Санкт-Петербургский государственный университет

Математико-механический факультет

Кафедра системного программирования

Разработка интерфейса программирования приложений и добавление скриптовой функциональности для ПО криминалистического анализа

Курсовая работа студента 361 группы

Макеева Давида Александровича

Научный руководитель ст. пр. Губанов Ю.А.

Санкт-Петербург

2012

Оглавление

Введение....3
Постановка задачи....4
Основные понятия....4
Обзор существующих решений....6
Скриптовые решения....6
CS-Script....6
.Net Script Editor (.NET SE)....7
IronPython (IP)....8
Дополнительные инструменты....9
Scintilla.NET....9
Схожие криминалистические продукты....10
EnCase7....10
Итоги....11
Реализация....12
Belkasoft Evidence Center....12
.NET Script Editor....14
Отладчик....16
Заключение....17
Результаты....17
Дальнейшее развитие....17
Список литературы....18

Введение

Многие приложения, обычно рассчитанные на рядовых пользователей, как правило, использующих лишь графический интерфейс, направлены также на программистов, или людей, хотя бы поверхностно знакомых с программированием. Таким пользователям может не хватать стандартных функций, предоставляемых приложением посредством графического интерфейса, им может понадобиться, к примеру, импорт или экспорт данных нестандартных форматов, у них может возникнуть необходимость совершать какие-то нестандартные действия над информацией с помощью третьестороннего ПО. Для решения всех вышеперечисленных задач вместе с приложением часто предоставляется интерфейс программирования приложений (API).

API может быть мощным и коммерчески важным активом компании. Многие клиенты вкладывают значительные средства в покупку ПО и обучение работе с ним, успешные API привлекают и “подсаживают” клиентов. С другой стороны, некачественный API может являться тяжким грузом. Плохой API может привести к большому числу звонков в службу поддержки, препятствовать движению вперед, приводить к потере клиентов. В статье [2] автор считает, что качественный API должен быть:

- Минимальным.
- Полным.
- Иметь прозрачную и простую семантику.
- Быть интуитивно понятным и легко запоминаться.
- Способствовать написанию читаемого кода.

Помимо API, разработчики нередко поддерживают в своих приложениях встроенную скриптовую функциональность (или, вкратце, “скрипты”). Поддержка скриптов дает пользователям еще б*о*льшие возможности и удобства. Скрипты позволяют обращаться к API, добавлять какую-то нестандартную функциональность, использовать глобальные переменные сред приложений, собственно, исполнять и писать сценарии и т.д. Очевидно, что любая качественная скриптовая система, как и любая система разработки, должна обладать большим набором различных инструментов, облегчающих разработку, которые обычно поставляются с различными IDE, например, подсветкой синтаксиса, отладчиком, автодополнением.

Постановка задачи

Belkasoft Evidence Center (в дальнейшем BEC) – мощный криминалистический продукт, разработанный на Microsoft.NET Framework 3.5. Продукт позволяет искать и извлекать истории интернет-пейджеров и браузеров, анализировать удаленные данные, почту и дампы памяти, распознавать лица, порнографию и тексты в изображениях и видео, разбирать сетевой трафик и т.п.

Чтобы пользователи могли совершать какие-либо нестандартные действия с найденной информацией, требуется разработать API, и создать скриптовый язык (желательно C#), работающий внутри продукта. Язык, опираясь на API, мог бы дать пользователю бо́льшую гибкость в работе с продуктом.

Отметим, большую важность визуальной составляющей конечного продукта.

Основные понятия

Так как данная работа имеет тесную связь с технологией Microsoft.NET, необходимо провести краткий экскурс по ней с изложением самых основных понятий и положений.

