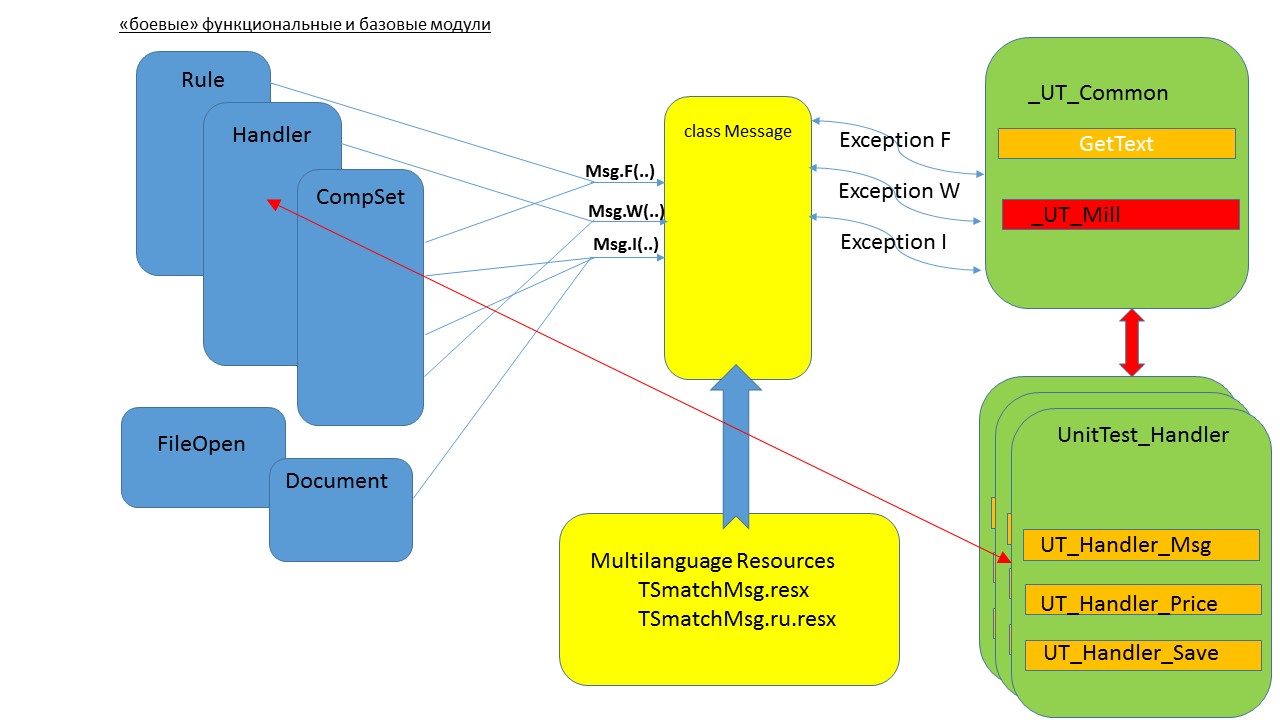
# Описание архитектуры работы с сообщениями Message в текущем состоянии TSmatch v1.0.2 8.10.2017

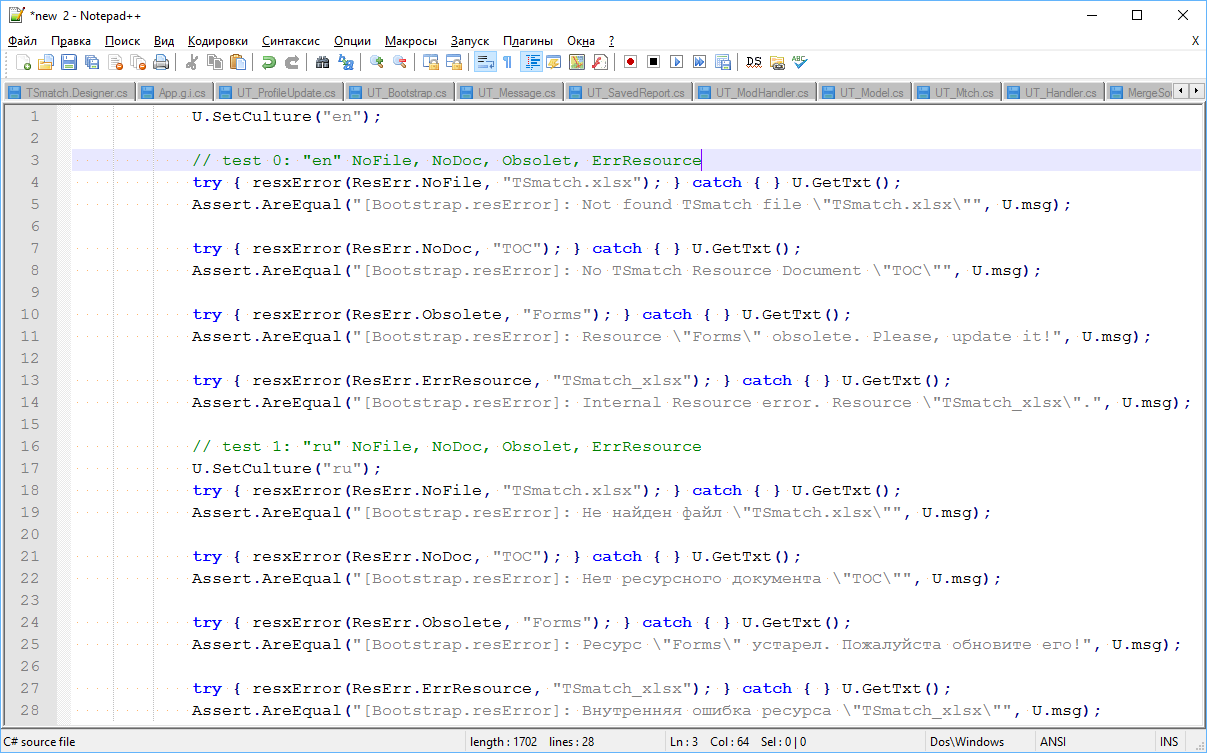
Назначение – поддержка диагностических и информационных сообщений в различных модулях системы TSmatch. Функциональные возможности:

