ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙНАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК
КАФЕДРА МАТЕМАТИЧЕСКОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

РАЗРАБОТКА ПРОГРАММНОГО МОДУЛЯ ДЛЯ АНАЛИЗА УСТОЙЧИВОСТИ И ДОЛГОВЕЧНОСТИ ИНЖЕНЕРНЫХ КОНСТРУКЦИЙ

Выпускная квалификационная работа
обучающегося по направлению подготовки 02.03.02
Фундаментальная информатика и информационные технологии
заочной формы обучения,
группы 07001250
Бережного Валерия Владимировича

Научный руководитель
к.т.н., доцент
Муромцев В.В.

БЕЛГОРОД 2017

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время в деятельность инженерных и проектных организаций быстро проникает компьютеризация, поднимающая проектную работу на качественно новый уровень, при котором резко повышаются темпы и качество проектирования, более обоснованно решаются многие сложные инженерные задачи, которые раньше рассматривались лишь упрощенно. Во многом это происходит благодаря использованию эффективных специализированных программ, которые могут быть как самостоятельными, так и в виде расширений к общетехническим программам. Деятельность по созданию программных продуктов и технических средств для автоматизации проектных работ имеет общее название - САПР.

САПР (англ. CAD, Computer-Aided Design) - программа, необходимая для проектирования (разработки) объектов производства (или строительства), а также оформления конструкторской и/или технологической документации.

Компоненты систем САПР обычно группируются в три блока: CAD (Computer Aided Designed), CAM (Computer Aided Manufacturing), CAE (Computer Aided Engineering). Модули блока CAD предназначены в основном для выполнения графических работ, модули CAM — для решения задач технологической подготовки производства, модули CAE — для инженерных расчетов, анализа и проверки проектных решений.

Крупнейшим в мире поставщиком программного обеспечения для промышленного и гражданского строительства, машиностроения, рынка средств информации является компания Autodesk, Inc. С 1982 года компанией Autodesk был разработан широкий спектр решений для инженеров, конструкторов, архитекторов, позволяющих им создавать

цифровые модели. Технологии Autodesk используются для моделирования, визуализации и анализа поведения разрабатываемых конструкций на ранних стадиях проектирования и позволяют не просто увидеть модель на экране, но и испытать её.

Строительство всегда развивалось в ногу с научно-техническим прогрессом, но совершенствование программных средств далеко опережает квалификацию специалистов, призванных использовать их в своей работе. Часто наблюдается картина, когда современные и многофункциональные комплексы простаивают или используются незначительно из-за низкого уровня подготовки пользователей [1,12].

Другая проблема заключается в использовании пиратских копий программных продуктов. В этом случае пользователи лишают себя любой технической поддержки со стороны разработчиков: нет регулярного обновления программ, технической документации и квалифицированного обучения. Покупая нелицензионное программное обеспечение, пользователи лишают финансовой поддержки разработчиков, что в свою очередь тормозит развитие программ.

Указанные выше проблемы развития САПР могут быть причиной неправильного подбора программных средств автоматизации.

Если после проектирования здания необходимо рассчитать смету, передать данные в бухгалтерскую программу или произвести расчет каких-либо конструкций, программы должны быть взаимосвязаны. Такая интеграция позволит автоматизировать в едином информационном пространстве все стадии строительства и проектирования.

Цель данной работы является разработка программного модуля для Cad - систем, позврляющего осуществлять расчет устойчивости и долговечности инженерных конструкций по методике АО «Проектмашприбор»:

1.Обзор методов для анализа инженерных конструкций

2.Изучение и обоснование выбора программного обеспечения для анализа инженерных конструкций

3.Определение требований к создаваемому программному модулю

4.Разработка архитектура программного модуля

5.Внедрение в программу AutoCad разработанного модуля

Дипломная работа состоит из введения, трех глав, заключения и списка использованных источников. Объем работы – 46 страниц, в работе содержится 24 рисунка и приложение.

В первой главе рассматривается механизм расчета инженерных конструкций, а также обзор компьютерах средств для их моделирования.

Во второй главе произведено проектирование архитектуры модуля, и продемонстрированы основные этапы разработка программного модуля.

В третьей главе описаны процессы тестирования и отладки разработанного расширения.

ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ И ВЫЧИСЛИТЕЛЬНЫЕ АСПЕКТЫ ЗАДАЧ АНАЛИЗА ИНЖЕНЕРНЫХ КОНСТРУКЦИЙ

В данной главе рассмотрена сфера деятельности предприятия АО «Проектмашприбор» и механизмы расчета инженерных конструкций, а также проводится обзор компьютерах средств для проектирования. В заключении дается постановка задачи.

1.1 Направление работы предприятия АО «Проектмашприбор»

Институт по проектированию предприятий машиностроения и приборостроения (АО «Проектмашприбор») основан в 1932 году и изначально был предназначен для проектирования предприятий по производству различных видов боеприпасов.

В предвоенные годы институт спроектировал свыше 20 объектов по выпуску боеприпасов разных типов.

В условиях Великой Отечественной войны коллективом института была разработана проектная документация более чем на 40 перебазированных на восток предприятий, сделав весомый вклад в организацию массового выпуска боеприпасов.

В послевоенные годы проектирование было направлено на восстановление разрушенных войной зданий и сооружений промышленных предприятий, а также на проектирование крупных промышленных комплексов по выпуску продукции мирного времени.

По проектам предприятия реализованы многие технологии изготовления стрелково-пушечного и ракетного вооружения, автоматизированных комплексов патронного производства, производства

мотоциклов и автомобилей, специальных станков, оптических и оптико-электронных приборов.

В трудные годы конверсии «оборонки» и перехода к рыночной экономике коллектив института сумел сохранить основной кадровый потенциал и закрепить свои позиции на рынке проектных услуг.

Институт работает в сфере комплексного проектирования разделов проектной и рабочей документации для строительства, реконструкции и технического перевооружения промышленных объектов различных отраслей народного хозяйства, а также гражданских объектов и социально-бытового назначения.

Среди заказчиков предприятия следует отметить:

- АО «НПК «Конструкторское бюро машиностроения» г. Коломна;
- АО «Центральный научно-исследовательский институт автоматики и гидравлики» г. Москва;
- АО «Уральский оптико-механический завод им. Э.С.Яламова» г. Екатеринбург;
- АО « Конструкторское бюро точного машиностроения им. А.Э. Нудельмана» г. Москва.

Институт работает в сфере проектирования для строительства, реконструкции и технического перевооружения промышленных объектов, а также жилищно-гражданских и объектов социально-бытового назначения. Комплексное проектирование разделов проекта и рабочей документации осуществляется по следующим направлениям:

- управление производством предприятия и организация условий и охраны труда рабочих и служащих;
- мероприятия по пожарной безопасности и охранные;
- разработка стройэнергокомплекса;
- охрана окружающей среды;
- технологическое проектирование;

•эффективность инвестиций;

•автоматизация управления инженерными системами;

•мероприятия по гражданской обороне и предупреждению чрезвычайных ситуаций;

•сметная документация.

А также осуществляет:

•функции генерального проектировщика;

•расчет санитарно-защитных зон;

•авторский надзор за строительством;

•обследования состояния строительных конструкций;

•инженерные изыскания.

1.2 Задача анализа инженерных конструкций на прочность

Расчеты на прочность. Различают два вида расчетов: проектный/проектировочный и проверочный.

Проектирование детали можно вести в следующей последовательности:

1.Составление расчетной схемы в упрощенном виде.

2.Определение нагрузки, действующей на деталь в процессе ее функционирования.

3.Выбор материала и назначение допускаемого напряжения.

4.Определение размеров детали и ее характер с другими из условий прочности, жесткости, долговечности.

5.Выполнение рабочего чертежа детали с указанием всех сведений, необходимых для ее изготовления.

При проверочном расчете по известным нагрузкам и известных размерах детали определяют уровень напряжений, которые сравнивают с допускаемыми.

1.3 Проектирование стальной балки в нескольких вариантах поперечного сечения с оценкой их долговечности

Для стальной балки с учетом характеристик материалов расчет будет выглядеть следующим образом:

Построение эпюр Q_y и M_x. Опорные реакции

$\sum Y_i = 0$, $R_A - qa - qa = 0$, $R_A = 2qa$,

$\sum m_A = O$, $M_A - 0.5\,qa^2 - 2qa^2 + 3qa^2 = 0$, $M_A = 0.5qa^2$.

Эпюра $\boldsymbol{Q_y}$**.** Она строится по формуле $\boldsymbol{Q = Qo} \pm qz$. В данном случае следует взять знак "минус", так как погонная нагрузка направлена вниз. Поперечная сила постоянна на участках *BC, CD* ($q = 0$) и изображается наклонной прямой на участке *AB* (q = const). Вычисляем значения $\boldsymbol{Q_y}$ в характерных точках [2,6,8].

$Qa = 2qa$, $Q_{bc} = 2q - qa = qa$ $Q_{cd} = qa - qa = 0$.

Эпюра M_x. Она строится по формуле $M_x = M_o + Q_o z - 0{,}5qz^2$. Изгибающий момент изменяется по квадратичному закону на участке *AE* (q = const) и по линейному закону — на всех остальных участках, где $q = 0$. По значениям момента в характерных точках

$M_A = -0.5qa^2$, $M_B = -0.5qa^2 - 1.5qa^2 = -2qa^2$

$M_{CD} = -2qa^2 - qa^2 = -3qa^2$, $M_d = -3qa^2 + 3qa^2 = 0$

Расчетный изгибающий момент равен $M_{рас} = |M_{CD}| = 3qa^2 = 3 \cdot 12 \cdot 1.5^2 = 81$ кН•м.

•*Подбор сечений.* Из условия прочности по нормальным напряжениям $\sigma_{max} = \boldsymbol{M_{рас}/W_x} < [\sigma]$ определяем требуемый момент сопротивления поперечного сечения

$W_x \geq M_{рас} / [\sigma] = 81 * 10^3 / 160 * 10^6 = 506 см^3$, по которому подбираем конкретные сечения.

Круг: $W_x = \pi d^3/32$, $d = \sqrt[3]{32W_x/\pi} = \sqrt[3]{32506/3.14} = 17.3$ см.

Принимаем по ГОСТ 6636-86 нормализованное значение

d_o= 180 мм, тогда $A_1 = \pi d^2/4 = \pi \cdot 18^2/4 = 254$ см^2.

Прямоугольник (h / b = 2): $W_x = b(2b)^2/b = 2b^3/3$,

$b \geq \sqrt[3]{3W_x/2} = \sqrt[3]{3 * 506/2} = 9.1$ см.

Ближайшее меньшее стандартное значение равно b_0 = 90 мм. При этом балка будет работать с перенапряжением, равным,

$\delta\sigma = [(b^3 - b_0^3)/ b_0^3] * 100\% = [(9.1^3 - 9^3)/9^3] \cdot 100\% = 2.2\%$,

что допустимо. $A_2 = 2b_0^2 = 162$ см^2.

Двутавр. По ГОСТ 8239-89 выбираем двутавр №30 а, для которого W_{X0} =518 см^3, A_3=49.5см^2.

Два швеллера. По ГОСТ 8240-89 выбираем два швеллера №24а, для которых W_{X0}=2*265= 530 см^3,A_4 = 2*32.9 = 65.8 см^2.

Неравнобокие уголки. Они находятся подбором, так как в сортаменте не даны значения момента сопротивления. Используя формулу $W_X = 2*I_x/y_{наиб} = 2I_X/ (B — y_0)$, сделав несколько попыток, выбираем два уголка 250х160х18, для которыхW_{Xo} =542 см^3,

A_5 = 2*71.1 = 142.2 см^2.

•Оценка экономичности подобранных сечений. Масса балки определяется как произведение плотности материала на ее объем *m = pA1,* т.е. расход материала при прочих равных условиях зависит только от площади поперечного сечения *A.* Сравнивая массы балок. [3,6]

$m_1 = m_2 = m_3 = m_4 = m_5 = A_1 : A_2 : A_3 : A_4 : A_5 = 1 : 0{,}64 : 0{,}19 : 0{,}26 : 0{,}56$,

Делаем вывод, что самым неэкономичным является круглое сечение. При замене круга другими формами (прямоугольник, двутавр, два швеллера, два уголка) достигается экономия, равная соответственно 36%, 81%, 74% и 44%.

•Исследование напряжений в опорном сечении для балки двутаврового профиля №30а, параметры которой по ГОСТ 8239-89[17] равны:

h = 30 см. *b* =14.5 см; *d* =0,65 см; t =1.07 см; I_x= *7780* см^4; S_x = 292 см^3.

Эпюра σ. Нормальные напряжения в поперечном сечении изменяются по линейному закону $\sigma_z = (M_x / I_x)y$. Вычисляем напряжения в крайних точках .

$\sigma_{max} = -\sigma_{min} = M_X/W_X = 81*10^3 / 518*10^{-6} = 156$МПа.

Эпюра τ. Она строится по формуле Журавского

$T = QS_x^{отс} / (bI_x)$

Находим значения τ в 4 характерных точках по высоте сечения (необходимые вычисления представлены в таблице) и строим эпюру касательных напряжений.

Определение главных напряжений в точке *К* ($y_к$ = 0,4h):

- напряжения в поперечном сечении

$\sigma_к = (M_c/I_x)y_к = (-81:10^3 / 7780\text{-}10^{-8})*(-0.4*30) = 125$МПа,

$\tau_к = Q_c S_X^{отс} /(b_k I_x) = 18*10^3*245\text{-}10^{-6}/(0{,}65*10^{-2}*7780*10^{-8}) = 8.7$ МПа;

- величины главных напряжений

$\sigma_{1,3} = 0{,}5(\sigma_к \pm \sqrt[3]{3 * 506/2}) = 0{,}5(125 \pm \sqrt{125^2 + 4 * 8.7^2})$;

$\sigma_1 = 125.6$ МПа $\sigma_3 = -0.6$ МПа;

- ориентация главных площадок

$tg\alpha_1 = (\sigma_1 - \sigma_k) / \tau_к = (126.5 + 125) / 8.7 = 0.17241$; $\alpha_1 = 10°52'$.

Экстремальные касательные напряжения равны по величине

$\tau_{min;max} = \pm 0{,}5(\sigma_1 - \sigma_3) = \pm 0{,}5(125.6+0.6) = \pm 63.1$ МПа

и действуют на площадках, равнонаклоненных к осям 1 и 3.

1.4 Инструментальная компьютерная поддержка методов анализа инженерных конструкций

Существует большое количество пакетов САПР разного уровня. Значительное распространение получили системы, в которых основное внимание сосредоточено на создании "открытых" (допускающих расширение) базовых графических модулей CAD, а модули для выполнения расчетных или технологических задач (соответствующие блокам CAM и CAE) остаются для разработки пользователям или организациям, подготовленным на соответствующем программировании. Дополнительные модули могут использоваться и самостоятельно, без CAD-систем, что находит свое применение в строительном проектировании.

В России широко распространен программный пакет AutoCAD. Разработанный Autodesk более 20 лет назад, он долгое время отвечал самым высоким требованиям проектировщиков. Но на сегодняшний день, обладая богатым инструментарием и возможностями адаптации к требованиям пользователя, рынок предлагает большое количество программ инженерной направленности. Существует также возможность создавать дополнительный расширения, позволяющие адаптироваться к растущим требованиям проектирования.