.NET Framework – программная платформа, разработанная и выпущенная корпорацией Microsoft в 2002 году. Основным компонентом .NET Framework является *общеязыковая исполняющая среда* (common language runtime, CLR). При компиляции кода для .NET, компилятор создает *управляемый модуль* (managed module), который содержит код на *общем промежуточном языке* (common intermediate language, CIL, IL) и метаданные. IL-код зачастую называют *управляемым* (managed code), потому что CLR управляет его жизненным циклом и исполнением. По правде говоря, CLR работает не с модулями, а со *сборками*. Сборка (assembly), во-первых - это логическая группировка одного или нескольких управляемых модулей, или файлов ресурсов; во-вторых - это минимальная единица, с точки зрения, повторного использования, безопасности и управления версиями.[8] Условно, сборки можно разделить на два вида: сборки со строгим именем и обычные сборки. Сборки со строгим именем обычно предназначаются для совместного использования несколькими приложениями. Чтобы совместное использование было возможно, сборку необходимо поместить в общеизвестный каталог. Место, где располагаются совместно используемые сборки, называется - *глобальный кэш сборок* (global assembly cache, GAC). Также кроме сборок Microsoft.NET поддерживает COM-объекты. COM (component object model, объектная модель компонентов) – это технологический стандарт от Microsoft, предназначенный для создания ПО на основе

взаимодействующих компонентов, каждый из которых может использоваться во многих программах одновременно.

В .NET существует такое понятие, как *домен приложения* (application domain, AppDomain) – логический контейнер набора сборок. Основная задача домена приложения – обеспечить изоляцию. Домены приложения удобны благодаря следующим свойствам:

- Объекты созданные одним AppDomain не видят другие домены приложения.
- Домены приложения можно выгружать.
- Домены приложений можно защищать по отдельности.
- Домены приложения можно сконфигурировать по отдельности.

Изначально CLR поддерживала исключительно статическую типизацию. Однако, в четвертой версии .NET Framework появилась *исполняющая среда динамического языка* (dynamic language runtime, DLR). DLR представляет собой набор служб, добавленных к CLR, которые позволяют обрабатывать код языков с динамической типизацией.[9] Доступна версия DLR с открытым исходным кодом.[10]

Обзор существующих решений

Скриптовые решения

Существуют следующие решения, добавляющие скриптовую функциональность в .NET приложения.

CS-Script

Cs-Script – это скриптовая система, использующая C# в качестве языка программирования. CS-Script является бесплатным проектом с открытым исходным кодом [1]. Основным элементом CS-Script является скриптовый “движок”.

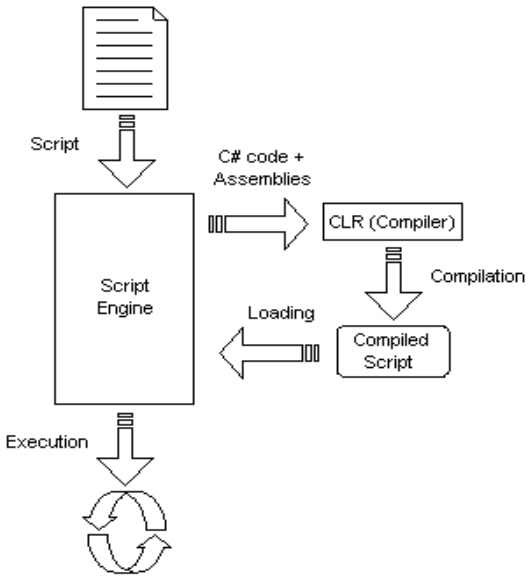

Рис. 1: Схема исполнения скрипта в CS-Script

Исполнение скрипта выглядит так: пользователь запускает приложение со встроенным скриптовым движком и указывает в параметрах командной строки скрипт для выполнения. Скрипт – это файл, с кодом на C#, который должен иметь статический метод Main. Движок компилирует скрипт в сборку и исполняет её. CLR является основный частью этого процесса - она выполняет компиляцию скрипта, а также IL-кода.

CS-Script предоставляет возможность передавать в скрипт из хост-приложения (т.е. приложения, в которое встроена скриптовая система), объекты любого типа, правда, только в том случае, если хост-сборка зарегистрирована в GAC.

К недостаткам CS-Script можно отнести:

- Отсутствие удобного пользовательского интерфейса.
- Отсутствие автодополнения.
- Отсутствие отладчика.

.NET Script Editor (.NET SE)

.NET SE реализует схожую с CS-Script функциональность - позволяет поддерживать скрипты в приложениях, добавлять объекты хост-приложения в скрипт (при помощи метода *AddObject*). Аналогичен и процесс исполнения скриптов. [3]

Из преимуществ стоит отметить:

- Удобный текстовой редактор.
- Синтаксический анализ без компиляции.
- Поддержка COM.
- Поддержка веб-ссылок.
- Присутствует подсветка синтаксиса
- Автодополнение
- Возможность ставить точки останова.

