Создание Дополнения CAM-системы

Руководство по созданию Дополнения CAM-системы: интеграция CAD-систем (а также иных источников геометрии) с CAM-системой и перенос геометрии через SGF API.

Содержание

1. Введение
    - Глоссарий
    - Требования
2. Устройство Панели инструментов
    - Описание компонентов
        - Мастер дополнений
        - Файлы дистрибуции
        - Транслятор
        - Файл XML дополнения
        - Файл CAMInfo.ini
        - Утилита CheckLockToolbar
        - Панель инструментов
    - Сценарии взаимодействия
        - Установка Дополнения
        - Удаление Дополнения
        - Импорт из CAD-системы в CAM-систему
        - Импорт CAD-файла в CAM-систему
3. Импорт геометрии в CAM
    - Структура CAD-модели
    - Интерфейсы импорта
    - Подключение STGeomFile.dll
    - Формирование файла SGF
    - Структуры данных SGF
    - Работа со сборкой
    - Работа с деталью
    - Кривые и точки
        - Точка и аналитические кривые
            - 3D-точка
            - Система координат
            - Отрезок
            - Окружность
            - Дуга окружности
            - Эллипс
            - Дуга эллипса

        - Коническая спираль
        - NURBS-кривая
        - Составная кривая
    - Производные кривые
        - Кривая на поверхности
        - Обрезанная кривая
        - Инвертированная кривая
        - Кривая пересечения
        - Граница скругления
    - Параметризация кривых
- Грань (обрезанная поверхность)
- Поверхности
    - Аналитические поверхности
        - Плоскость
        - Цилиндр
        - Конус
        - Сфера
        - Тор
        - NURBS-поверхность
        - Полигональная сетка
    - Производные поверхности
        - Поверхность вращения
        - Поверхность выдавливания
        - Транспонированная UV-поверхность
        - Поверхность скругления
        - Эквидистантная поверхность
    - Параметризация поверхностей
- PMI (Product Manufacturing Information)
    - IPMIGeomThread
- Устаревшие и неиспользуемые методы

Введение

Документ описывает архитектуру и протоколы взаимодействия Дополнений CAM-системы с CAD-системами. По этой инструкции разработчик может реализовать Дополнение для любой CAD-системы, которая удовлетворяет минимальным требованиям к API.

Документ не привязан к конкретной CAD-системе и конкретному языку программирования. Все примеры формата файлов и контрактов взяты из реальных Дополнений (КОМПАС-3D, SolidWorks и др.) и приведены в обобщённом виде.

Основное содержание документа — две большие главы:

- Устройство Панели инструментов — из каких компонентов состоит Дополнение (Транслятор, Панель инструментов и сопутствующие файлы), как они связаны между собой и в каких сценариях взаимодействуют: установка и удаление Дополнения, а также два пути переноса геометрии в CAM-систему.
- Импорт геометрии в CAM — ядро Дополнения: как через API библиотеки `STGeomFile.dll` сформировать файл внутреннего формата CAM-системы (SGF), перенеся в него топологию и геометрию модели — тела, грани, поверхности, кривые и PMI.

Хотя источником геометрии в примерах выступает CAD-система, сам импорт геометрии к ней не привязан. CAD-систему можно заменить любым другим источником — например, нейтральным форматом обмена (STEP, IGES, Parasolid и т.п.): изменится лишь способ получения исходной геометрии, а её запись в SGF останется неизменной. Поэтому главу «Импорт геометрии в CAM» можно использовать и как самостоятельное руководство по конвертации в SGF из произвольного источника.