Рассмотрим программы с подобным функционалом:

ArchiCAD — программное обеспечение компании Graphisoft является на данный момент одной из лучших систем архитектурно-строительного проектирования. ArchiCAD — мощная среда 3D-моделирования для работы с объектами по современным технологиям. При работе в ArchiCAD не просто создаются отдельные чертежи, а разрабатывается полный набор документации по проекту в одном файле [4].

APM Civil Engineering — CAD/CAE система автоматизированного проектирования строительных объектов гражданского и промышленного назначения. Эта система в полном объеме учитывает требования государственных стандартов и строительных норм и правил, относящиеся как к оформлению конструкторской документации, так и к расчетным алгоритмам [5].

Bocad—3D — мощная пространственная CAD-система проектирования стальных и деревянных конструкций. При этом происходит постоянный процесс совершенствования системы в соответствии с пожеланиями конструкторов [6].

BricsCad Pro — обеспечивает совместимость с AutoCAD, а также делает возможным применение сотен программ, разработанных третьими фирмами. Удобные возможности визуальной настройки пользовательского интерфейса, а также поддержка файлов AutoCAD, пользовательских меню, панелей инструментов, сценариев, снимков [7].

CADdy — (немецкая фирма ZIEGLER-Informatics GmbH) по функциональным возможностям занимает промежуточное положение между системами низкого и высокого уровней. Предназначена для решения комплексных интегрированных технологий от стадии проектирования до стадии производства. В настоящее время в состав CADdy входит свыше 80 модулей, охватывающих такие направления, как машиностроение, архитектура, строительство, геодезия, картография и городское планирование [8].

Google SketchUp — простой и удобный инструмент для создания, обработки и презентации трёхмерных моделей. Здания, мебель, интерьер, строительные сооружения и многое другое проектируется за считанные минуты. SketchUP предоставляет возможность создавать многостраничные документы и презентации; раскладывать множество масштабированных

моделей на одной странице; создавать, документировать и делать презентацию проекта, используя один чертёж [9].

IronCAD — это профессиональная система самого последнего поколения. Полнофункциональный инструмент для разработчиков, которые хотят эффективно использовать рабочее время. Система IronCAD дает пользователю мощнейший инструмент для оформления чертежей, избавляет от необходимости экспортировать геометрию в какие-либо другие продукты с потерей ассоциативной связи. По своим возможностям программа является достойным конкурентом таким САПР, как AutoCAD, SolidWorks, T-Flex, КОМПАС 3D [10].

Pro/ENGINEER — является САПР верхнего уровня и охватывает все сферы проектирования, технологической подготовки производства и изготовления изделия [11].

КОМПАС — система автоматизированного проектирования, разработанная российской компанией <АСКОН> с возможностями оформления проектной и конструкторской документации согласно стандартам серии ЕСКД и СПДС. Существует в двух версиях: КОМПАС-График и КОМПАС-3D, соответственно предназначенных для плоского черчения и трёхмерного проектирования [6].

SolidWorks — продукт компании SolidWorks Corporation, система автоматизированного проектирования, инженерного анализа и подготовки производства изделий любой сложности и назначения. Инстрментальная среда, предназначенная для автоматизации проектирования сложных изделий в машиностроении и в других областях промышленности [7].

По перечню указанных выше программ можно видеть, что направление в строительной отрасли, архитектуре и собственно проектированию сооружений, развивается очень динамично.

Из многообразия доступных программ для разработки вычислительного модуля выбран AutoCAD, как наиболее используемая на предприятии программа, с поддержкой сторонних расширений.

1.5 Постановка задачи

В связи с большим количеством расчетов при проектировании уникальных разработок в АО «Проектмашприбор» возникает необходимость в автоматизации части их них, и так как разработка чертежей происходит в программе AutoCAD – написании отдельного модуля для этой цели. В ходе анализа проблемы были выявлены основные требования к разрабатываемому модулю:

1)В модуле должна быть поддержана совместимость с программой AutoCAD.

2)Модуль должен быть расширяемым.

3)Расчеты долговечность конструкции должны быть проведены на основании представленных чертежей и данных об используемых материалах.

4)Пользователь должен получить полный вывод полученных значений в текстовом файле и на экране.

ГЛАВА 2. РАЗРАБОТКА ПРОГРАММНОГО МОДУЛЯ

В данной главе рассмотрены основные программные средства и разработка программного модуля, также взаимодействие набора библиотек ObjectARX и готового проекта.

2.1 Разработка для AutoCAD C# расширений

AutoCAD .NET API позволяет управлять приложением AutoCAD и файлами чертежей на программном уровне с использованием доступных сборок или библиотек. Эти объекты могут быть доступны для множества различных языков программирования и всевозможных сред разработки программного обеспечения [12].

Для работы с Autocad можно использовать библиотеки напрямую из Autocad, либо использовать ObjectARX.

ObjectARX — это большой набор библиотек, предназначенный для разработки приложений для AutoCAD в среде программирования Microsoft Visual C++. Сам AutoCAD разработан с использованием ObjectARX.

Рекомендуется использовать одинаковые версии ObjectARX и AutoCAD Совместимость работы библиотеки одной версии с AutoCAD другой версии не гарантируется [13].

"Чистый" ObjectARX предназначен для работы с AutoCAD с помощью C++, однако часть библиотек представляют собой ни что иное, как обертки для классов ObjectARX для работы через .net.

Существует два способа взаимодействия AutoCAD и C#:

1. Программа реализуется в виде отдельного исполняемого файла с работой с файлами AutoCAD через COM-интерфейсы библиотеки

Autocad.Interpop.Common. Такой способ позволяет получить обычный исполняемый exe-файл, который будет работать с dwg-файлами через COM. Такой вариант ограничен функционально из-за малого числа доступных способов "воздействия" на чертеж и не используется в большинстве случаев при написании модулей.

2. В виде расширения (plugin) AutoCAD. Результатом работы будет являться dll-файл, который подгружается в AutoCAD командой "netload" и определяет новые команды (операции) и/или новое поведение стандартных операций.

Для разработки выбран 2й способ, так как дальнейшие планируемой расширение функционала затронет многие функции AutoCAD.

2.2 Архитектура проекта

Набор библиотек ObjectARX представляет разработчику огромный набор инструментов как для работы чертежами, так и с окнами Autocad.

Основные возможности, предоставляемые ObjectARX:

- создание нового файла чертежа;
- редактирование существующих чертежей, которое включает в себя: редактирование примитивов, блоков, словарей чертежа добавление новых команд;
- изменение интерфейса Autocad (добавление новых кнопок, панелей, закладок);

На рисунке 2.1 можно увидеть схему взаимодействия расширения с программой AutoCAD:

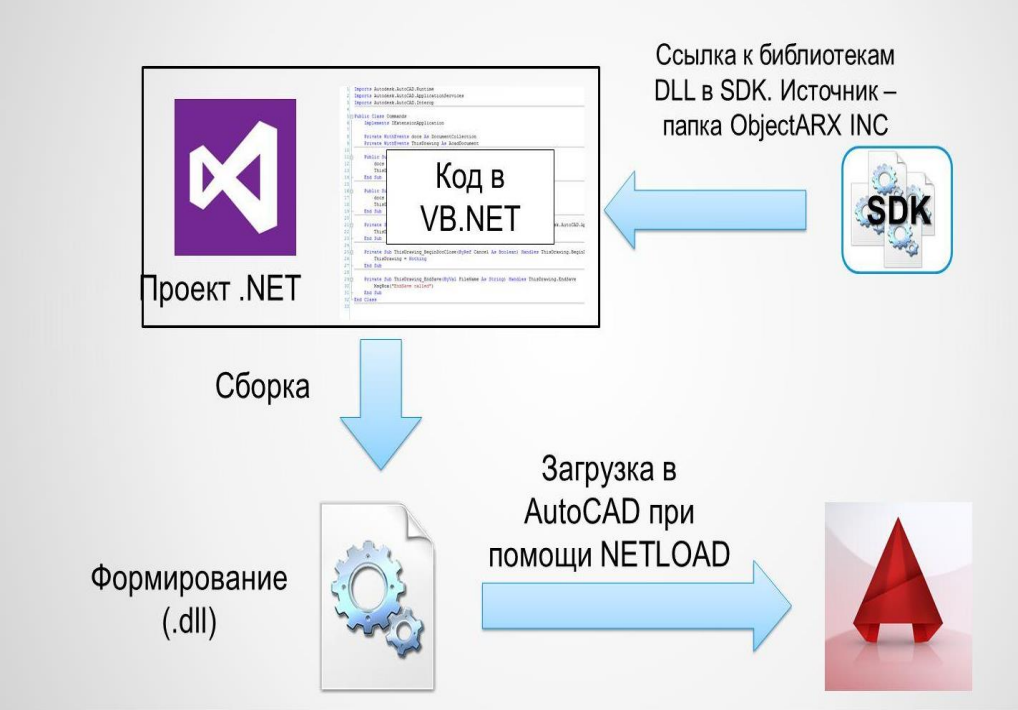


Рис.2.1. Взаимодействие AutoCAD с исполняемым модулем

Исполняемый код преобразуется .dll и загружается в программу функцией NETLOAD [14,15]. Библиотеки и ссылка на них указываются в файлах проекта. Также возможно отладка программы во время запуска.

2.3 Разработка плагина для AutoCAD

2.3.1 Добавление меню

Для проведения расчетов по чертежу необходимо будет добавить в меню панель, отвечающую за расчет и обработку чертежа.

В данном случае пригодятся библиотеки AutoCAD .NET API с именами:

- AcMgd.dll
- AcDbMgd.dll
- AdWindows.dll

Помимо этого, нужно добавим ссылки на три сборки самой .NET: Presentation Core, Presentation Framework и WindowsBase.

В AutoCAD .NET API уже имеются классы, отвечающие за работу с лентой. Они находятся в пространстве имен Autodesk.Windows (его содержит контейнер AdWindows.dll).

Чтобы создать новую вкладку (листинг 1) на ленте, необходимо:

- создать элементы интерфейса;
- сгруппировать эти элементы в контейнеры;
- создать панели, на которых будут размещены эти контейнеры;
- создать вкладку, на которой будут размещены эти панели;
- добавить созданную вкладку на ленту AutoCAD.

Существует 2 способа работы с лентой:

1)Ribbon Runtime API – содержащееся в AdWindows.dll в пространстве имён Autodesk.Windows. Позволяет менять Ленту, но изменения не будут сохраняться в CUIx файл, так что при перезапуске AutoCAD, или переключении рабочего пространства (системной переменной WSCURRENT) все сделанные вами изменения пропадут и вам понадобится всё повторять.

2)CUI API – содержится в AcCui.dll в пространстве имён Autodesk.AutoCAD.Customization. Существует возможность использовать это API для изменения пользовательского интерфейса так же, и эти изменения сохраняются.

Попробуем реализовать оба метода и проверить их на работоспособность в нашем проекте (листинги 1-2)

Листинг 1. Создание панели команд при помощи Ribbon Runtime API

```
      [CommandMethod("TestCommand")]
      public void MyCommand()
      {
            // создаем выпадающий список
            Autodesk.Windows.RibbonCombo comboBox1 = new RibbonCombo();
            comboBox.Id = "_combobox1";

            // создаем кнопку
            Autodesk.Windows.RibbonButton
      button = new Autodesk.Windows.RibbonButton();
            button.Id = "_button";

            // создаем контейнер для элементов
            Autodesk.Windows.RibbonPanelSource
      rbPanelSource = new Autodesk.Windows.RibbonPanelSource();
            rbPanelSource.Title = "Новая панель элементов";
            // добавляем в контейнер элементы управления
            rbPanelSource.Items.Add(comboBox);
            rbPanelSource.Items.Add(new RibbonSeparator());
            rbPanelSource.Items.Add(button);

            // создаем панель
            RibbonPanel rbPanel = new RibbonPanel();
            // добавляем на панель контейнер для элементов
            rbPanel.Source = rbPanelSource;

            // создаем вкладку
            RibbonTab rbTab = new RibbonTab();
            rbTab.Title = "Новая вкладка";
            rbTab.Id = "HabrRibbon";
            // добавляем на вкладку панель
            rbTab.Panels.Add(rbPane);

            // получаем указатель на ленту AutoCAD
            Autodesk.Windows.RibbonControl             rbCtrl             =
ComponentManager.Ribbon;
            // добавляем на ленту вкладку
            rbCtrl.Tabs.Add(rbTab);
            // делаем созданную вкладку активной ("выбранной")
            rbTab.IsActive = true;
      }
```

Собираем проект, запускаем AutoCAD, загружаем с помощью команды NETLOAD наш плагин, выполняем команду TestCommand и получаем результат (рисунок 2.2):

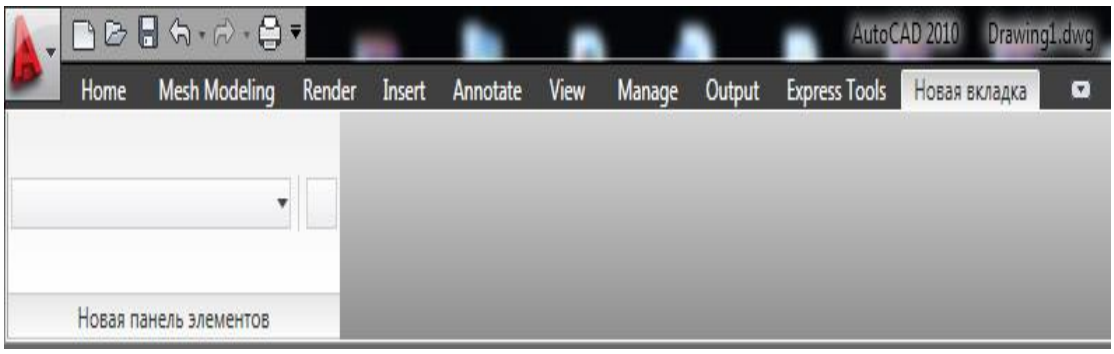


Рис. 2.2. Новая панель элементов

Аналогичным образом создадим новую панель вторым способом:

Листинг 2. Создание панели команд при помощи CUI API

```
[CommandMethod("HideShowHomeTabInCurrentWorkspace")]
static public void HideShowHomeTabInCurrentWorkspace()
{
  string menuName =
    (string)Autodesk.AutoCAD.ApplicationServices.
    Application.GetSystemVariable("MENUNAME");

  string curWorkspace =
    (string)Autodesk.AutoCAD.ApplicationServices.
    Application.GetSystemVariable("WSCURRENT");

  Autodesk.AutoCAD.Customization.CustomizationSection cs =
    new Autodesk.AutoCAD.Customization.CustomizationSection(
      menuName + ".cuix");

  Autodesk.AutoCAD.Customization.WSRibbonRoot rr =
    cs.getWorkspace(curWorkspace).WorkspaceRibbonRoot;

  Autodesk.AutoCAD.Customization.WSRibbonTabSourceReference tab =
    rr.FindTabReference("ACAD", "ID_TabHome3D");

  if (tab != null)
    tab.Show = !tab.Show;

  if (cs.IsModified)
  {
    cs.Save();

    // перезагружаем, чтобы увидеть изменения
    acApp.ReloadAllMenus();
  }
}
```

Для поиска вкладки на ленте используется метод ComponentManager.Ribbon.FindTab(string id). Аргументом функции необходимо указать Id вкладки, заданный при ее создании.