.NET SE является, своего рода, мини-IDE, реализованной в одном элементе управления Windows Forms.

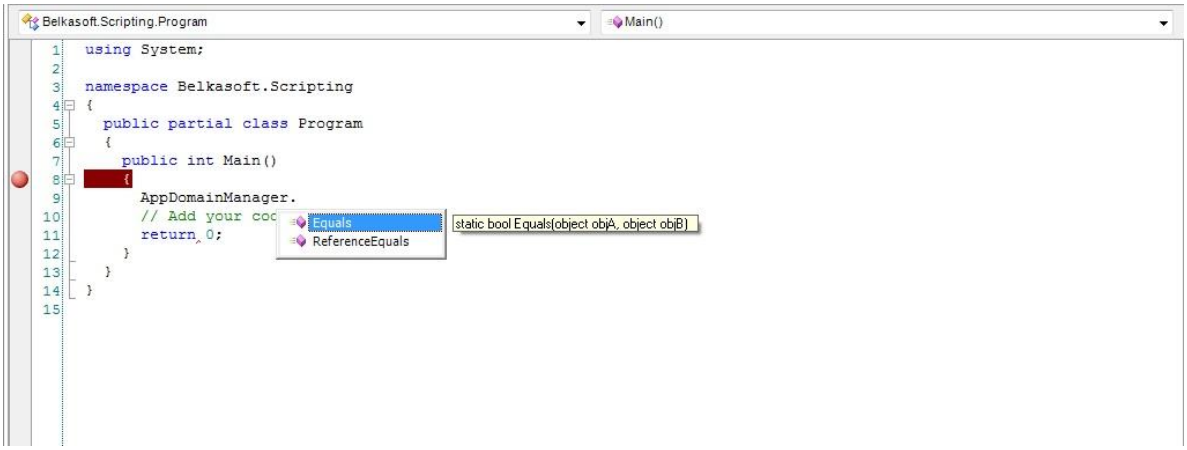

Рис.2: Возможности .NET SE

К минусам .NET SE можно отнести:

- Отсутствие отладчика.
- Общая недоработанность проекта.
- Большое количество различных ошибок.

IronPython (IP)

Свободная реализация популярного языка Python под .NET [4], [5]. Для IP реализована полная интеграция с .NET Framework, что позволяет поддерживать все возможности .NET в IP. Отметим, что это стало возможно только с появлением полноценной динамической типизации и DLR.

Существуют следующие инструменты, облегчающие разработку на IP:

IronPython Studio (IP Studio).

IP Studio – полноценная IDE для разработки на IP, может также использоваться в качестве расширения для Visual Studio. Проект с открытым исходным кодом, распространяемый по лицензии Microsoft Public License. Так как IP Studio базируется на VS 2008 Shell, для ее работы требуется либо установленная Visual Studio, либо VS 2008 Shell. IP Studio поддерживает все возможности Visual Studio.

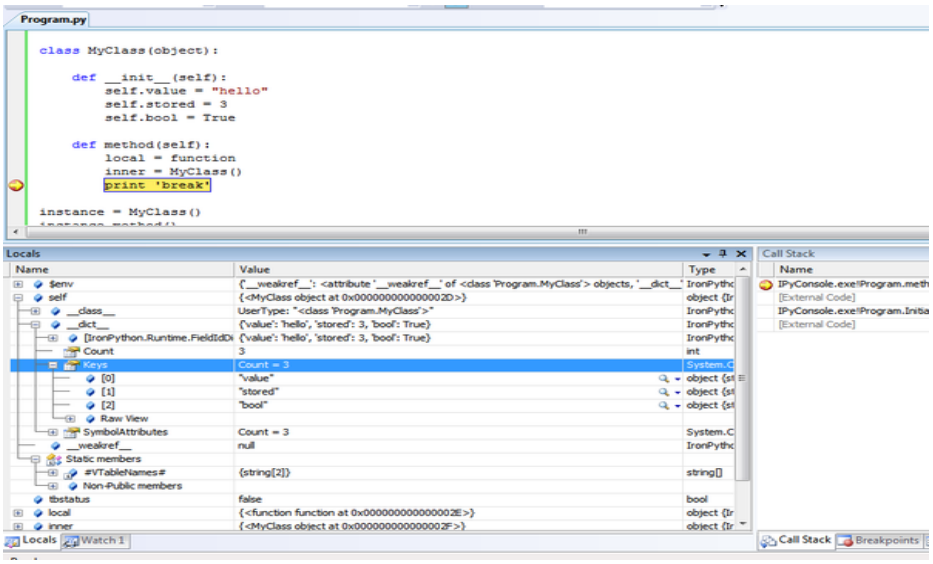

Рис. 3: Пример отладки в IronPython Studio

Помимо вышеперечисленных решений, были также исследованы: SharpDevelop, Lua, Boo, но их обзор проводиться не будет.

