**Описание**.

# Основа архитектуры

Приложение основано на компонентной системе. И состоит из следующих частей:

* Ядро;
* Общая часть (*Common*). Содержит общие для данного приложения типы, интерфейсы и идентификаторы;
* Фасады;
* Динамические компоненты.

Суть этой системы заключается в том, что загружаемые приложением компоненты, а именно конкретные классы этих компонентов, регистрируются в реестре ядра под индивидуальными идентификаторами. Это могут быть синглтоны, которые можно запросить у ядра по идентификатору или фабричные классы, которые можно так же по идентификатору создать. В реестре классы так же группируются по категориям. Что позволяет осуществить дополнительный удобный функционал внутри категории, например перебор элементов категории. Обращение к компонентам системы, кроме ядра и общих компонентов, **строго через интерфейсы.**

Таким образом, сохраняется анонимность **динамических компонентов,** а взаимодействие с их классами происходит через интерфейс, не привязываясь к реализации. Это позволяет оперативно, фактически *run-time*, конфигурировать приложение, а так же обеспечивает его гибкость и расширяемость. Структурная схема такой компонентной системы выгладит следующим образом.

Динамический компонент …

Динамический компонент **N**

Динамический компонент 1

Фасад **N**

Фасад …

Фасад 1

**ЯДРО**

Common-компонент **N**

Common-компонент …

Common-компонент1

Хранитель настроек (SettingsProvider)

Переводчик идентификаторов (IDTranslator)

Загрузчик (Loader)

Реестр (Register)

**ЯДРО**

Фасад (CoreFacade)

Регистрация динамических компонентов реализована в части ядра, называемой **Реестр**(*Register*). Реестр организует соответствие объектов динамических компонентов и фабричных классов уникальным идентификаторам. И размещению их по категориям. Дерево реестра ядра с зарегистрированными элементами выглядит следующим образом.

**Реестр**

Категория фабричных классов

Категория N

Категория…

Категория 1

Тип **N**

Тип …

Тип 1

Объект 1

Объект …

Объект **N**

Категория – это некий раздел реестра, объединяющий свои элементы по каким-либо общим свойствам или назначению. Один объект может быть зарегистрирован одновременно в нескольких категориях, как отвечающий нескольким группам требований или реализующий комбинированный функционал. Категория фабричных классов объединяет типы объектов, которые можно создавать по средствам ядра, зная только его идентификатор.

Для загрузки динамических компонентов используется модуль ядра **Загрузчик** (*Loader*). Он обеспечивает выполнение всех необходимых действий для подключения модуля к системе.

**Хранитель настроек** (*SettingsProvider*) предназначен для централизованного доступа к конфигурационным данным и информации о кастомной настройке приложения и модулей.

**Переводчик идентификаторов** (*IDTranslator*) – модуль, предназначенный для конвертирования идентификаторов в читабельные и понятные человеку строковые описания или названия элементов, зарегистрированных под ними в реестре ядра.

**Фасад** (*CoreFacade*) – фасад ядра. Позволяет централизовано работать с элементами ядра.

# Common

Компонент, содержащий описание типов и идентификаторов, общих для всех компонентов и специфичных для данного приложения.

## Интерфейсы

## IPlugin

Интерфейс, реализовывать который должен каждый подключаемый класс, объект которого в данной системе является синглтоном. Через этот интерфейс осуществляется инициализация объектов и их деактивация.

## IProject

Описывает функционал класса проекта. Проекты являются фабричными классами, так как могут существовать в нескольких экземплярах.

## IProjectKeeper

Интерфейс класса, предоставляющего дополнительный функционал, для работы с типом проекта. Восновном используется для получения дополнительной информации о типе проекта. Киперы проектов являются синглтонами, так как предоставляют возможность взаимодействия с данными, относящимися к типу проекта.

## IProjectElement

Интерфейс элемента проекта. Всё, из чего состоит проект – каждая его часть и подчасть должна реализовывать этот интерфейс.

## IVisualContainer

Интерфейс визуального компонента, осуществляющего отображение визуальных подключаемых модулей.

## ILogSender

Классы, которые реализуют этот интерфейс, могут инициировать событие при необходимости вывода служебных сообщений. Например, для ведения логов.

## Вспомогательные классы

## XMLHelper

Этот класс позволяет более удобно работать с XML-классами и конструировать XML-документы в контексте данного приложения.

## XmlProject

Расширение XMLHelper'а для более удобной работы с XML-сериализованными проектами.

## StringConverter

Вспомогательный переводчик некоторых типов в строку и из строки.

## ElementTypeConvertor

Класс конвертирования имён элементов в идентификаторы и применяется для XML-десериализации. Для того чтобы этот класс знал о нужном вам соответствии, необходимо при загрузке модуля добавить в него необходимые соответствия. Они хранятся в словаре, зарегистрированном в категории CATID\_COMMON под идентификатором CLSID\_PROJECT\_ELEMENT\_DICTIONARY.

## ControlTypes

Перечисление типов элементов интерфейса. Пример использования: визуальный компонент вызывает у плагина через интерфейс IPlugin функцию

ControlTypes CreateControl(ref Control \_control);

При этом функцией возвращается тип созданного элемента управления. Так как у плагина может быть несколько независимых элементов управления, функция вызывается до тех пор, пока не будет возвращено значение ControlTypes.None.

## Align, TextAlign,…

Перечисления выравниваний и их комбинаций

## Идентификаторы

## Категории

Описываются структурой следующего вида:

public struct CATID\_NAME

{

public const string Literal = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";

public static Guid ID = new Guid(Literal);

}

В Common-модуле описаны следующие категории:

* CATID\_COMMON - общая категория для индивидуальных синглтонов;
* CATID\_PROJECT\_KEEPERS - киперы проектов;
* CATID\_PROJECT\_PLUGINS - плагины, сипользуемые только в проектах, т.е. инициализируются при создании/открытии проекта и деинициализируются при закрытии (например, панель редактирования);
* CATID\_STARTUP\_PLUGINS - плагины, используемые на протяжении всей работы программы(например, главная панель приложения).

Инстанции

Описываются структурой следующего вида:

public struct CLSID\_NAME

{

public const string Literal = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";

public static Guid ID = new Guid(Literal);

}

В Common-модуле описаны следующие идентифиикаторы инстанций:

* CLSID\_PROJECT\_ELEMENT\_DICTIONARY - словарь отношений имени элементов проекта к идентификаторам (используется вспомогательным классом ElementTypeConvertor);
* CLSID\_CURRENT\_PROJECT - текущий открытый проект;
* CLSID\_VISUAL\_CONTAINER - модуль визуализации.