Для привязки обработчика нажатия кнопки нам понадобится свойство CommandHandler класса RibbonButton. В этом свойстве необходимо указать метод, реализующий интерфейс System.Windows.Input.ICommand.

В рамках интерфейса ICommand класс должен реализовать событие CanExecuteChanged, а также функции CanExecute и Execute (листинг 3).

Листинг 3. Код обработчика нажатия кнопки

```
using System;
using Autodesk.AutoCAD.Runtime;
using Autodesk.Windows;
namespace MyAutoCADDll
{
    public class Commands : IExtensionApplication
    {
        // эта функция будет вызываться при выполнении в AutoCAD команды
"TestCommand"
        [CommandMethod("TestCommand")]
        public void MyCommand()
        {
            // создаем выпадающий список
            Autodesk.Windows.RibbonCombo comboBox1 = new RibbonCombo();
            comboBox1.Id = "_combobox1";
            // создаем кнопку
            Autodesk.Windows.RibbonButton button1 = new
Autodesk.Windows.RibbonButton();
            button1.Id = "_button1";
            // привязываем к кнопке обработчик нажатия
            button1.CommandHandler = new CommandHandler_Button1();
        }
        public void Initialize()
        { }
        public void Terminate()
        {}
    }
// обработчик нажатия кнопки
    public class CommandHandler_Button1 : System.Windows.Input.ICommand
    {
        public event EventHandler CanExecuteChanged;

        public bool CanExecute(object param)
        {
            return true;
        }
        public void Execute(object parameter)
        {
            System.Windows.MessageBox.Show("Habr!");
        }
    }
}
```

Событие CanExecuteChanged оповещает пользователей команды о возможном изменении ее доступности для выполнения (возможно ли изменение). Функция CanExecute позволяет узнать, доступна ли команда для выполнения в данный момент времени. А функция Execute — это сами те действия, которые должна выполнять команда, когда ее вызвали.

Теперь после нажатия кнопки на экране появится окно с сообщением (рисунок 2.3).

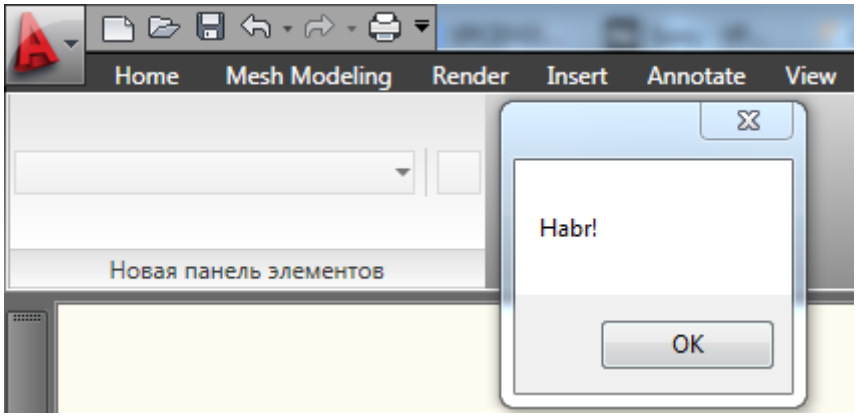


Рис. 2.3.Взаимодействие с выпадающим списком

При выборе элемента списка RibbonCombo генерируется событие CurrentChanged (листинг 4, рис 2.4).

Листинг 4. Обработчик события CurrentChanged

```
        // обработчик выбора нового элемента выпадающего списка
        public       static       void       comboBox1_CurrentChanged(object       o,
RibbonPropertyChangedEventArgs args)
        {
            if (args.NewValue != null)
            {
                System.Windows.MessageBox.Show((args.NewValue                    as
RibbonButton).Text);
            }
        }
```

И в коде события вызывается диалоговое окно, оповещающее о взаимодействии с пользователем и вызове конкретного элемента панели меню.

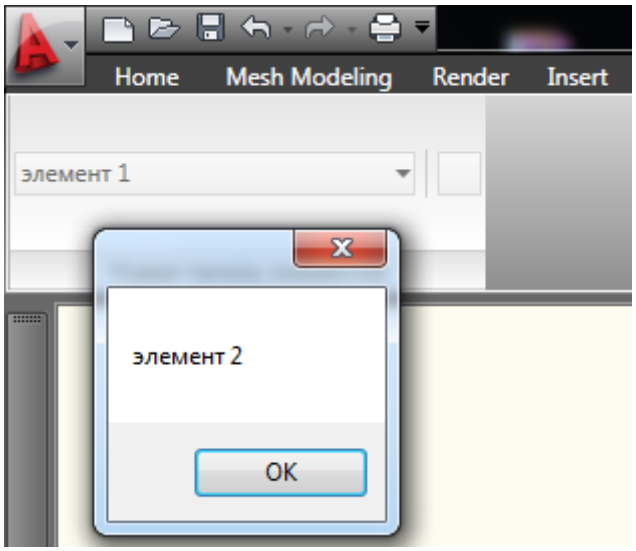


Рис.2.4.Настройка внешнего вида элементов управления

У каждого элемента управления есть свойства, позволяющие изменять его отображение. Например, для выпадающего списка можно задать заголовок и ширину, для кнопки — размер и подпись. Кроме того, можно добавить всплывающие подсказки (рис.2.5).

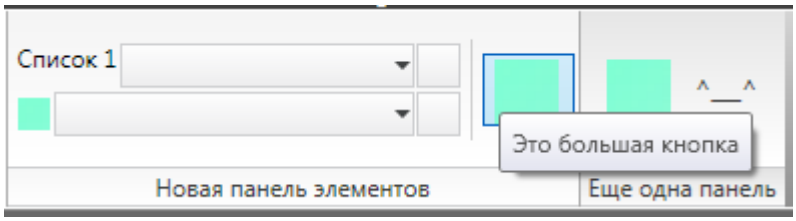


Рис.2.5. Изменение внешнего вида панели инструментов

Визуальное изменение новой панели меню позволит удобнее ориентироваться в ее функциях.

2.3.2 Работа с чертежом

Для расчетов нам необходимо не только ввести данные, но и получить из чертежа размеры сечений/расстояния.

Одним из способов поиска по чертежу является перебор в цикле всех объектов.

Принцип такого подхода выглядит так: открываем пространство модели (ModelSpace) и получаем ссылки на все объекты внутри него. Затем приводим эти объекты к типу Entity и обрабатываем нужные нам свойства (листинг 5).

Листинг 5. Поиск объектов на чертеже

```
      [CommandMethod("Habr_IterateThroughAllObjects_1")]
      public void iterateThroughAllObjects()
      {
          // получаем текущую БД
          Database db1= HostApplicationServices.WorkingDatabase;

          // начинаем транзакцию
          using (Transaction tr = db.TransactionManager.StartTransaction())
          {
              // получаем ссылку на пространство модели (ModelSpace)
              BlockTableRecord                          ms                          =
(BlockTableRecord)tr.GetObject(SymbolUtilityServices.GetBlockModelSpaceId(db),
OpenMode.ForRead);

              // "пробегаем" по всем объектам в пространстве модели
              foreach (ObjId id in ms)
              {
                  // приводим каждый из них к типу Entity
                  Entity entity = (Entity)tr.GetObject(id, OpenMode.ForRead);

                  // выводим в консоль слой (entity.Layer), тип
      (entity.GetType().ToString()) и цвет (entity.Color) каждого объекта

acad.DocumentManager.MdiActiveDocument.Editor.WriteMessage(string.Format("\nLayer:
{0}; Type:{1}; Colors: {2},{3},{4}\n»,
                        entity.Layer,                   entity.GetType().ToString(),
entity.Color.Red.ToString(),                          entity.Color.Green.ToString(),
entity.Color.Blue.ToString()));
              }

              tr.Commit();
          }
      }
```

Результатом будет служить вывод консоли (рисунок 2.6), будет соответствовать заданным в коде параметрам:

- Layer,
- Type,
- Color,

•entity.Layer,

•entity.GetType(),

•entity.Color.Red,

•entity.Color.Green,

•entity.Color.Blue;

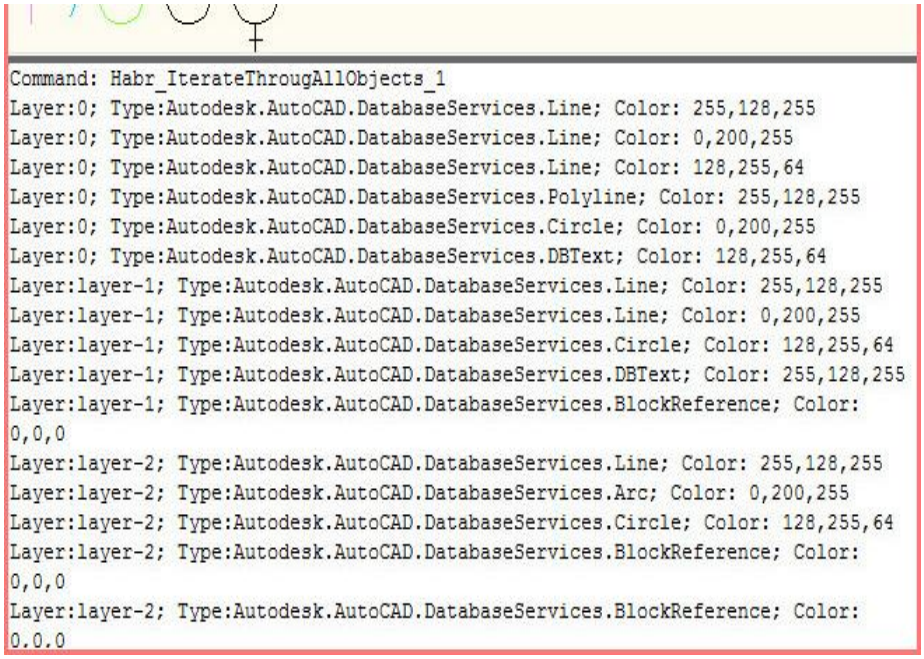


Рисунок 2.6. Вывод данных чертежа

Результат, как можно видеть, полностью совпадает с ожиданиями: у нас 16 объектов, цвета которых различаются.

Получив ссылку на пространство модели, становится возможным делать проход по всем его объектам, приводим их к типу Entity (чтобы можно было просматривать их свойства) и выводим в консоль данные о слое, типе и цвете очередного объекта [16,17].

Если необходимо выделить все окружности, то просмотрим по порядку все объекты чертежа, определим тип каждого объекта; если объект окажется окружностью — выведем информацию о нем в консоль.

```
if (entity.GetType() == typeof(Circle))
// если условие верно - значит, это окружность
```

В Листинге 6 продемонстрирован поиск, основанный на данном подходе:

Листинг 6. Поиск окружностей

```
        [CommandMethod("Habr_FindCircles")]
       public void findCircles()
       {
           // получаем текущую БД
           Database db1= HostApplicationServices.WorkingDatabase;

           // начинаем
           using (Transaction tr = db.TransactionManager.StartTransaction())
           {
               // получаем ссылку на пространство (ModelSpace)
               BlockTableRecord                          ms                          =
(BlockTableRecord)tr.GetObject(SymbolUtilityServices.GetBlockModelSpaceId(db),
OpenMode.ForRead);

               // "пробегаем" по всем объектам в пространстве модели
               foreach (ObjId id in ms)
               {
                   // приводим каждый из них к типу Entity
                   Entity entity = (Entity)tr.GetObject(id, OpenMode.ForRead);

                   // если это окружность - выводим в консоль
                   if (entity.GetType() == typeof(Circle))
                   {
       //выводим необходимые данные в требуемом нам порядке
acad.DocumentManager.MdiActiveDocument.Editor.WriteMessage(string.Format("\nLayer:
{0}; Type:{1}; Colors: {2},{3},{4}\n», entity.Layer, entity.GetType().ToString(),
entity.Color.Red.ToString(),                          entity.Color.Green.ToString(),
entity.Color.Blue.ToString()));
                   }
               }

               tr.Commit();
           }
       }
```

Результат поиска совпадает в необходимым нам и мы видимо описания найденных окружностей (рисунок 2.7):

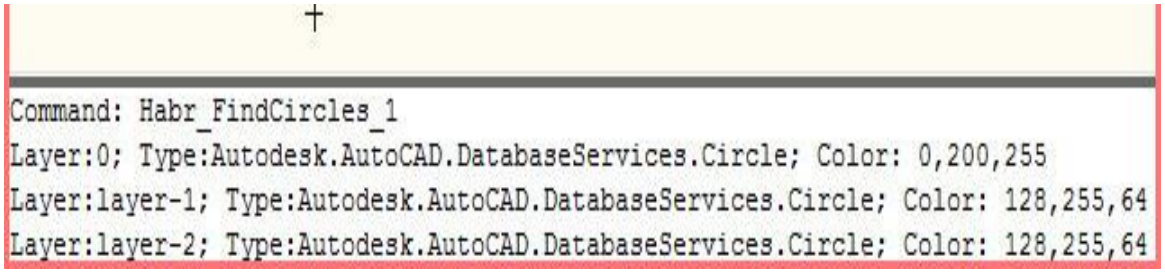


Рис.2.7. Результат поиска окружностей

Аналогично можно попробовать другие типы фигур. Условие для всех линий будет выглядеть следующим образом:

if (entity.GetType() == typeof(Line))

Найти все аннотации (текстовые элементы):

if (entity.GetType() == typeof(DBText))

Все вхождения блоков:

if (entity.GetType() == typeof(BlockReference))

Вместо типа возможен поиск по свойству объекта — например, цвету (листинг 7).

Листинг 7. Поиск объекта чертежа по цвету

```
      // команда для поиска салатовых объектов
      [CommandMethod("Habr_FindLightGreenObjects")]
      public void findLightGreenObjects()
      {
          // получаем текущую БД
          Database db1= HostApplicationServices.WorkingDatabase;

          // начинаем транзакцию
          using (Transaction tr = db.TransactionManager.StartTransaction())
          {
              // получаем ссылку на пространство модели (ModelSpace)
              BlockTableRecord                          ms                          =
(BlockTableRecord)tr.GetObject(SymbolUtilityServices.GetBlockModelSpaceId(db),
OpenMode.ForRead);

              // "пробегаем" по всем объектам в пространстве модели
              foreach (ObjId id in ms)
              {
                  // приводим каждый из них к типу Entity
                  Entity entity = (Entity)tr.GetObject(id, OpenMode.ForRead);

                  // если цвет объекта - салатовый, то выводим в консоль слой,
тип и цвет каждого объекта
                  if (entity.Color == Autodesk.AutoCAD.Colors.Color.FromRgb(128,
255, 64))
                  {

acad.DocumentManager.MdiActiveDocument.Editor.WriteMessage(string.Format("\nLayer:
{0}; Type:{1}; Colors: {2},{3},{4}\n»,
                          entity.Layer,              entity.GetType().ToString(),
entity.Color.Red.ToString(),                        entity.Color.Green.ToString(),
entity.Color.Blue.ToString()));
                  }
              }
              tr.Commit();
          }
      }
```

Класс Entity задает свойства, характерные для любого объекта чертежа: слой, цвет, начертание. Для поиска по свойствам, специфичным для какого-то конкретного класса объектов, необходимо вначале найти все объекты такого типа, затем выполнить приведение к этому типу и просмотреть значение нужного свойства.

С помощью описанных в ней свойств создаваемый плагин будет находить нужные параметры на чертеже.