Дополнительные инструменты

Следующие инструменты могли бы помочь для реализации графического интерфейса для скриптовых систем.

Scintilla.NET

Scintilla .NET – мощный текстовой редактор, реализованный как элемент управления Windows Forms, представляющий управляемую версию текстового редактора Scintilla[6]. Из возможностей отметим:

- Подсветка синтаксиса. Поддержано более 80 языков.
- Сворачивание и разворачивание блоков.
- Возможность ставить закладки и точки останова.
- Поддержка автодополнения.
- Проверка правильности скобочных последовательностей.

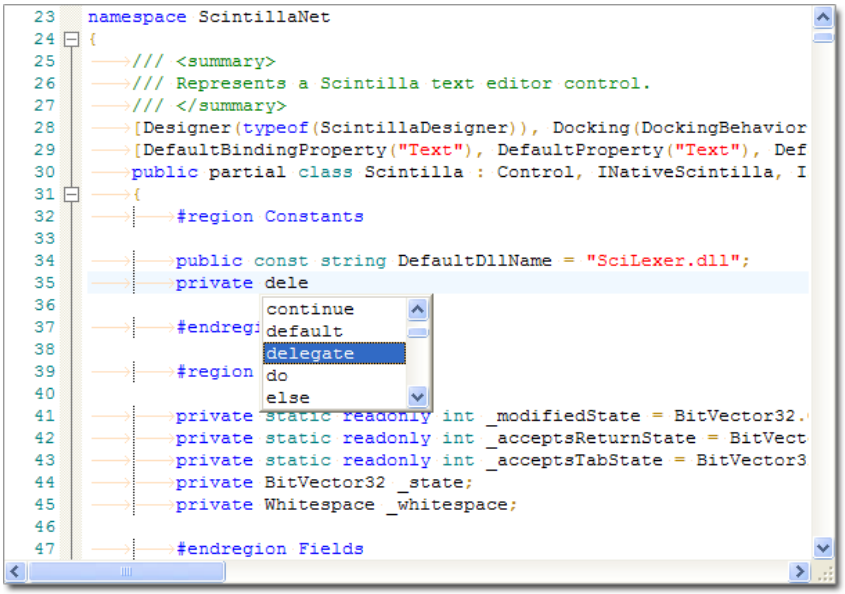

Рис. 4: Scintilla.Net

Минусом является практически полное отсутствие документации.

Схожие криминалистические продукты

Существуют следующие продукты цифровой криминалистики, предоставляющие пользователям API и скриптовые возможности.

EnCase7

EnCase7 - это криминалистический продукт, предназначенный для операционных систем семейства Windows. Продукт поддерживает скриптовую функциональность, и имеет публичный API [7].

В качестве скриптового языка - свой собственный язык, похожий на C++. Язык и среда выполнения позволяют работать со всеми сущностями EnCase7: получать доступ к файловой системе, на локальной или удаленной машине, взаимодействовать с web-сервисами для интеграции со сторонними инструментами, расширять пункты контекстного меню. Для скриптов предусмотрена возможность отладки.