Глоссарий

Термин	Значение
CAD-система	Приложение, в которое встраивается Панель инструментов. Минимально должно позволять регистрировать сторонние плагины, давать им доступ к текущей открытой модели и предоставлять способ экспорта геометрии (через свой API или встроенный экспорт в STEP/Parasolid).
CAM-система	Потребитель геометрии. Загружает конвертированную геометрию CAD-системы.
Дополнение	Цельный пакет интеграции CAD-системы с CAM-системой. Состоит из Транслятора и Панели инструментов.
Транслятор (`Translator`)	EXE-часть Дополнения. Выполняет установку/удаление Дополнения, а также по запросу от CAM-системы импортирует CAD-файл в её внутренний формат.
Панель инструментов (`Toolbar`)	Часть Дополнения (обычно библиотека), встроенная в CAD-систему как кнопка или панель кнопок. По нажатию конвертирует и передаёт текущую геометрию в CAM-систему.
Мастер дополнений (`AddinManager.dll`)	UI-инструмент CAM-системы для установки/удаления Дополнений. Управляющие команды отправляет Транслятору.
`HostApplication`	Строковый идентификатор CAD-системы (одноимённый тег в Файле XML дополнения). По нему CAM-система находит нужное Дополнение при передаче геометрии из Панели инструментов.
Файл XML дополнения (`<Translator>.xml`)	XML-файл описывает метаданные Дополнения, поддерживаемые форматы и

Термин	Значение
	команды.
`CAMInfo.ini`	Конфигурационный файл с некоторыми параметрами CAM-системы, автоматически создаётся Мастером дополнений.
CheckLockToolbar (`CheckLockToolbar.exe`)	Вспомогательная утилита: проверяет, не блокирует ли процесс CAD-системы установку/удаление.
STGeomFile.dll	Библиотека, предоставляющая доступ к интерфейсу `ISTGeomFiler` — он содержит методы для конвертирования геометрии во внутренний формат CAM-системы (SGF).
SGF	Structured Geometry File. Внутренний формат геометрии CAM-системы.
`ISTGeomFiler` (`sgf`)	Интерфейс управления SGF-файлом: открыть на запись (`StartFile`), закрыть (`CloseFile`). Возвращается функцией `CreateGeomFiler` из STGeomFile.dll.
`ISTGeomReceiver` (`sgr`)	Основной интерфейс импорта: принимает геометрию последовательностью команд (поверхности, кривые, грани, тела, PMI). Тот же COM-объект, что и `ISTGeomFiler`.
ComboModel / ComboSolid	Пара объектов SGF для записи замкнутого тела со сшивкой: грани, рёбра и вершины оболочки собираются в единый твёрдотельный солид (B-rep) без потери связей.
PMI	Product Manufacturing Information — аннотации производственной информации: размеры, допуски, резьбы и т.п.

Требования

Перед началом разработки Дополнения для CAM-системы убедитесь, что готовый продукт будет удовлетворять следующим требованиям:

1. Лицензирование CAD-системы. Файлы CAD-SDK, используемые в работе Дополнения, должны быть допустимы к распространению в составе дистрибутива CAM-системы.
2. Внешняя установка/удаление. Дополнение должно поддерживать установку и удаление без участия пользователя в UI CAD-системы — это нужно для корректной работы Мастера дополнений.
3. Экспорт геометрии. Дополнение должно уметь экспортировать геометрию в один из внутренних форматов CAM-системы.

Дополнительно — если Дополнение поддерживает импорт CAD-файлов по запросу от CAM-системы, к нему предъявляется ещё одно требование:

- Программный импорт CAD-файлов. API CAD-системы должно позволять программно открыть файл проекта этой CAD-системы и сконвертировать его — без интерактивного участия пользователя.