Ознакомиться со свойствами и методами класса Entity можно в контекстном меню:

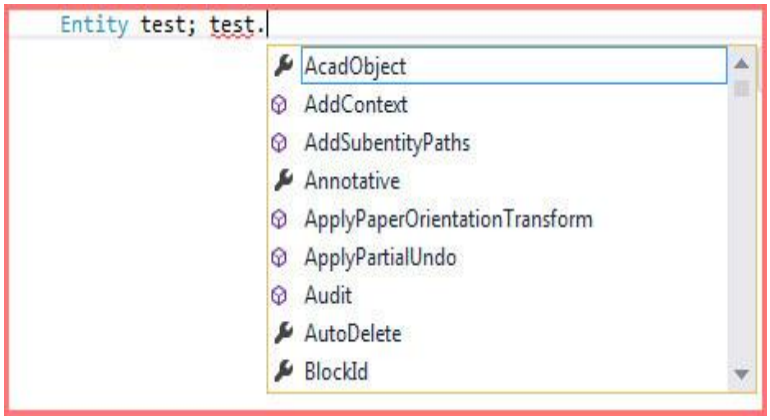


Рис.2.8. Свойства касса Entity

Для улучшения совместимости плагина и AutoCAD будем учитывать при поиске свойства объектов. Когда чертёж создаётся с помощью сторонних программ. использующих при создании чертежа собственные объекты то при открытии в AutoCAD заменяет неизвестные ему объекты на понятные ему примитивы.

Командой _APPLOAD вызываем окно загрузки программ. Команда REMOVEALLPROXY удаляет все proxy-объекты и proxy-примитивы. Команда EXPLODEALLPROXY Разбивает вообще все proxy-объекты, неважно, где они лежат. Это делается для корректности поисков на чертеже.

Найдем все вхождения блока «block-1». Класс Entity не содержит сведений об имени определения блока; зато эта информация есть в свойстве Name класса BlockReference. И таким образом поможет поиску (листинг 8):

Листинг 8. Поиск вхождений блока 1

```
      [CommandMethod("Habr_FindBlocks_1")]
      public void findBlocks_1()
      {
          using (Transaction tr = db.TransactionManager.StartTransaction())
          {
              BlockTableRecord                          ms                          =
(BlockTableRecord)tr.GetObject(SymbolUtilityServices.GetBlockModelSpaceId(db),
OpenMode.ForRead);
              foreach (ObjId id in ms)
              {
                  Entity entity = (Entity)tr.GetObject(id, OpenMode.ForRead);
                  if (entity.GetType() == typeof(BlockReference))
                  {
                      BlockReference br = (BlockReference)entity;
                      if (br.Name == "block-1")
                      {

acad.DocumentManager.MdiActiveDocument.Editor.WriteMessage(string.Format("\nLayer:
{0}; Type:{1}; Colors: {2},{3},{4}\n»,
                                  entity.Layer,           entity.GetType().ToString(),
entity.Color.Red.ToString(),                          entity.Color.Green.ToString(),
entity.Color.Blue.ToString()));
                      }
                  }
              }

              tr.Commit();
          }
      }
```

В результате мы имеет нужные нам координаты и линии (рис.2.9):

```
Command: Habr_IterateThrougAllObjects_2
Layer:layer-2; Type:Autodesk.AutoCAD.DatabaseServices.BlockReference; Color:
0,0,0
Layer:layer-2; Type:Autodesk.AutoCAD.DatabaseServices.BlockReference; Color:
0,0,0
Layer:layer-2; Type:Autodesk.AutoCAD.DatabaseServices.Circle; Color: 128,255,64
Layer:layer-2; Type:Autodesk.AutoCAD.DatabaseServices.Arc; Color: 0,200,255
Layer:layer-2; Type:Autodesk.AutoCAD.DatabaseServices.Line; Color: 255,128,255
Layer:layer-1; Type:Autodesk.AutoCAD.DatabaseServices.BlockReference; Color:
0,0,0
Layer:layer-1; Type:Autodesk.AutoCAD.DatabaseServices.DBText; Color: 255,128,255
Layer:layer-1; Type:Autodesk.AutoCAD.DatabaseServices.Circle; Color: 128,255,64
Layer:layer-1; Type:Autodesk.AutoCAD.DatabaseServices.Line; Color: 0,200,255
Layer:layer-1; Type:Autodesk.AutoCAD.DatabaseServices.Line; Color: 255,128,255
Layer:0; Type:Autodesk.AutoCAD.DatabaseServices.DBText; Color: 128,255,64
Layer:0; Type:Autodesk.AutoCAD.DatabaseServices.Circle; Color: 0,200,255
Layer:0; Type:Autodesk.AutoCAD.DatabaseServices.Polyline; Color: 255,128,255
Layer:0; Type:Autodesk.AutoCAD.DatabaseServices.Line; Color: 128,255,64
Layer:0; Type:Autodesk.AutoCAD.DatabaseServices.Line; Color: 0,200,255
Layer:0; Type:Autodesk.AutoCAD.DatabaseServices.Line; Color: 255,128,255
```

Рис. 2.9. Вывод в консоль всех объектов балки

Поиск объектов чертежа с помощью метода Editor.SelectAll() поможет сразу получаем массив идентификаторов всех интересующих нас объектов, при необходимости отфильтровывая нужные (листинг 9).

Листинг 9. Получение все объектов чертежа

```
      // команда для итерации по объектам (подход 2)
      [CommandMethod("Habr_IterateThroughAllObjects_2")]
      public void iterateThroughAllObjects_2()
      {
          // получаем БД и Editor текущего документа
          Document doc = Application.DocumentManager.MdiActiveDocument;
          Database db1= doc.Database;
          Editor ed = doc.Editor;

          // пытаемся получить ссылки на все объекты
          // ВНИМАНИЕ! Нужно проверить работоспособность метода с замороженными и
заблокированными слоями!
          PromptSelectionResult selRes = ed.SelectAll();

          // если произошла ошибка - сообщаем о ней
          if (selRes.Status != PromptStatus.OK)
          {
              ed.WriteMessage("\nError!\n");
              return;
          }

          // получаем массив ID объектов
          ObjId[] ids = selRes.Value.GetObjIds();

          // начинаем транзакцию
          using (Transaction tr = db.TransactionManager.StartTransaction())
          {
              // "пробегаем" по всем полученным объектам
              foreach (ObjId id in ids)
              {
                  // приводим каждый из них к типу Entity
                  Entity entity = (Entity)tr.GetObject(id, OpenMode.ForRead);

                  //    выводим    в    консоль    слой    (entity.Layer),    тип
(entity.GetType().ToString()) и цвет (entity.Color) каждого объекта

acad.DocumentManager.MdiActiveDocument.Editor.WriteMessage(string.Format("\nLayer:
{0}; Type:{1}; Colors: {2},{3},{4}\n»,
                      entity.Layer,                  entity.GetType().ToString(),
entity.Color.Red.ToString(),                        entity.Color.Green.ToString(),
entity.Color.Blue.ToString()));
              }

              tr.Commit();
          }
      }
```

Мы вызываем метод Editor.SelectAll(). Так как при вызове метода не используются фильтры, нам должны вернуться идентификаторы (ObjId) всех объектов на чертеже. Они записываются в переменную типа PromptSelectionResult.

На всякий случай нужно убедиться, что метод отработал корректно — для этого мы проверяем статус результата (PromptSelectionResult.Status). Если что-то не так, значение этого свойства будет отлично от PromptStatus.OK — в этом случае мы завершаем выполнение функции.

Если же метод Editor.SelectAll() отработал корректно, мы получаем идентификаторы всех объектов, возвращенных этим методом. Для этого используется метод PromptSelectionResult.Value.GetObjIds(). Затем идет обработка всех объекты в цикле — точь-в-точь как в первом разделе, когда мы обращались к ModelSpace.

Для использования метода Editor.SelectAll() необходимо существование объекта класса Editor.

При работе с документом, который непосредственно открыт в AutoCAD, проблем не возникнет; но использовать метод для обработки БД сторонних документов (не открытых в настоящий момент в AutoCAD) не получится.

Чтобы найти пересечения нужно найти линии из разных слоев чертежа на слое «layer-1» и все круги на слое «layer-2». Очевидно, мы не сможем этого сделать, просто добавив несколько объектов TypedValue.

Итак, нам нужно реализовать выбор по такому условию:

((СЛОЙ == «layer-1») И

(ТИП == «Линия»)) ИЛИ

((СЛОЙ == «layer-2») И (ТИП == «Окружность»))

В синтаксисе AutoCAD наше условие можно записать так (листинг 10):

Листинг 10. Общее условие поиска в AutoCAD

```
<OR

<AND

Layer == «layer-1»
Entity type == «LINE»
AND>
<AND

Layer == «layer-2»
Entity type == «CIRCLE»
AND>
OR>
```

А вот так это условие перепишется в виде массива элементов TypedValue (листинг 11):

Листинг 11. Массив элементов TypedValue

```
TypedValue[] filterlist = new TypedValue[10];
filterlist[0] = new TypedValue((int)DxfCode.Operator, "<OR");
filterlist[1] = new TypedValue((int)DxfCode.Operator, "<AND");
filterlist[2] = new TypedValue((int)DxfCode.LayerName, "layer-1");
filterlist[3] = new TypedValue((int)DxfCode.Start, "LINE");
filterlist[4] = new TypedValue((int)DxfCode.Operator, "AND>");
filterlist[5] = new TypedValue((int)DxfCode.Operator, "<AND");
filterlist[6] = new TypedValue((int)DxfCode.LayerName, "layer-2");
filterlist[7] = new TypedValue((int)DxfCode.Start, "CIRCLE");
filterlist[8] = new TypedValue((int)DxfCode.Operator, "AND>");
filterlist[9] = new TypedValue((int)DxfCode.Operator, "OR>");
```

После применения фильтра на базе этих условий мы увидим следующий вывод в консоли AutoCAD (листинг 12):

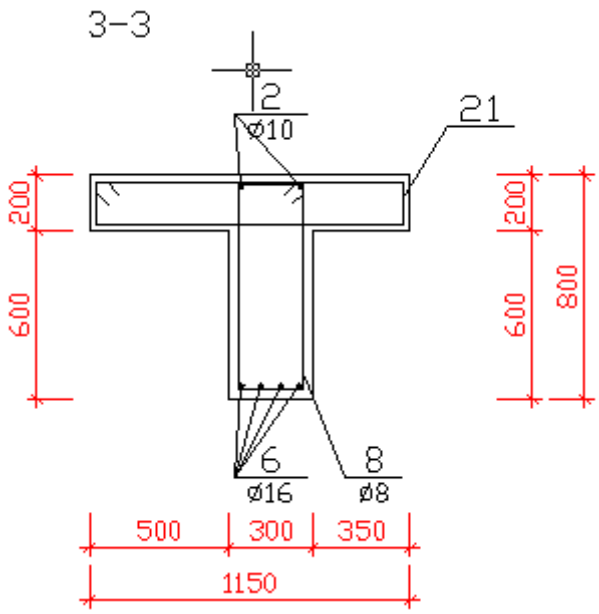
Листинг 12. Вывод в консоль

```
Layer:layer-2;     Type:Autodesk.AutoCAD.DatabaseServices.Circle;     Color:
128,255,64
Layer:layer-1; Type:Autodesk.AutoCAD.DatabaseServices.Line; Color: 0,200,255
Layer:layer-1; Type:Autodesk.AutoCAD.DatabaseServices.Line; Color: 255,128,255
```

Объединяем необходимые части чертежа балки с помощью операторов OR, AND, NOT, XOR.

Рассмотренный способ можно использовать для модификации всех изменяемых свойств, которые доступны в классе Entity. Плагину удастся получить доступ к объекту на запись, либо делаем проверку. Флаг forceOpenOnLockedLayer или средства индивидуальной защиты

конструкцию try...catch и перехватывайте возможные исключения способствуют корректности кода и полученных значений чертежа для расчетов сечения балки и расчета долговечности.



Нагрузки, кг/пог. м	Сечение балок при длине пролета, м						
	3,0	3,5	4,0	4,5	5,0	5,5	6,0
150	50x140	50x160	60x180	80x180	80x200	100x200	100x220
200	50x160	50x180	70x180	70x200	100x200	120x220	140x220
250	60x160	60x180	70x200	100x200	120x200	140x220	160x220
350	70x160	70x180	80x200	100x220	120x220	160x220	200x220

Рис. 2.10. Сечение балки и ее расчетные данные

Таким образом разработанный модуль решает несколько задач – поиск, обработку и анализ элементов чертежа, на основании которых и будет происходить расчет инженерных характеристик.

ГЛАВА 3. ТЕСТИРОВАНИЕ И ОТЛАДКА МОДУЛЯ

В данной главе демонстрируется установка и применение полученного модуля в системе AutoCAD.

После сборки этого проекта у нас получится готовый к употреблению плагин.

Для его применения необходимо запустить AutoCAD и выполнить команду «NETLOAD» (рис.3.1):

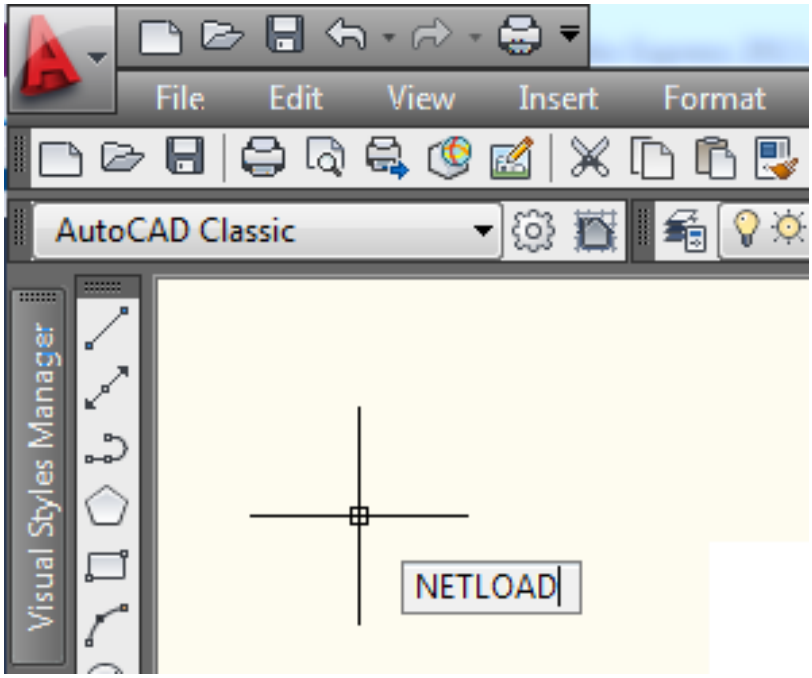


Рис.3.1. Запуск команды NETLOAD

Команда производит загрузку приложения .NET. Отображается диалоговое окно "Выбор сборки .NET" (стандартное диалоговое окно выбора файлов). Если значение системной переменной FILEDIA равно 0 (нуль), команда NETLOAD выводит в командной строке следующую подсказку: “Имя файла сборки: *Ввести имя файла и нажать* ENTER”.