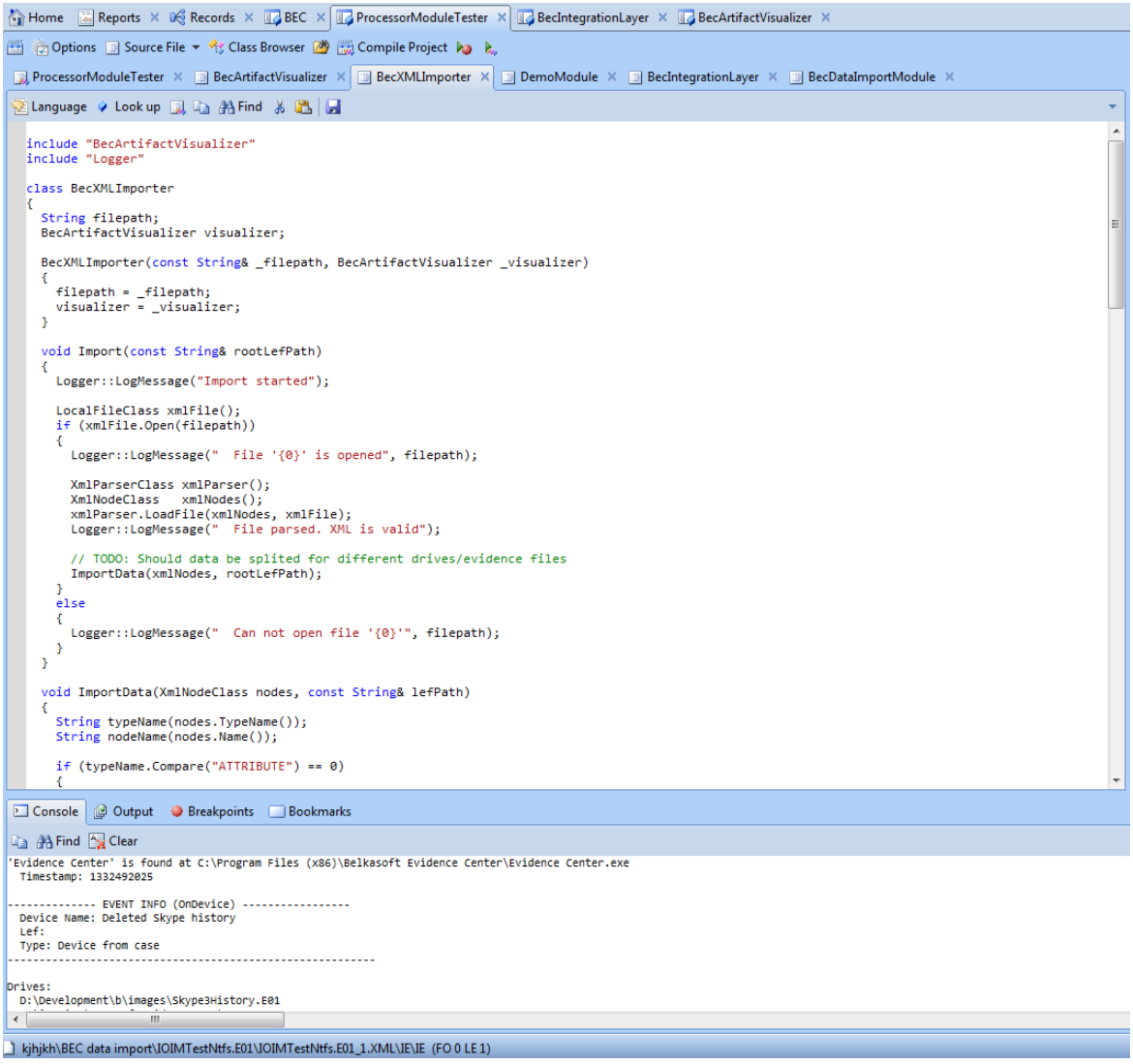

Рис. 5: Скриптовая система в EnCase7.

Из недостатков EnScript стоит отметить:

- Документация к продукту скудная, примеров крайне мало.
- Временами некорректное поведение скриптов.
- Отсутствие механизма обработки исключительных ситуаций.
- Некорректное сохранение скриптовых проектов.
- Отсутствие автодополнения.

Итоги

После обзора решений стало ясно, что интерес для данной работы представляют только .NET SE и IronPython. Для поддержки скриптовой функциональности в BEC было принято решение использовать .NET SE. Решение аргументировано следующими пунктами:

- Постановка задачи. Желательно было использовать C# в качестве скриптового языка.
- Отсутствие свободных графических оболочек для IP.
- Бо́льшая часть требуемого функционала в .NET SE уже была реализована.
- Использовать .NET SE - более рациональный подход

Реализация

Реализацию данной работы можно условно разделить на три части:

- Разработка API для BEC.
- Добавление скриптовой функциональности в BEC.
- Реализация отладчика скриптов.

