Accomplishments and Challenges of KDevelop Team

KDevelop
Published under GNU Licences
KDE DEVELOPMENT ENVIRONMENT
We understand your C++ since 2002
class Foo {
    public:
    /**There's smth to do*/
    void doSmth();
    /**There's another way*/
    void doSmthElse();
};

class FooP {
    public:
    Foo* operator->();
    Foo* data();
};

int foo()
{
    FooP p;
    p.
    Foo* data() [Container: FooP
                 Kind: Function
                 Access: public
                 File: /home/gremlin/projects/oss/kde_svn/trunk/kdevelop/plugins/filemanager/test.h
                 Line: 13 Column: 7
    Foo* operator->()
}
```cpp
class Foo {
public:
    /** There's smth to do*/
    void doSmth();
    /** There's another way*/
    void doSmthElse();
};

class FooP {
public:
    Foo* operator->();
    Foo* data();
};

int foo() {
    FooP p;
    p->doSmth();
}
```
We understand your buildsystem

Auto Hell Tools?
qmake?
cmake?
make?
whatever else?
We do support KDE4 development

How?

http://www.kdedevelopers.org/node/2286

Thanks to Andras Mantia
cmake -G KDevelop3

ok, ok

cmake -DCMAKE_INSTALL_PREFIX=path_to_kde4_install_dir
-DCMAKE_BUILD_TYPE=debugfull path_to_source_dir
-DKDE4_BUILD_TESTS -G KDevelop3

Thanks to Alex Neundorf for CMake generator
We do support KDE4 development.
Some environment vars to set

QTDIR=<your qt dir>
KDEDIR=<your kde4 dir>
KDE4_DIR=<your kde4 dir>
PATH=$QTDIR/bin:$KDEDIR/bin:$PATH
We do support KDE4 development

Alexander Dymo <adymo@kdevelop.org>, aKademy 2006

Some more environment vars to set

KDEHOME=path_to_local_KDE4 folder (/home/user/.kde4)
KDETMP=path_to_KDE4 temp dir (/tmp/user-kde4)
KDEVARTMP=similar to the above in /var (/var/tmp/user-kde4)

Don't forget about
eval `dbus-launch --auto-syntax`
More cool stuff: Ruby Debugger
not as easy
not as complete
not as slick
Cleaner architecture
Powerful platform
Speaking the languages natively
Native CMake support
Teamwork
KDevelop 4: C++

Lexer → Parser → Binder → DU-Chain Builder

Token Stream → AST → Code Model → DU Chain

Syntax Checker → Class Tree → Code Completion → Smart Highlighting → Refactoring?
namespace Blah {
    class Foo {
        int m_test;
        static int s_test2;
        void test(int input);
        void test5() {}
    }
    int nsTest;
}

int Blah::Foo::s_test2 = 0;

int test3() {
    // Unqualified + before using statement
    nsTest = 2;
    return Blah::nsTest;
}

using namespace Blah;

int test2() {
    // Success - using statement applies
    nsTest = 4;
    return Blah::nsTest;
}

class Foo2 {}

int Foo::test(int input) {
    // Use before definition - error
    result = 3;
    int result = m_test;
Not so crazy to implement all these manually
Crazy enough to use a tool
-- test.g

%token ID ("identifier")
identifier + identifier
-> expression
ID
-> identifier

#kdev-pg –output=test test.g

test_ast.h
test_default_visitor.cpp
test_default_visitor.h
test_parser.cpp
test_parser.h
test_visitor.cpp
test_visitor.h
struct ast_node {
    enum ast_node_kind_enum {
        Kind_expression = 1000,
        Kind_identifier = 1001,
        AST_NODE_KIND_COUNT
    };
    int kind;
    std::size_t start_token;
    std::size_t end_token;
};

struct expression_ast: public ast_node {
    enum { KIND = Kind_expression };
};

struct identifier_ast: public ast_node {
    enum { KIND = Kind_identifier };
};

class visitor {
    public:
        virtual void visit_node(ast_node *node) {}
        virtual void visit_expression(expression_ast *) {}
        virtual void visit_identifier(identifier_ast *) {}
};
Bringing it to the next level
How to interoperate with CMake?
Teamwork Mode

Client/Server

File Collaboration

Conversation

Patch Management
The next step is WORLD DOMINATION
The next step is WORLD DOMINATION
The next step is WORLD DOMINATION

KDevelop 3.4 and 4.0 commit rate
Thanks and any questions?