Затем в открывшемся окне указать путь к файлу плагина (рисунок 3.2):

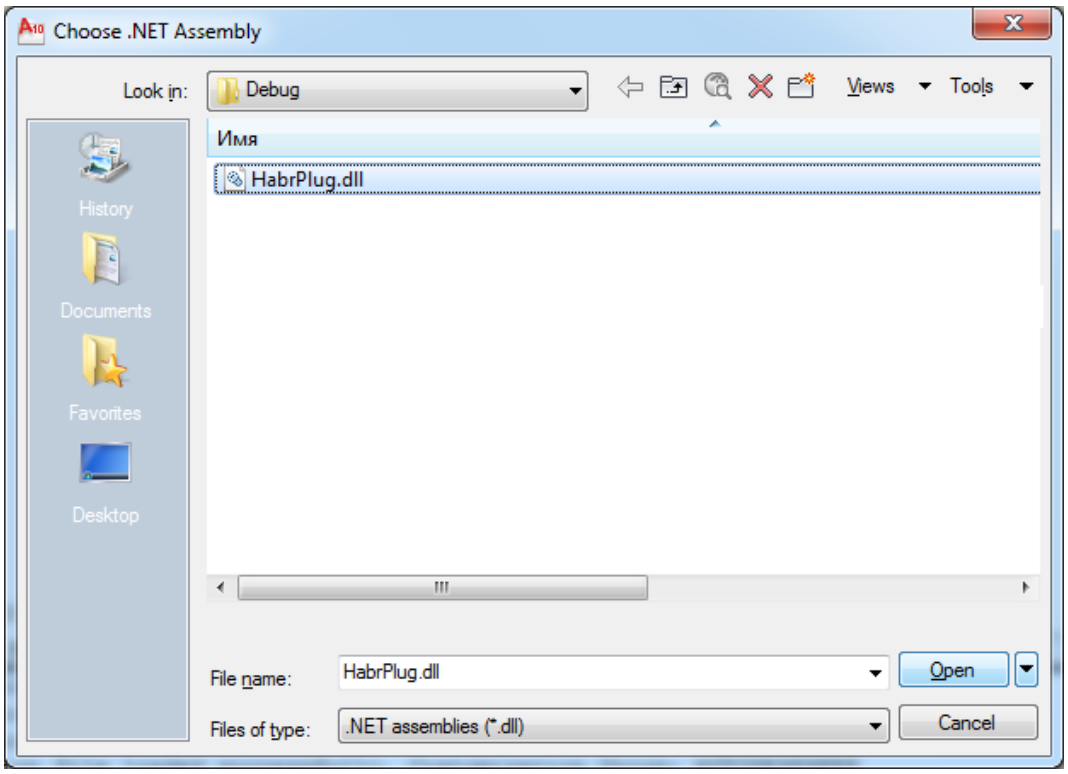


Рис.3.2. Выбор плагина для запуска

После этого плагин будет загружен в AutoCAD. Мы должны увидеть первое сообщение (рисунок 3.3):

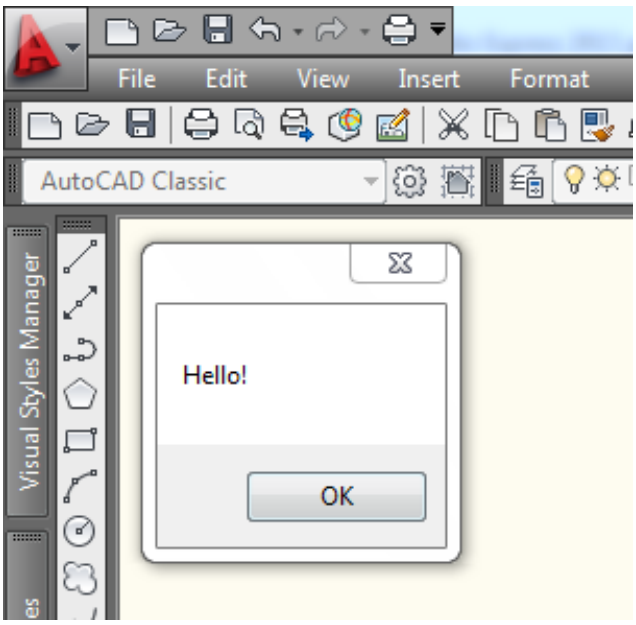


Рис.3.3. Загруженный плагин в AutoCAD

Если при загрузке плагина произошла критическая ошибка, она будет выведена в консоль AutoCAD (рис.3.4):

```
Command: NETLOAD
Cannot load assembly. Error details: System.BadImageFormatException: Could not
load file or assembly
'file:///C:\VSProjects\HabrPlug\HabrPlug\bin\Debug\HabrPlug.dll' or one of its
dependencies. This assembly is built by a runtime newer than the currently
loaded runtime and cannot be loaded.
File name: 'file:///C:\VSProjects\HabrPlug\HabrPlug\bin\Debug\HabrPlug.dll'
   at System.Reflection.Assembly._nLoad(AssemblyName fileName, String codeBase,
Evidence assemblySecurity, Assembly locationHint, StackCrawlMark& stackMark,
Boolean throwOnFileNotFound, Boolean forIntrospection)
```

Рис.3.4. Ошибка загрузки плагина

Если плагин не смог загрузиться из-за ошибки, то перед тестированием очередного (исправленного) варианта нужно закрыть и заново запустить AutoCAD. В противном случае он может отказаться загружать плагин, даже если ошибок в коде уже не будет.

Теперь, когда плагин загружен, можно выполнить нашу тестовую команду (3.5):

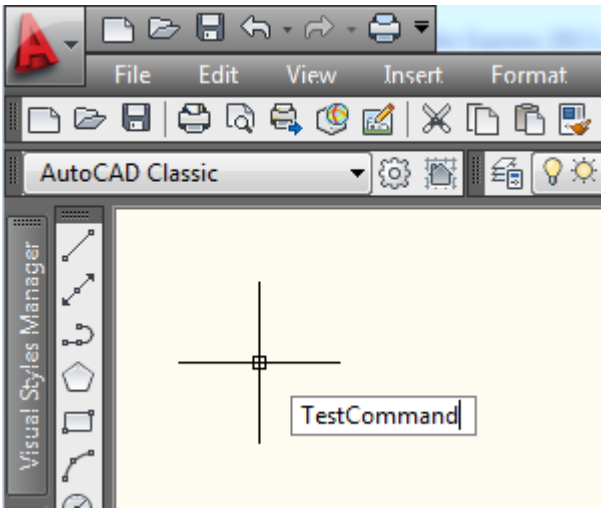


Рис.3.5. Тестовые команды для работы плагина

3.1 Отладка плагина

Так как включение отладки кода при написании плагинов под AutoCAD связано с определенными трудностями и соответственно тратой времени на поиск решения, пошагово рассмотрим, каким образом включить отладку в Visual Studio.

В проекте используется .NET Framework 3.5. На рисунках 3.6 -3.14 рассмотрим подробную отладку модуля.

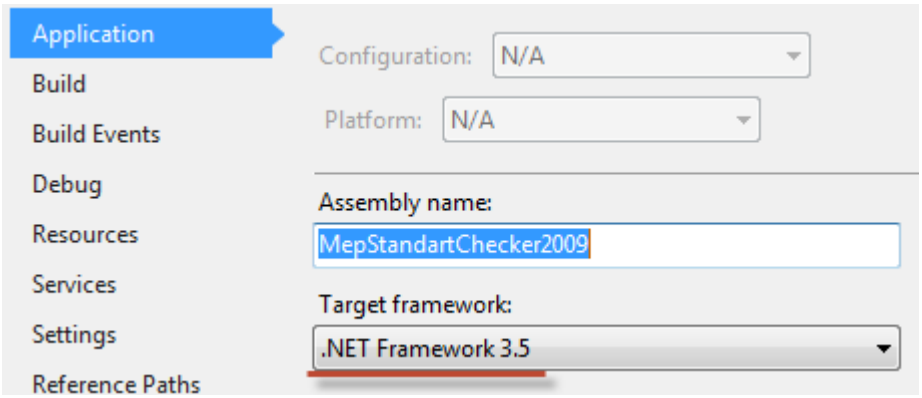


Рис.3.6. Создаем новый проект для отладки

Файл acad.exe.config изменять не надо.

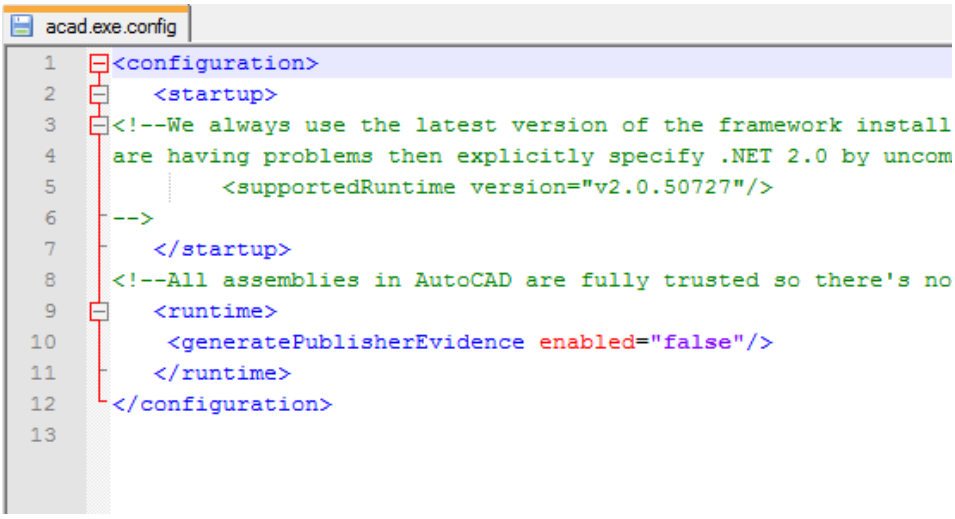


Рис.3.7. Конфигурация отладочного проекта

Шаг 1. Добавить в Solution существующий проект, указав файл acad.exe (у меня он расположен по адресу C:\Program Files\AutoCAD Architecture 2009\acad.exe).

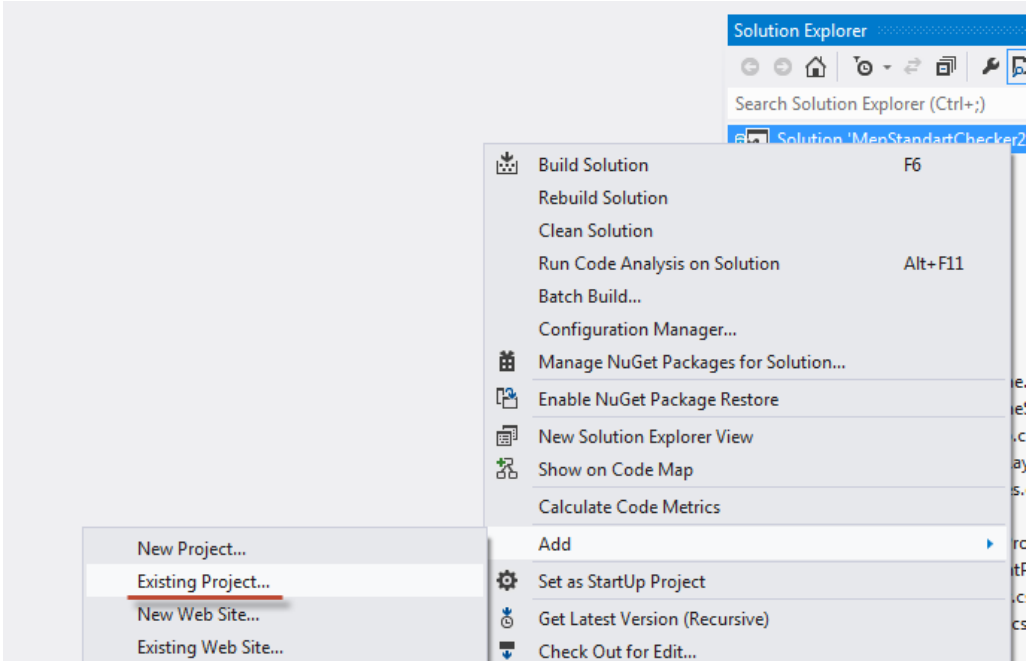


Рис.3.8. Добавление исполняемого файла в плагин

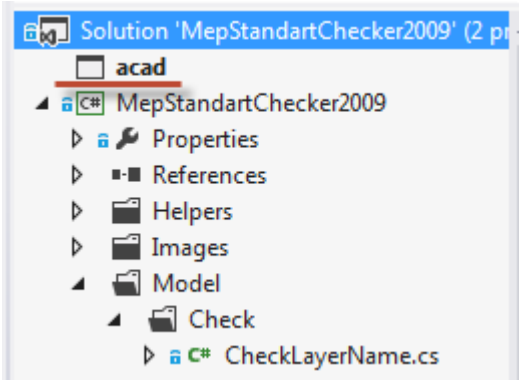


Рис.3.9. Отладочный файл в системе проекта

Шаг 2. Назначить проект **acad** стартовым проектом.

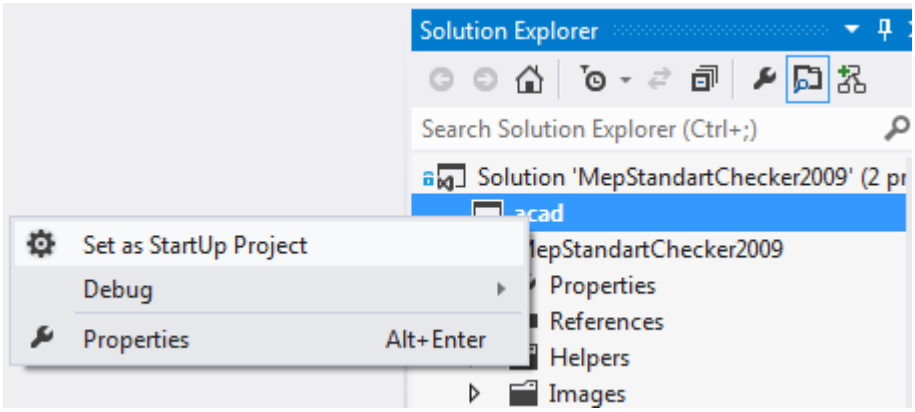


Рис.3.10. Назначаем отладчик стартовым проектом

Шаг 3. В свойствах проекта acad свойству Debugger Type установить значение Managed (v3.5, v3.0, v2.0).