Belkasoft Evidence Center

Для первой версии реализации API была выбрана самая “легкая” версия BEC – Forensic IM Analyzer. Эта версия поддерживает анализ истории более семидесяти систем мгновенного обмена сообщениями, в том числе таких популярных, как Skype, ICQ, Miranda, QIP, в операционных системах Windows, Linux и MacOS X, интеграцию с EnCase7.

Главной сущностью BEC является *дело* (Case). Дело содержит в себе *профили* (Profiles), *закладки* (Bookmarks) и информацию о себе. Профиль, неформально говоря, это учетная запись системы мгновенного обмена сообщениями. В нем хранится история учетной записи, её артефакты, и дополнительная информация о профиле. Закладки содержат ссылки на артефакты. Эти сущности были реализованы в трех классах: *Case*, *Profile* и *Bookmark*. Класс *Case* предоставляет возможность искать профили, извлекать их истории, проводить поиск в истории, создавать закладки. Классы *Bookmark* и *Profile* позволяют проводить поиск в истории и экспортировать её. Более подробно с описываемым функционалом можно познакомиться на следующем рисунке.

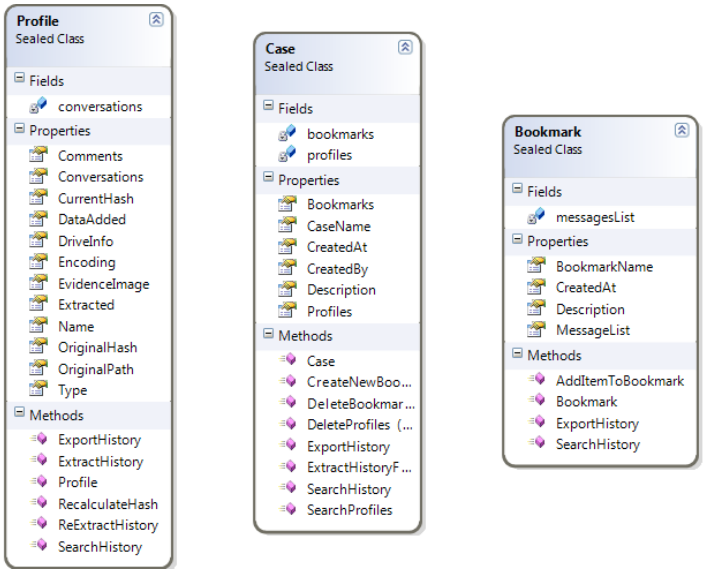

Рис. 6: Диаграммы классов Profile, Case и Bookmark

Помимо вышеперечисленных классов, были разработаны классы, вспомогательные для поиска и экспортирования информации и профилей: *SearchHistoryOptions*, *SearchProfileOptions* и *ReportOptions*. Эти классы предоставляют обширные опции для вышеперечисленных операций. К примеру, чтобы найти информацию в неком профиле, нужно вызвать метод *SearchHistory* у объекта класса *Profile* и передать ему в качестве параметра объект класса *SearchHistoryOptions*. На диаграмме ниже предоставлена более подробная информация об опциях.