1. **Логирование**: Обеспечена запись в Log-файл TSmatch\_Log.txt через пакет ***log4net***. Настройка конфигурации логирования в ***App.confid***. При тестировании Log-файлы не пишутся.
2. **Multilanguage support:** сейчас работают языка русский и английский языки, но налаженная система редактирования ресурсных файлов **TSmatchMsg.resx** позволяет легко добавить любой другой язык и настраивать TSmatch на язык Windows при запуске. Unit Test’ы проводятся только для русского и английского языков.
3. **Архитектура:** Диаграмма ниже показывает общую схему работы:
   1. Функциональные и базовые модули выдают различные сообщения с параметрами: Msg.F(“text”,arg1, arg2), Msg.W(“предупреждение”, file5), Msg.i(“MD5=”, md5)
   2. Фатальные сообщения в рабочем режиме TSmatch приводят к выводу сообщения об ошибке в MessageBox, затем к остановке приложения. При прогоне Unit Test’ов, остановка недопустима, поэтому модуль Message вместо остановки генерирует Exception F. Сообщения- предупреждения и информационные сообщения тоже порождают Exception W и Exception I, но после их проверки, управление надо вернуть в тестируемый модуль.
   3. Общие для всех Unit Test’ов функции выполняет модуль \_UT\_Common. Этот модуль, являясь наследником класса Message, позволяет при помощи метода GetText, получить диагностический текст и данные параметров сообщения.



1. Красный метод \_UT\_Mill мне написать не удалось. Его задача – перехватывать Exception F, как это делается сейчас в тексте ниже, а при необходимости вернуться в тестируемый модуль после диагностического сообщения Msg.W или Msg.I, просто складывать текст сообщения в стек и возвращать управление. Скорее всего, тут вообще не надо выдавать Exception, а обращаться из Message к \_UT\_Common.\_UT\_Mill как к подпрограмме!

Сейчас проверка всех сообщений рабочих модулей выглядит так:



И это только по Msg.F по единственному модулю и лишь по двум языкам! Приходится много раз повторять очень похожие обращения к тестируемым методам.

Если try-catch не делать, то я, глядя на вид MessageBox, проверяю и настраиваю многострочные сообщения. Но это разовая работа, в Unit Test’ах я комментирую флаг Dialog=false:

//чтобы посмотреть, как выглядит MessageBox, но с Assert.Fault по Msg.FOK(), используй

//var U = new UT\_TSmatch.\_UT\_Msg(true);

var U = new UT\_TSmatch.\_UT\_Msg();

Полагаю, теперь понятно, почему передавая в красный метод \_UT\_Mill я попытался передать имя тестируемого метода через делегата ***Funk<..., string>****,* но не справился.

1. **Даты модификации**: В TSmatch есть несколько мест, где делаются изменения:
   1. Код C#
   2. Ресурсные файлы TSmatch.resx, TSmatchMsg.resx и TSmatchMsg.ru.resx и все последующие
   3. Диспетчерский файл Excel TSmatch.xlsx и указанные в нем файлы- прайс-листы.

Заметная часть ошибок, с которыми я сталкивался – несоответствие изменений содержимого этих источником между собой, когда не во всех нужных местах изменения отражены, и они соответствуют друг другу.   
Сейчас это выявляется при прогоне Unit Test’ов, хотя в TSmatch.resx для форм документов предусмотрен атрибут Date:<дата и время> и Bootstrap при старте TSmatch проверяет, что эти данные не устарели.

Нужно ли проверять остальные источники? Или прогон Unit Test’ов – исчерпывающая проверка?