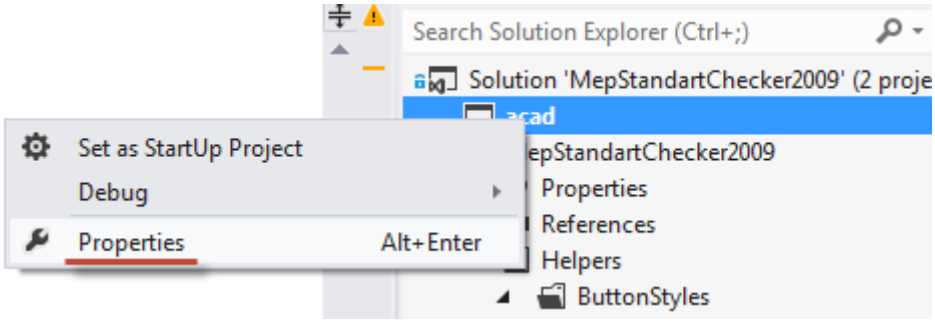


Рис. 3.11. Изменение свойств проекта

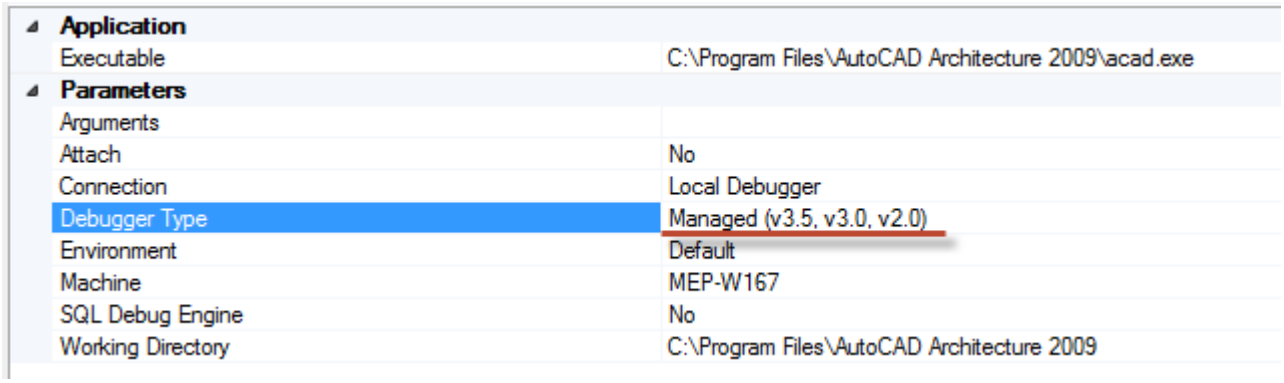


Рис.3.12. Изменение значений свойств

Шаг 4. После запуска отладки (F5) запустится AutoCAD. Загружаем сборку набрав в командной строке AutoCAD команду NETLOAD и выбрав необходимую dll.

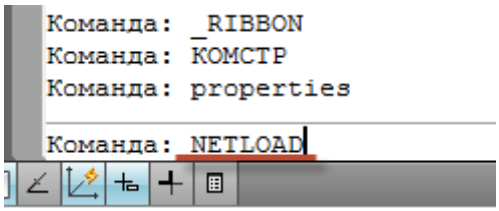
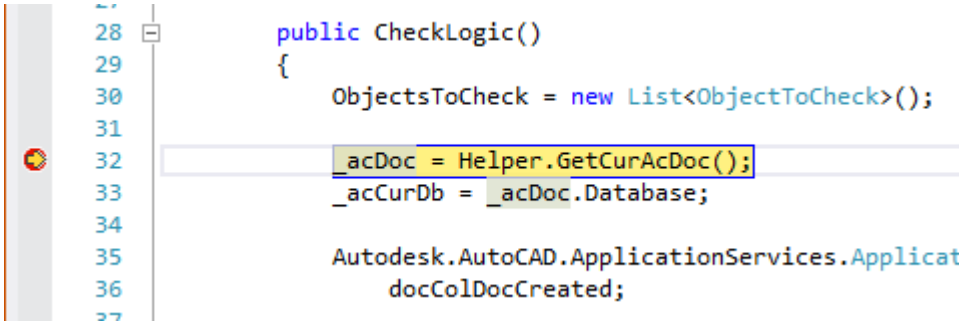


Рис.3.13. Запускаем проект

При запуске плагина видим, что Visual Studio прерывает выполнение программы в указанных точках останова.



3.14. Тестируем с отладкой модуль

И проверяем полученные значения прочности конструкции. Программа работает корректно.

ЗАКЛЮЧЕНИЕ

Перспективой развития САПР, является тесная интеграция с программами смежных направлений. Суть этого процесса заключается, например, во взаимосвязи между чертежными и расчетными программами. Если после проектирования здания необходимо рассчитать смету, передать данные в бухгалтерскую программу или произвести расчет каких-либо конструкций, программы должны быть взаимосвязаны. Такая интеграция позволит автоматизировать в едином информационном пространстве все стадии строительства и проектирования.

Главной целью данной работы являлась в разработка программного модуля для анализа устойчивости и долговечности инженерных конструкций. В процессе разработки программа также решены поставленные задачи:

- Проведен обзор методов для анализа инженерных конструкций\
- Изучен перечень программного обеспечения для анализа инженерных конструкций
- Определены требований к создаваемому программному модулю
- Разработана архитектура программного модуля
- Создан и отлажен в программе AutoCad разработанный модуль для расчетов

Строительство всегда развивалось в ногу с научно-техническим прогрессом, но совершенствование программных средств далеко опережает квалификацию специалистов, призванных использовать их в своей работе. Сегодня часто наблюдается картина, когда современные и многофункциональные комплексы простаивают или используются незначительно из-за низкого уровня подготовки пользователей.

Без предварительного исследования предприятия и квалифицированной помощи специалистов невозможно правильно выбрать программные средства, которые не только бы решали поставленные задачи, но и обеспечивали полную комплексную автоматизацию. В противном случае, вложение средств в автоматизацию может обернуться простоем программ или только решением очень узких задач на предприятии.

Разработанный модуль позволит ускорить расчетные работы по проектам и упростит работу специалистов. Приложение успешно протестировано и отлажено на существующих проектах.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1.Сторчак Н. Применение системы КОМПАС3D в преподавании инженерных дисциплин // САПР и графика. 2013. ? 10. С. 8889.

2.Клещёва Н.А., Тарасова И.М. Применение теории графов в процессе формирования системы математической подготовки бакалавров // International journal of applied and fundamental research. 2015. ? 1. С. 130-135.

3.Горелов В.Н, Кокорев И.А. Принципы построения 3Dмоделей корпусных деталей в системе КОМПАС3D // МТО13, 18 октября 2013 года. Том 1. С. 321324.

4.Дудак Н.С., Касенов А.Ж., Муканов Р.Б., Оспантаев А.К., Истай Т.Б., Миллер С.А., Ахметова А.А. Прочностной конечноэлементный экспрессанализ // Материалы Международной научной конференции молодых ученых, магистрантов, студентов и школьников «XVI Сатпаевские чтения». Том 16. Павлодар: ПГУ им. С. Торайгырова, 2016. С. 4754.

5.Мендебаев Т.М., Дудак Н.С., Касенов А.Ж., Муканов Р.Б., Смаилова Г.А. Применение системы Autodesk Inventor при проектировании резцовой головки для обработки отверстий // Труды Международных Сатпаевских чтений «Конкурентоспособность технической науки и образования», Т.1, часть 1. Алматы, 2016. С. 238243.

6.Дудак Н.С., Муканов Р.Б., Касенов А.Ж., Таскарина А.Ж. Применение системы АРМ WinMachine при проектировании металлорежущих инструментов // Материалы X Международной научно-практической конференции «Ключевые вопросы современной науки2014», 1725 апреля 2014 г., Т. 37. Технологии. София «БелГрадБГ» ООД. 2014. С. 4750.

7.Махов А.А., Копейкин Е.А. Проектирование шлицевых протяжек с применением MatLab и TFlex CAD 3D // Вестник МГТУ «Станкин». 2014. ? 3 (30), С. 7378.

8.Горбунов И.В., Ефременков И.В., Леонтьев В.Л., Гисметулин А.Р. Особенности моделирования процессов механической обработки в CAE-системах // Известия Самарского научного центра Российской академии наук. 2013. ? 44, Т. 15. С. 846853.

9.Денисова Н., Доронин А., Завалишин Ю., Меньков А., Терушкина Н. Применение программного обеспечения «Аскон» в реализации образовательных дисциплин вуза // САПР и графика. 2014. ? 3. С. 7479.

10.Компания «Топ Системы» выпустила новую версию системы автоматизированного проектирования TFlex CAD 15/ URL: http://www.mashportal.ru/company_news43039.aspx.

11.Преимущества КОМПАС3D в САПР — небольшой обзор/ URL: http://texdizain.net/proektirovanie/37preimuschestvakompas3dvsapr-nebolshoyobzor.html.

12.Шелофаст В., Розинский С. Программные продукты компании НТЦ «АПМ» — новые возможности и перспективы // САПР и графика. 2015. ? 8 (226). С. 5258.

13.Фрей Д. Изучаем AutoCAD® 2007 и AutoCAD® LT 2007 с самого начала. AutoCAD® 2007 и AutoCAD® LT 2007: Практическое руководство / Д. Фрей; [пер. с англ. И. Л. Волкова]. Москва, 2008. 688 с.

14.Денисов М.А. Компьютерное проектирование. ANSYS. Учебное пособие / Министерство образования и науки РФ, Уральский федеральный университет им. первого Президента России Б.Н. Ельцина. Екатеринбург, 2014. С. 77.

15.Comsol. Программный пакет для мультифизического моделирования / URL: https://www.comsol.ru/products.

16.Simufact Forming / URL: http://www.lavteam.org/tags/Simufact.

17.Adams — система виртуального моделирования машин и механизмов / URL: http://rusapr.ru/prod/progs/element.php?ID=835.

18.САПР для машиностроения и промышленного производства / Инженерные расчеты и моделирование технологических процессов / MSC. Nastran / URL: http://www.cad.ru/ru/software/detail.php?ID=3181.