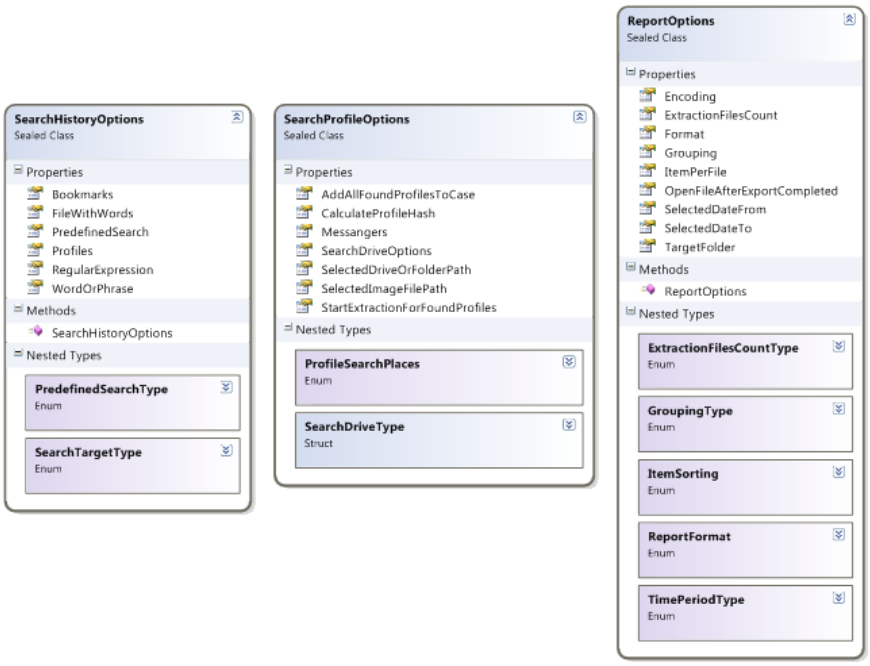

Рис. 7: Классы опций.

Это основные классы API, покрывающие полную функциональность приложения.

.NET Script Editor

Кратко, .NET SE можно представить следующими компонентами:

1. Script Control.
   Обеспечивает поддержку всей функциональности в одном элементе управления и предоставляет графический интерфейс пользователя. Здесь реализована поддержка автодополнения, синтаксического анализа до компиляции,

2. Script Engine.
   Эта компонента исполняет сборку, скомпилированную из исходного кода скриптов, в отдельном домене приложения.

3. Script Run.
   Предоставляет интерфейс *IRun* и класс *ScriptInstance*, унаследованный от класса *MarshalByRefObject*. Используется для предоставления связи между разными доменами приложения.

4. Code Editor.
   Редактор кода. Реализует подсветку синтаксиса.

Любой скрипт в .NET SE должен иметь *partial* класс *Program* и этот класс должен предоставлять метод *static void Main*. С метода *Main* начинается исполнение скрипта. Необходимость того, что класс Program должен быть *partial* обусловлена тем, как реализовано исполнение скриптов в .NET SE.

Для каждого пользовательского скрипта, .NET SE генерирует вторую часть *partial* класса *Program*, скрытую от пользователя, в файле *Program.Sys.cs*. В этом файле находится код на C#. Код указывает, что класс *Program* является наследником класса *MarshalByRefObject* и реализует интерфейс *IRun*. Реализация *IRun* позволяет инициализировать скрипт глобальными переменными приложения, запускать скрипт и освобождать ресурсы. *IRun* содержит следующие методы:

- *void Initialize(IDictionary<string, object> variables)*
- *void Run(string startMethod, params object[] parameters)*
- *void Dispose(IDictionary<string, object> variables)*

Наследование от класса *MarshalByRefObject* позволяет передавать класс *Program* между разными доменами приложения. Передача глобальных объектов в скрипт реализована при помощи метода *AddObject(string name, object variable).*

Так как скрипт и хост-приложение находилится в разных доменах приложения, то любой тип объекта переданного в скрипт из приложения необходимо наследовать от *MarshalByRefObject* и помечать атрибутом *[Serializable()]*. Причем, не только тип объекта, но и все типы, который использовал когда-либо этот объект. Очевидно, что надо

отказываться от подхода с использованием сериализации. Чтобы было понятно как это сделать нужно знать, как исполняется скрипт в .NET SE:

- Пользователь пишет скрипт и компилирует его. Выходными данными компилятора является сборка.
- Сборка загружается в новый домен приложения (отличный от главного домена).
- У класса *Program* сборки последовательно вызываются методы *Initialize*, *Run* и *Dispose*.
- Домен приложения, в котором исполнялась сборка, уничтожается.

Просто так отказаться от сериализации и исполнять скрипты в одном домене с приложением нельзя, потому что .NET не позволяет выгружать сборки из домена. Приходиться версионировать каждую скомпилированную сборку и хранить все эти сборки до завершения приложения.

В .NET SE была добавлена поддержка скриптовых проектов. Проекты хранятся в формате XML и содержат информацию об исходном коде проекта и ссылках на сборки. Проекты можно открывать, сохранять, создавать новые. .NET SE был отлажен, протестирован и глобализован. Проведен общий рефакторинг и реструктурирование исходного кода. Стоит отметить, что исходный код .NET SE оставлял желать много лучшего.

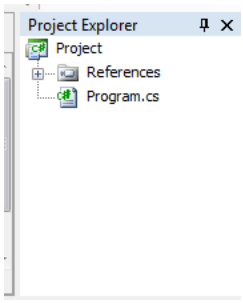

Рис. 8: Обозреватель проектов.

