

Special Thanks to Rod Oldehoeft
Advanced Computing Laboratory, LANL
Los Alamos, NM 87545
rro@acl.lanl.gov

Powered by DOE 2000 & ASCI

http://www.acl.lanl.gov/pdtoolkit/