ПРИЛОЖЕНИЕ

acadMODULE.cs

```
using System;
using System.Collections.Generic;

using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.EditorInput;
using acad = Autodesk.AutoCAD.ApplicationServices.Application;

namespace HabrPlug_SearchAndRescue
{
   public class ClassMyAutoCADDLL_SearchAndRescue
   {
      public class Commands : IExtensionApplication
      {
         // используемые цвета
         Autodesk.AutoCAD.Colors.Color color_Pink =
Autodesk.AutoCAD.Colors.Color.FromRgb(255, 128, 255);
         Autodesk.AutoCAD.Colors.Color color_Blue =
Autodesk.AutoCAD.Colors.Color.FromRgb(0, 200, 255);
         Autodesk.AutoCAD.Colors.Color color_LightGreen =
Autodesk.AutoCAD.Colors.Color.FromRgb(128, 255, 64);

         // ID слоев "layer-1" и "layer-2"
         ObjectId layer_1;
         ObjectId layer_2;

         // создаем слои
         public void createLayers()
         {
            // получаем текущий документ и его БД
            Document acDoc =
Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActi
veDocument;
            Database acCurDb = acDoc.Database;
```

```
            // блокируем документ
            using (DocumentLock docloc = acDoc.LockDocument())
            {
               // начинаем транзакцию
               using (Transaction tr =
acCurDb.TransactionManager.StartTransaction())
               {
                  // открываем таблицу слоев документа
                  LayerTable acLyrTbl = tr.GetObject(acCurDb.LayerTableId,
OpenMode.ForWrite) as LayerTable;

                  // создаем новый слой и задаем ему имя
                  LayerTableRecord acLyrTblRec_1 = new LayerTableRecord();
                  acLyrTblRec_1.Name = "layer-1";
                  // заносим созданный слой в таблицу слоев, сохраняем ID
созданной записи слоя
                  layer_1 = acLyrTbl.Add(acLyrTblRec_1);
                  // добавляем созданный слой в документ
                  tr.AddNewlyCreatedDBObject(acLyrTblRec_1, true);


                  // создаем новый слой и задаем ему имя
                  LayerTableRecord acLyrTblRec_2 = new LayerTableRecord();
                  acLyrTblRec_2.Name = "layer-2";
                  // заносим созданный слой в таблицу слоев, сохраняем ID
созданной записи слоя
                  layer_2 = acLyrTbl.Add(acLyrTblRec_2);
                  // добавляем созданный слой в документ
                  tr.AddNewlyCreatedDBObject(acLyrTblRec_2, true);

                  // фиксируем транзакцию
                  tr.Commit();
               }
            }
         }

         // создаем определение блока "block-1"
         public void createBlock_1()
         {
            // получаем ссылки на документ и его БД
            Document doc = Application.DocumentManager.MdiActiveDocument;
            Database db = doc.Database;

            // имя создаваемого блока
```

```
            const string blockName = "block-1";

            // начинаем транзакцию
            Transaction tr = db.TransactionManager.StartTransaction();
            using (tr)
            {
               // открываем таблицу блоков на запись
               BlockTable bt = (BlockTable)tr.GetObject(db.BlockTableId,
OpenMode.ForWrite);

               // проверяем, нет ли в таблице блока с таким именем; если есть -
заканчиваем выполнение команды
               if (bt.Has(blockName))
               {
                  return;
               }

               // создаем новое определение блока, задаем ему имя
               BlockTableRecord btr = new BlockTableRecord();
               btr.Name = blockName;

               // добавляем созданное определение блока в таблицу блоков и в
транзакцию
               bt.Add(btr);
               tr.AddNewlyCreatedDBObject(btr, true);

               // добавляем к блоку элементы

               // создаем окружность
               Circle acCircle = new Circle();
               // устанавливаем параметры созданного объекта
               acCircle.SetDatabaseDefaults();
               acCircle.Center = Point3d.Origin;
               acCircle.Radius = 25;
               // добавляем созданный объект в определение блока и в
транзакцию
               btr.AppendEntity(acCircle);
               tr.AddNewlyCreatedDBObject(acCircle, true);

               // создаем линию
               Line acLine = new Line(new Point3d(18, 18, 0), new Point3d(35, 35,
0));
               // устанавливаем параметры созданного объекта равными
параметрам по умолчанию
```

```
            acLine.SetDatabaseDefaults();
            // добавляем созданный объект в определение блока и в
транзакцию
            btr.AppendEntity(acLine);
            tr.AddNewlyCreatedDBObject(acLine, true);

            // создаем полилинию
            Polyline acPolyline = new Polyline();
            // устанавливаем параметры созданного объекта равными
параметрам по умолчанию
            acPolyline.SetDatabaseDefaults();
            // добавляем к полилинии вершины
            acPolyline.AddVertexAt(0, new Point2d(20, 35), 0, 0, 0);
            acPolyline.AddVertexAt(1, new Point2d(35, 35), 0, 0, 0);
            acPolyline.AddVertexAt(2, new Point2d(35, 20), 0, 0, 0);
            // добавляем созданный объект в определение блока и в
транзакцию
            btr.AppendEntity(acPolyline);
            tr.AddNewlyCreatedDBObject(acPolyline, true);

            // фиксируем транзакцию
            tr.Commit();
        }
    }

    // создаем определение блока "block-2"
    public void createBlock_2()
    {
        // получаем ссылки на документ и его БД
        Document doc = Application.DocumentManager.MdiActiveDocument;
        Database db = doc.Database;

        // имя создаваемого блока
        const string blockName = "block-2";

        // начинаем транзакцию
        Transaction tr = db.TransactionManager.StartTransaction();
        using (tr)
        {
            // открываем таблицу блоков на запись
            BlockTable bt = (BlockTable)tr.GetObject(db.BlockTableId,
OpenMode.ForWrite);

            // проверяем, нет ли в таблице блока с таким именем; если есть -
```

```
заканчиваем выполнение команды
            if (bt.Has(blockName))
            {
               return;
            }

            // создаем новое определение блока, задаем ему имя
            BlockTableRecord btr = new BlockTableRecord();
            btr.Name = blockName;

            // добавляем созданное определение блока в таблицу блоков и в
транзакцию
            bt.Add(btr);
            tr.AddNewlyCreatedDBObject(btr, true);

            // добавляем к блоку элементы

            // создаем окружность
            Circle acCircle = new Circle();
            // устанавливаем параметры созданного объекта
            acCircle.SetDatabaseDefaults();
            acCircle.Center = Point3d.Origin;
            acCircle.Radius = 25;
            // добавляем созданный объект определение блока и в
транзакцию
            btr.AppendEntity(acCircle);
            tr.AddNewlyCreatedDBObject(acCircle, true);

            // создаем первую линию
            Line acLine_1 = new Line(new Point3d(0, -25, 0), new Point3d(0, -
50, 0));
            // устанавливаем параметры созданного объекта равными
параметрам по умолчанию
            acLine_1.SetDatabaseDefaults();
            // добавляем созданный объект в определение блока и в
транзакцию
            btr.AppendEntity(acLine_1);
            tr.AddNewlyCreatedDBObject(acLine_1, true);

            // создаем вторую линию
            Line acLine_2 = new Line(new Point3d(-7, -39, 0), new Point3d(7, -
39, 0));
            // устанавливаем параметры созданного объекта равными
параметрам по умолчанию
```

```
            acLine_2.SetDatabaseDefaults();
            // добавляем созданный объект в определение блока и в
транзакцию
            btr.AppendEntity(acLine_2);
            tr.AddNewlyCreatedDBObject(acLine_2, true);

            // фиксируем транзакцию
            tr.Commit();
        }
    }

    // создаем объекты на нулевом слое
    public void layer_0_createObjects()
    {
        // получаем текущий документ и его БД
        Document doc = acad.DocumentManager.MdiActiveDocument;
        Database db = doc.Database;

        // начинаем транзакцию
        using (Transaction tr = db.TransactionManager.StartTransaction())
        {
            // открываем таблицу блоков документа
            BlockTable acBlkTbl;
            acBlkTbl = tr.GetObject(db.BlockTableId, OpenMode.ForRead) as
BlockTable;

            // открываем пространство модели (Model Space) - оно является
одной из записей в таблице блоков документа
            BlockTableRecord ms =
tr.GetObject(acBlkTbl[BlockTableRecord.ModelSpace], OpenMode.ForWrite) as
BlockTableRecord;

            // добавляем розовую линию
            Line acLine_1 = new Line(new Point3d(225, 225, 0), new
Point3d(225, 175, 0));
            // устанавливаем параметры созданного объекта равными
параметрам по умолчанию
            acLine_1.SetDatabaseDefaults();
            // устанавливаем для объекта нужный слой и цвет
            acLine_1.Layer = "0";
            acLine_1.Color = color_Pink;
            // добавляем созданный объект в пространство модели и в
транзакцию
            ms.AppendEntity(acLine_1);
```

```
            tr.AddNewlyCreatedDBObject(acLine_1, true);

            // добавляем голубую линию
            Line acLine_2 = new Line(new Point3d(250, 225, 0), new
Point3d(250, 175, 0));
            // устанавливаем параметры созданного объекта равными
параметрам по умолчанию
            acLine_2.SetDatabaseDefaults();
            // устанавливаем для объекта нужный слой и цвет
            acLine_2.Layer = "0";
            acLine_2.Color = color_Blue;
            // добавляем созданный объект в пространство модели и в
транзакцию
            ms.AppendEntity(acLine_2);
            tr.AddNewlyCreatedDBObject(acLine_2, true);

            // добавляем салатовую линию
            Line acLine_3 = new Line(new Point3d(275, 225, 0), new
Point3d(275, 175, 0));
            // устанавливаем параметры созданного объекта равными
параметрам по умолчанию
            acLine_3.SetDatabaseDefaults();
            // устанавливаем для объекта нужный слой и цвет
            acLine_3.Layer = "0";
            acLine_3.Color = color_LightGreen;
            // добавляем созданный объект в пространство модели и в
транзакцию
            ms.AppendEntity(acLine_3);
            tr.AddNewlyCreatedDBObject(acLine_3, true);

            // добавляем розовую полилинию
            Polyline acPolyline = new Polyline();
            // устанавливаем параметры созданного объекта равными
параметрам по умолчанию
            acPolyline.SetDatabaseDefaults();
            // добавляем к полилинии вершины
            acPolyline.AddVertexAt(0, new Point2d(300, 225), 0, 0, 0);
            acPolyline.AddVertexAt(1, new Point2d(325, 175), 0, 0, 0);
            acPolyline.AddVertexAt(2, new Point2d(350, 225), 0, 0, 0);
            // устанавливаем для объекта нужный слой и цвет
            acPolyline.Layer = "0";
            acPolyline.Color = color_Pink;
            // добавляем созданный объект в пространство модели и в
транзакцию
```

```
            ms.AppendEntity(acPolyline);
            tr.AddNewlyCreatedDBObject(acPolyline, true);

            // добавляем голубую окружность
            Circle acCircle = new Circle();
            // устанавливаем параметры созданного объекта
            acCircle.SetDatabaseDefaults();
            acCircle.Center = new Point3d(400, 200, 0);
            acCircle.Radius = 25;
            // устанавливаем для объекта нужный слой и цвет
            acCircle.Layer = "0";
            acCircle.Color = color_Blue;
            // добавляем созданный объект в пространство модели и в
транзакцию
            ms.AppendEntity(acCircle);
            tr.AddNewlyCreatedDBObject(acCircle, true);

            // добавляем салатовый текст
            DBText text = new DBText();
            text.Position = new Point3d(450, 175, 0);
            text.Height = 50;
            text.TextString = "HABR!";
            // устанавливаем для объекта нужный слой и цвет
            text.Layer = "0";
            text.Color = color_LightGreen;
            // добавляем созданный объект в пространство модели и в
транзакцию
            ms.AppendEntity(text);
            tr.AddNewlyCreatedDBObject(text, true);

            // фиксируем изменения
            tr.Commit();
        }
    }

    // создаем объекты на слое "layer-1"
    public void layer_1_createObjects()
    {
        // получаем текущий документ и его БД
        Document doc = acad.DocumentManager.MdiActiveDocument;
        Database db = doc.Database;

        // начинаем транзакцию
        using (Transaction tr = db.TransactionManager.StartTransaction())
```

```
        {
            // запоминаем текущий слой и временно меняем его на нужный
нам
            // (это позволит не задавать слой отдельно для каждого
создаваемого объекта)
            ObjectId currentLayer = db.Clayer;
            db.Clayer = layer_1;

            // открываем таблицу блоков документа
            BlockTable acBlkTbl;
            acBlkTbl = tr.GetObject(db.BlockTableId, OpenMode.ForRead) as
BlockTable;

            // открываем пространство модели (Model Space) - оно является
одной из записей в таблице блоков документа
            BlockTableRecord ms =
tr.GetObject(acBlkTbl[BlockTableRecord.ModelSpace], OpenMode.ForWrite) as
BlockTableRecord;

            // добавляем розовую линию
            Line acLine_1 = new Line(new Point3d(225, 25, 0), new Point3d(225,
-25, 0));
            // устанавливаем параметры созданного объекта
            acLine_1.SetDatabaseDefaults();
            acLine_1.Color = color_Pink;
            // добавляем созданный объект в пространство модели и в
транзакцию
            ms.AppendEntity(acLine_1);
            tr.AddNewlyCreatedDBObject(acLine_1, true);

            // добавляем голубую линию
            Line acLine_2 = new Line(new Point3d(250, 25, 0), new Point3d(250,
-25, 0));
            // устанавливаем параметры созданного объекта
            acLine_2.SetDatabaseDefaults();
            acLine_2.Color = color_Blue;
            // добавляем созданный объект в пространство модели и в
транзакцию
            ms.AppendEntity(acLine_2);
            tr.AddNewlyCreatedDBObject(acLine_2, true);

            // добавляем салатовую окружность
            Circle acCircle = new Circle();
            // устанавливаем параметры созданного объекта
```

```
                acCircle.SetDatabaseDefaults();
                acCircle.Center = new Point3d(300, 0, 0);
                acCircle.Radius = 25;
                acCircle.Color = color_LightGreen;
                // добавляем созданный объект в пространство модели и в
транзакцию
                ms.AppendEntity(acCircle);
                tr.AddNewlyCreatedDBObject(acCircle, true);

                // добавляем розовый текст
                DBText text = new DBText();
                // устанавливаем параметры созданного объекта
                text.Position = new Point3d(350, -25, 0);
                text.Height = 50;
                text.TextString = "HABR!";
                text.Color = color_Pink;
                // добавляем созданный объект в пространство модели и в
транзакцию
                ms.AppendEntity(text);
                tr.AddNewlyCreatedDBObject(text, true);

                // добавляем вхождение блока "block-1"
                // открываем таблицу блоков для чтения
                BlockTable bt = (BlockTable)tr.GetObject(db.BlockTableId,
OpenMode.ForRead);
                // получаем ObjectID блока
                ObjectId btrId = bt["block-1"];
                // создаем новое вхождение блока, используя полученный ID
определения блока
                BlockReference br = new BlockReference(new Point3d(600, 0, 0),
btrId);
                // добавляем вхождение блока на пространство модели и в
транзакцию
                ms.AppendEntity(br);
                tr.AddNewlyCreatedDBObject(br, true);

                // возвращаем обратно старый текущий слой
                db.Clayer = currentLayer;

                // фиксируем изменения
                tr.Commit();
            }
        }
```

```
        // создаем объекты на слое "layer-2"
        public void layer_2_createObjects()
        {
            // получаем текущий документ и его БД
            Document doc = acad.DocumentManager.MdiActiveDocument;
            Database db = doc.Database;

            // начинаем транзакцию
            using (Transaction tr = db.TransactionManager.StartTransaction())
            {
                // запоминаем текущий слой и временно меняем его на нужный
нам
                // (это позволит не задавать слой отдельно для каждого
создаваемого объекта)
                ObjectId currentLayer = db.Clayer;
                db.Clayer = layer_2;

                // открываем таблицу блоков документа
                BlockTable acBlkTbl;
                acBlkTbl = tr.GetObject(db.BlockTableId, OpenMode.ForRead) as
BlockTable;

                // открываем пространство модели (Model Space) - оно является
одной из записей в таблице блоков документа
                BlockTableRecord ms =
tr.GetObject(acBlkTbl[BlockTableRecord.ModelSpace], OpenMode.ForWrite) as
BlockTableRecord;

                // добавляем розовую линию
                Line acLine_1 = new Line(new Point3d(225, -175, 0), new
Point3d(225, -225, 0));
                // устанавливаем параметры созданного объекта
                acLine_1.SetDatabaseDefaults();
                acLine_1.Color = color_Pink;
                // добавляем созданный объект в пространство модели и в
транзакцию
                ms.AppendEntity(acLine_1);
                tr.AddNewlyCreatedDBObject(acLine_1, true);

                // добавляем голубую дугу
                Arc acArc = new Arc(new Point3d(250, -200, 0), 25, -45 / 180.0 *
Math.PI, 45 / 180.0 * Math.PI);
                // устанавливаем параметры созданного объекта
                acArc.SetDatabaseDefaults();
```

```
            acArc.Color = color_Blue;
            // добавляем созданный объект в пространство модели и в
транзакцию
            ms.AppendEntity(acArc);
            tr.AddNewlyCreatedDBObject(acArc, true);

            // добавляем салатовую окружность
            Circle acCircle = new Circle();
            // устанавливаем параметры созданного объекта
            acCircle.SetDatabaseDefaults();
            acCircle.Center = new Point3d(325, -200, 0);
            acCircle.Radius = 25;
            acCircle.Color = color_LightGreen;
            // добавляем созданный объект в пространство модели и в
транзакцию
            ms.AppendEntity(acCircle);
            tr.AddNewlyCreatedDBObject(acCircle, true);

            // добавляем вхождение блока "block-1"
            // открываем таблицу блоков для чтения
            BlockTable bt = (BlockTable)tr.GetObject(db.BlockTableId,
OpenMode.ForRead);
            // получаем ObjectID блока
            ObjectId btrId = bt["block-1"];
            // создаем новое вхождение блока, используя полученный ID
определения блока
            BlockReference br = new BlockReference(new Point3d(400, -200, 0),
btrId);
            // добавляем вхождение блока на пространство модели и в
транзакцию
            ms.AppendEntity(br);
            tr.AddNewlyCreatedDBObject(br, true);

            // добавляем вхождение блока "block-2"
            // получаем ObjectID блока
            btrId = bt["block-2"];
            // создаем новое вхождение блока, используя полученный ID
определения блока
            br = new BlockReference(new Point3d(475, -200, 0), btrId);
            // добавляем вхождение блока на пространство модели и в