Отладчик

Ни в одном из исследованных решений не было возможности отладки, поэтому отладчик был реализован самостоятельно. Перейдем к описанию реализации.

.NET предоставляет неуправляемый API для отладки, ознакомиться с которым можно здесь.[12] Самые основные задачи сеанса отладки, такие как создание нового процесса или подключение к уже существующему, реализованы с помощью интерфейса *ICorDebug*, при помощи содержит методов:

```
HRESULT CreateProcess (
        [in]  LPCWSTR                  lpApplicationName,
        [in]  LPWSTR                   lpCommandLine,
        [in]  LPSECURITY_ATTRIBUTES    lpProcessAttributes,
        [in]  LPSECURITY_ATTRIBUTES    lpThreadAttributes,
        [in]  BOOL                     bInheritHandles,
        [in]  DWORD                    dwCreationFlags,
        [in]  PVOID                    lpEnvironment,
        [in]  LPCWSTR                  lpCurrentDirectory,
        [in]  LPSTARTUPINFOW           lpStartupInfo,
        [in]  LPPROCESS_INFORMATION    lpProcessInformation,
        [in]  CorDebugCreateProcessFlags  debuggingFlags,
        [out] ICorDebugProcess         **ppProcess);

HRESULT DebugActiveProcess (
        [in] DWORD                     id,
        [in] BOOL                      win32Attach,
        [out] ICorDebugProcess         **ppProcess);
```

Для *ICorDebug* создана COM-обертка при помощи класса *Wrapper*.

Для установки точек останова на функции необходимо вызвать метод *CreateBreakpoint* интерфейса *ICorDebugFunction*, а затем активировать созданную точку останова с помощью метода *Activate* интерфейса *IDebugBreakPoint*. Трудность в том, как найти нужный *ICorDebugFunction*, например, на основе строки предоставленной пользователем. Для того чтобы узнать к какому типу относится метод, необходимо знать метаинформацию модуля в котором метод находится. Чтобы устанавливать, точки останова на определенную строку в файле, нужно найти модуль, который был построен из данного файла. Мы перебираем все модули, и проверяем у каждого их ссылки на совпадение с именем нужного файла.

Заключение

Результаты

На текущий момент разработан API для BEC, покрывающий всю требуемую функциональность продукта.

Встроена и дописана полноценная скриптовая система. При этом учтены недостатки рассмотренных существующих средств, а их преимущества, такие как отладчик, если не реализованы, будут добавлены в будущем без затраты лишних усилий, так как работа практически завершена.

Реализован отладчик, позволяющий отлаживать скрипты.

Дальнейшее развитие

В дальнейшем планируется продолжить данную разработку в следующих направлениях:

- Распространение API на все версии приложения.
- Улучшение пользовательского интерфейса. В контексте скриптовой системы – это, прежде всего наглядный и удобный пользовательский интерфейс отладчика.
- Создание SDK на основе API.
- Создание скриптовых библиотек.

Список литературы

1. Сайт CsScript:
   http://www.csscript.net/
2. Статья *Qt Quarterly 13 article about API design*:
   http://doc.qt.nokia.com/qq/qq13-apis.html
3. .NET Script Editor на CodeProject:
   http://www.codeproject.com/Articles/27744/Net-Script-Editor-C-Vb-net-Mini-IDE
4. Сайт IronPython:
   http://ironpython.net/
5. IronPython на CodePlex:
   http://ironpython.codeplex.com/
6. Scintilla.NET на CodePlex:
   http://scintillanet.codeplex.com/
7. Сайт GuidanceSoftware, разработчика EnCase7:
   http://www.guidancesoftware.com/computer-forensics-fraud-investigation-software.htm
8. Джеффри Рихтер, CLR via C#,
9. Нейгел К., Ивьен Б., Глинн Дж., Уотсон К. - C# 4.0 и платформа .NET 4 для профессионалов, 2011
10. DLR на CodePlex:
    http://dlr.codeplex.com/
11. IronPython Studio на CodePlex:
    http://ironpythonstudio.codeplex.com/
12. Неуправляемый API для отладки:
    http://msdn.microsoft.com/ru-ru/library/ms404484