транзакцию
            ms.AppendEntity(br);
            tr.AddNewlyCreatedDBObject(br, true);
```

```
            // возвращаем обратно старый текущий слой
            db.Clayer = currentLayer;

            // фиксируем изменения
            tr.Commit();
        }
    }

    // создаем объекты при загрузке плагина
    public void Initialize()
    {
        createLayers();
        createBlock_1();
        createBlock_2();
        layer_0_createObjects();
        layer_1_createObjects();
        layer_2_createObjects();
    }

    // функция Terminate() необходима, чтобы реализовать интерфейс IExtensionApplication
    public void Terminate()
    {

    }


    //////////////////////////////////////////////////////////
    // ПЕРВЫЙ ВАРИАНТ РЕАЛИЗАЦИИ ПОИСКА (<<MODEL SPACE>>)
    //////////////////////////////////////////////////////////

    // команда для итерации по объектам (подход 1)
    [CommandMethod("Habr_IterateThroughAllObjects_1")]
    public void iterateThroughAllObjects_1()
    {
        // получаем текущую БД
        Database db = HostApplicationServices.WorkingDatabase;

        // начинаем транзакцию
        using (Transaction tr = db.TransactionManager.StartTransaction())
        {
            // получаем ссылку на пространство модели (ModelSpace)
            BlockTableRecord ms =
```

```
(BlockTableRecord)tr.GetObject(SymbolUtilityServices.GetBlockModelSpaceId(
db), OpenMode.ForRead);

                // "пробегаем" по всем объектам в пространстве модели
                foreach (ObjectId id in ms)
                {
                    // приводим каждый из них к типу Entity
                    Entity entity = (Entity)tr.GetObject(id, OpenMode.ForRead);

                    // выводим в консоль слой (entity.Layer), тип
(entity.GetType().ToString()) и цвет (entity.Color) каждого объекта

acad.DocumentManager.MdiActiveDocument.Editor.WriteMessage(string.Format
("\nLayer:{0}; Type:{1}; Color: {2},{3},{4}\n",
                        entity.Layer, entity.GetType().ToString(),
entity.Color.Red.ToString(), entity.Color.Green.ToString(),
entity.Color.Blue.ToString()));
                }

                tr.Commit();
            }
        }

        // команда для поиска окружностей (подход 1)
        [CommandMethod("Habr_FindCircles_1")]
        public void findCircles_1()
        {
            // получаем текущую БД
            Database db = HostApplicationServices.WorkingDatabase;

            // начинаем транзакцию
            using (Transaction tr = db.TransactionManager.StartTransaction())
            {
                // получаем ссылку на пространство модели (ModelSpace)
                BlockTableRecord ms =
(BlockTableRecord)tr.GetObject(SymbolUtilityServices.GetBlockModelSpaceId(
db), OpenMode.ForRead);

                // "пробегаем" по всем объектам в пространстве модели
                foreach (ObjectId id in ms)
                {
                    // приводим каждый из них к типу Entity
                    Entity entity = (Entity)tr.GetObject(id, OpenMode.ForRead);
```

```
                // если это окружность - выводим в консоль слой, тип и цвет каждого объекта

                if (entity.GetType() == typeof(Circle))
                {

acad.DocumentManager.MdiActiveDocument.Editor.WriteMessage(string.Format("\nLayer:{0}; Type:{1}; Color: {2},{3},{4}\n",
                        entity.Layer, entity.GetType().ToString(), entity.Color.Red.ToString(), entity.Color.Green.ToString(), entity.Color.Blue.ToString()));
                }
            }

            tr.Commit();
        }
    }

    // команда для поиска салатовых объектов (подход 1)
    [CommandMethod("Habr_FindLightGreenObjects_1")]
    public void findLightGreenObjects_1()
    {
        // получаем текущую БД
        Database db = HostApplicationServices.WorkingDatabase;

        // начинаем транзакцию
        using (Transaction tr = db.TransactionManager.StartTransaction())
        {
            // получаем ссылку на пространство модели (ModelSpace)
            BlockTableRecord ms = (BlockTableRecord)tr.GetObject(SymbolUtilityServices.GetBlockModelSpaceId(db), OpenMode.ForRead);

            // "пробегаем" по всем объектам в пространстве модели
            foreach (ObjectId id in ms)
            {
                // приводим каждый из них к типу Entity
                Entity entity = (Entity)tr.GetObject(id, OpenMode.ForRead);

                // если цвет объекта - салатовый, то выводим в консоль слой, тип и цвет каждого объекта
                if (entity.Color == Autodesk.AutoCAD.Colors.Color.FromRgb(128, 255, 64))
                {
```

```
acad.DocumentManager.MdiActiveDocument.Editor.WriteMessage(string.Format
("\nLayer:{0}; Type:{1}; Color: {2},{3},{4}\n",
                        entity.Layer, entity.GetType().ToString(),
entity.Color.Red.ToString(), entity.Color.Green.ToString(),
entity.Color.Blue.ToString()));
                    }
                }

                tr.Commit();
            }
        }

        // команда для поиска всех вхождений блока "block-1" (подход 1)
        [CommandMethod("Habr_FindBlocks_1")]
        public void findBlocks_1()
        {
            // получаем текущую БД
            Database db = HostApplicationServices.WorkingDatabase;

            // начинаем транзакцию
            using (Transaction tr = db.TransactionManager.StartTransaction())
            {
                // получаем ссылку на пространство модели (ModelSpace)
                BlockTableRecord ms =
(BlockTableRecord)tr.GetObject(SymbolUtilityServices.GetBlockModelSpaceId(
db), OpenMode.ForRead);

                // "пробегаем" по всем объектам в пространстве модели
                foreach (ObjectId id in ms)
                {
                    // приводим каждый из них к типу Entity
                    Entity entity = (Entity)tr.GetObject(id, OpenMode.ForRead);

                    // проверяем, является ли объект вхождением блока
                    if (entity.GetType() == typeof(BlockReference))
                    {
                        // если является - приводим его к типу BlockReference
                        BlockReference br = (BlockReference)entity;

                        // если имя соответствующего определения блока - "block-1",
то выводим в консоль слой, тип и цвет каждого объекта
                        if (br.Name == "block-1")
                        {
```

```
acad.DocumentManager.MdiActiveDocument.Editor.WriteMessage(string.Format
("\nLayer:{0}; Type:{1}; Color: {2},{3},{4}\n",
                        entity.Layer, entity.GetType().ToString(),
entity.Color.Red.ToString(), entity.Color.Green.ToString(),
entity.Color.Blue.ToString()));
                    }
                }
            }

            tr.Commit();
        }
    }


    //////////////////////////////////////////////////////////
    // ВТОРОЙ ВАРИАНТ РЕАЛИЗАЦИИ ПОИСКА
(<<EDITOR.SELECTALL>>)
    //////////////////////////////////////////////////////////

    // команда для итерации по объектам (подход 2)
    [CommandMethod("Habr_IterateThroughAllObjects_2")]
    public void iterateThroughAllObjects_2()
    {
        // получаем БД и Editor текущего документа
        Document doc = Application.DocumentManager.MdiActiveDocument;
        Database db = doc.Database;
        Editor ed = doc.Editor;

        // пытаемся получить ссылки на все объекты
        // ВНИМАНИЕ! Нужно проверить работоспособность метода с
замороженными и заблокированными слоями!
        PromptSelectionResult selRes = ed.SelectAll();

        // если произошла ошибка - сообщаем о ней
        if (selRes.Status != PromptStatus.OK)
        {
            ed.WriteMessage("\nError!\n");
            return;
        }

        // получаем массив ID объектов
        ObjectId[] ids = selRes.Value.GetObjectIds();
```

```
            // начинаем транзакцию
            using (Transaction tr = db.TransactionManager.StartTransaction())
            {
               // "пробегаем" по всем полученным объектам
               foreach (ObjectId id in ids)
               {
                  // приводим каждый из них к типу Entity
                  Entity entity = (Entity)tr.GetObject(id, OpenMode.ForRead);

                  // выводим в консоль слой (entity.Layer), тип
(entity.GetType().ToString()) и цвет (entity.Color) каждого объекта

acad.DocumentManager.MdiActiveDocument.Editor.WriteMessage(string.Format
("\nLayer:{0}; Type:{1}; Color: {2},{3},{4}\n",
                     entity.Layer, entity.GetType().ToString(),
entity.Color.Red.ToString(), entity.Color.Green.ToString(),
entity.Color.Blue.ToString()));
               }

               tr.Commit();
            }
         }

         // команда для поиска окружностей (подход 2)
         [CommandMethod("Habr_FindCircles_2")]
         public void findCircles_2()
         {
            // получаем БД и Editor текущего документа
            Document doc = Application.DocumentManager.MdiActiveDocument;
            Database db = doc.Database;
            Editor ed = doc.Editor;

            // создаем переменную, в которой будут содержаться данные для
фильтра
            TypedValue[] filterlist = new TypedValue[1];

            // первый аргумент (0) указывает, что мы задаем тип объекта
            // второй аргумент ("CIRCLE") - собственно тип
            filterlist[0] = new TypedValue(0, "CIRCLE");

            // создаем фильтр
            SelectionFilter filter = new SelectionFilter(filterlist);

            // пытаемся получить ссылки на объекты с учетом фильтра
```

```
            // ВНИМАНИЕ! Нужно проверить работоспособность метода с
замороженными и заблокированными слоями!
            PromptSelectionResult selRes = ed.SelectAll(filter);

            // если произошла ошибка - сообщаем о ней
            if (selRes.Status != PromptStatus.OK)
            {
               ed.WriteMessage("\nError!\n");
               return;
            }

            // получаем массив ID объектов
            ObjectId[] ids = selRes.Value.GetObjectIds();

            // начинаем транзакцию
            using (Transaction tr = db.TransactionManager.StartTransaction())
            {
               // "пробегаем" по всем полученным объектам
               foreach (ObjectId id in ids)
               {
                  // приводим каждый из них к типу Entity
                  Entity entity = (Entity)tr.GetObject(id, OpenMode.ForRead);

                  // выводим в консоль слой, тип и цвет каждого объекта

acad.DocumentManager.MdiActiveDocument.Editor.WriteMessage(string.Format
("\nLayer:{0}; Type:{1}; Color: {2},{3},{4}\n",
                        entity.Layer, entity.GetType().ToString(),
entity.Color.Red.ToString(), entity.Color.Green.ToString(),
entity.Color.Blue.ToString()));
               }

               tr.Commit();
            }
         }



         ////////////////////////////////////////////////////////////
         // РЕДАКТИРОВАНИЕ ОБЪЕКТОВ
         ////////////////////////////////////////////////////////////

         // команда для удаления окружностей
         [CommandMethod("Habr_EraseCircles_2")]
```

```
        public void eraseCircles_2()
        {
            // получаем БД и Editor текущего документа
            Document doc = Application.DocumentManager.MdiActiveDocument;
            Database db = doc.Database;
            Editor ed = doc.Editor;

            // создаем переменную, в которой будут содержаться данные для
фильтра
            TypedValue[] filterlist = new TypedValue[1];

            // первый аргумент (0) указывает, что мы задаем тип объекта
            // второй аргумент ("CIRCLE") - собственно тип
            filterlist[0] = new TypedValue(0, "CIRCLE");

            // создаем фильтр
            SelectionFilter filter = new SelectionFilter(filterlist);

            // пытаемся получить ссылки на объекты с учетом фильтра
            // ВНИМАНИЕ! Нужно проверить работоспособность метода с
замороженными и заблокированными слоями!
            PromptSelectionResult selRes = ed.SelectAll(filter);

            // если произошла ошибка - сообщаем о ней
            if (selRes.Status != PromptStatus.OK)
            {
                ed.WriteMessage("\nError!\n");
                return;
            }

            // получаем массив ID объектов
            ObjectId[] ids = selRes.Value.GetObjectIds();

            // начинаем транзакцию
            using (Transaction tr = db.TransactionManager.StartTransaction())
            {
                // "пробегаем" по всем полученным объектам
                foreach (ObjectId id in ids)
                {
                    // приводим каждый из них к типу Entity
                    Entity entity = (Entity)tr.GetObject(id, OpenMode.ForRead);

                    // открываем приговоренный объект на запись
                    entity.UpgradeOpen();
```

```
                // удаляем объект
                entity.Erase();
            }

            tr.Commit();
        }
    }


    // команда для перекраски всех объектов в оранжевый цвет
    [CommandMethod("Habr_RepaintOrange_2")]
    public void repaintOrange_2()
    {
        // получаем БД и Editor текущего документа
        Document doc = Application.DocumentManager.MdiActiveDocument;
        Database db = doc.Database;
        Editor ed = doc.Editor;

        // пытаемся получить ссылки на все объекты
        // ВНИМАНИЕ! Нужно проверить работоспособность метода с замороженными и заблокированными слоями!
        PromptSelectionResult selRes = ed.SelectAll();

        // если произошла ошибка - сообщаем о ней
        if (selRes.Status != PromptStatus.OK)
        {
            ed.WriteMessage("\nError!\n");
            return;
        }

        // получаем массив ID объектов
        ObjectId[] ids = selRes.Value.GetObjectIds();

        // начинаем транзакцию
        using (Transaction tr = db.TransactionManager.StartTransaction())
        {
            // "пробегаем" по всем полученным объектам
            foreach (ObjectId id in ids)
            {
                // приводим каждый из них к типу Entity
                Entity entity = (Entity)tr.GetObject(id, OpenMode.ForRead);

                // открываем объект на запись
```

```
                entity.UpgradeOpen();

                // изменяем цвет на оранжевый
                entity.Color = Autodesk.AutoCAD.Colors.Color.FromRgb(255,
128, 0);
            }

            tr.Commit();
        }
    }


    // команда для изменения радиуса окружностей
    [CommandMethod("Habr_ModifyCircles_2")]
    public void modifyCircles_2()
    {
        // получаем БД и Editor текущего документа
        Document doc = Application.DocumentManager.MdiActiveDocument;
        Database db = doc.Database;
        Editor ed = doc.Editor;

        // создаем переменную, в которой будут содержаться данные для
фильтра
        TypedValue[] filterlist = new TypedValue[1];

        // первый аргумент (0) указывает, что мы задаем тип объекта
        // второй аргумент ("CIRCLE") - собственно тип
        filterlist[0] = new TypedValue(0, "CIRCLE");

        // создаем фильтр
        SelectionFilter filter = new SelectionFilter(filterlist);

        // пытаемся получить ссылки на объекты с учетом фильтра
        // ВНИМАНИЕ! Нужно проверить работоспособность метода с
замороженными и заблокированными слоями!
        PromptSelectionResult selRes = ed.SelectAll(filter);

        // если произошла ошибка - сообщаем о ней
        if (selRes.Status != PromptStatus.OK)
        {
            ed.WriteMessage("\nError!\n");
            return;
        }
```

```
            // получаем массив ID объектов
            ObjectId[] ids = selRes.Value.GetObjectIds();

            // начинаем транзакцию
            using (Transaction tr = db.TransactionManager.StartTransaction())
            {
               // "пробегаем" по всем полученным объектам
               foreach (ObjectId id in ids)
               {
                  // приводим каждый из них к типу Circle
                  Circle cir = (Circle)tr.GetObject(id, OpenMode.ForRead);

                  // открываем объект на запись
                  cir.UpgradeOpen();

                  // увеличиваем радиус
                  cir.Radius = cir.Radius * 2;
               }

               tr.Commit();
            }
         }

         // команда для перемещения блоков
         [CommandMethod("Habr_MoveBlocks_2")]
         public void moveBlocks_2()
         {
            // получаем БД и Editor текущего документа
            Document doc = Application.DocumentManager.MdiActiveDocument;
            Database db = doc.Database;
            Editor ed = doc.Editor;

            // создаем переменную, в которой будут содержаться данные для
фильтра
            TypedValue[] filterlist = new TypedValue[1];

            // первый аргумент (0) указывает, что мы задаем тип объекта
            // второй аргумент ("INSERT") - собственно тип
            filterlist[0] = new TypedValue(0, "INSERT");

            // создаем фильтр
            SelectionFilter filter = new SelectionFilter(filterlist);

            // пытаемся получить ссылки на объекты с учетом фильтра
```

```
            // ВНИМАНИЕ! Нужно проверить работоспособность метода с замороженными и заблокированными слоями!
            PromptSelectionResult selRes = ed.SelectAll(filter);

            // если произошла ошибка - сообщаем о ней
            if (selRes.Status != PromptStatus.OK)
            {
               ed.WriteMessage("\nError!\n");
               return;
            }

            // получаем массив ID объектов
            ObjectId[] ids = selRes.Value.GetObjectIds();

            // начинаем транзакцию
            using (Transaction tr = db.TransactionManager.StartTransaction())
            {
               // "пробегаем" по всем полученным объектам
               foreach (ObjectId id in ids)
               {
                  // приводим каждый из них к типу BlockReference
                  BlockReference br = (BlockReference)tr.GetObject(id, OpenMode.ForRead);

                  // открываем объект на запись
                  br.UpgradeOpen();

                  // перемещаем вхождение блока
                  br.Position = Point3d.Origin;
               }

               tr.Commit();
            }
         }



         ////////////////////////////////////////////////////////////
         // БОНУСНЫЙ ВАРИАНТ РЕАЛИЗАЦИИ ПОИСКА (<<ЧЕРНАЯ МАГИЯ>>)
         ////////////////////////////////////////////////////////////

         // команда для итерации по объектам (подход 3)
         // Осторожно! Используется черная магия!
```

```
        [CommandMethod("Habr_IterateThroughAllObjects_3")]
        public void iterateThroughAllObjects_3()
        {
            Database db = HostApplicationServices.WorkingDatabase;
            Editor ed = Application.DocumentManager.MdiActiveDocument.Editor;
            long amount = 0;
            Dictionary<string, int> d = new Dictionary<string, int>();
            ObjectId id = ObjectId.Null;
            for (long i = db.BlockTableId.Handle.Value; i < db.Handseed.Value;
i++)
            {
                Handle h = new Handle(i);
                if (db.TryGetObjectId(h, out id) && !id.IsNull && id.IsValid &&
!id.IsErased)
                {
                    string t = id.ObjectClass.DxfName;
                    amount++;
                    if (d.ContainsKey(t))
                        d[t]++;
                    else
                        d.Add(t, 1);
                }
            }
            foreach (KeyValuePair<string, int> kvp in d)
                ed.WriteMessage("\n{0}: {1} ", kvp.Key, kvp.Value);
            ed.WriteMessage("\nTotal {0} objects in drawing\n", amount);
        }
    }
  }
}
```