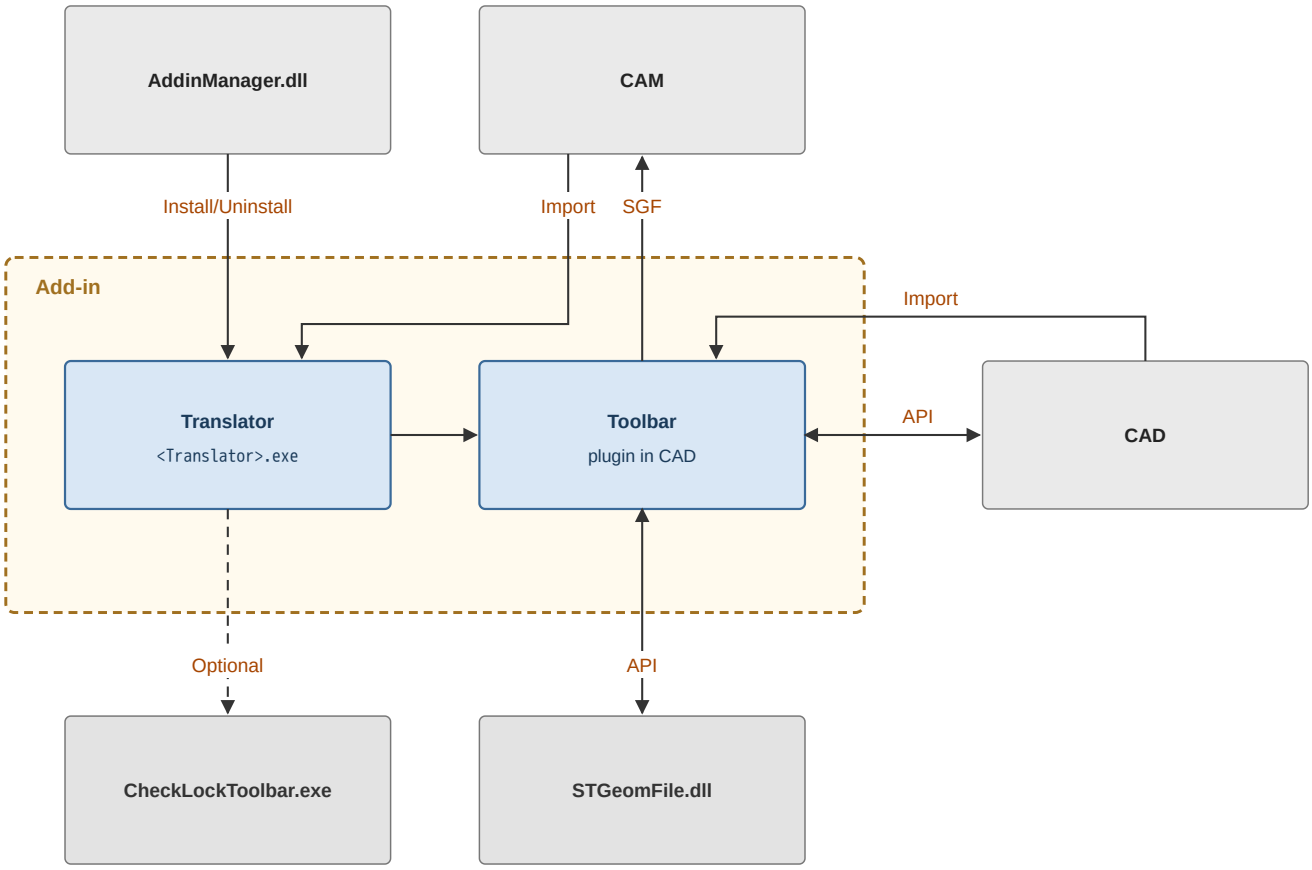
Устройство Панели инструментов

Дополнение состоит из двух частей — Транслятора и Панели инструментов, — которые взаимодействуют с тремя внешними участниками: Мастером дополнений, CAD-системой и CAM-системой. Этот раздел описывает их состав и сценарии работы.

- Описание компонентов — из каких частей состоит Дополнение, какие файлы входят в дистрибутив и за что отвечает каждый компонент.
- Сценарии взаимодействия — как компоненты взаимодействуют между собой: установка и удаление Дополнения, а также два пути переноса геометрии в CAM-систему.

Описание компонентов

Дополнение состоит из двух частей: Транслятор (`<Translator>.exe`) и Панель инструментов (плагин в CAD). Они взаимодействуют с тремя внешними участниками — Мастером дополнений, CAD-системой и CAM-системой.



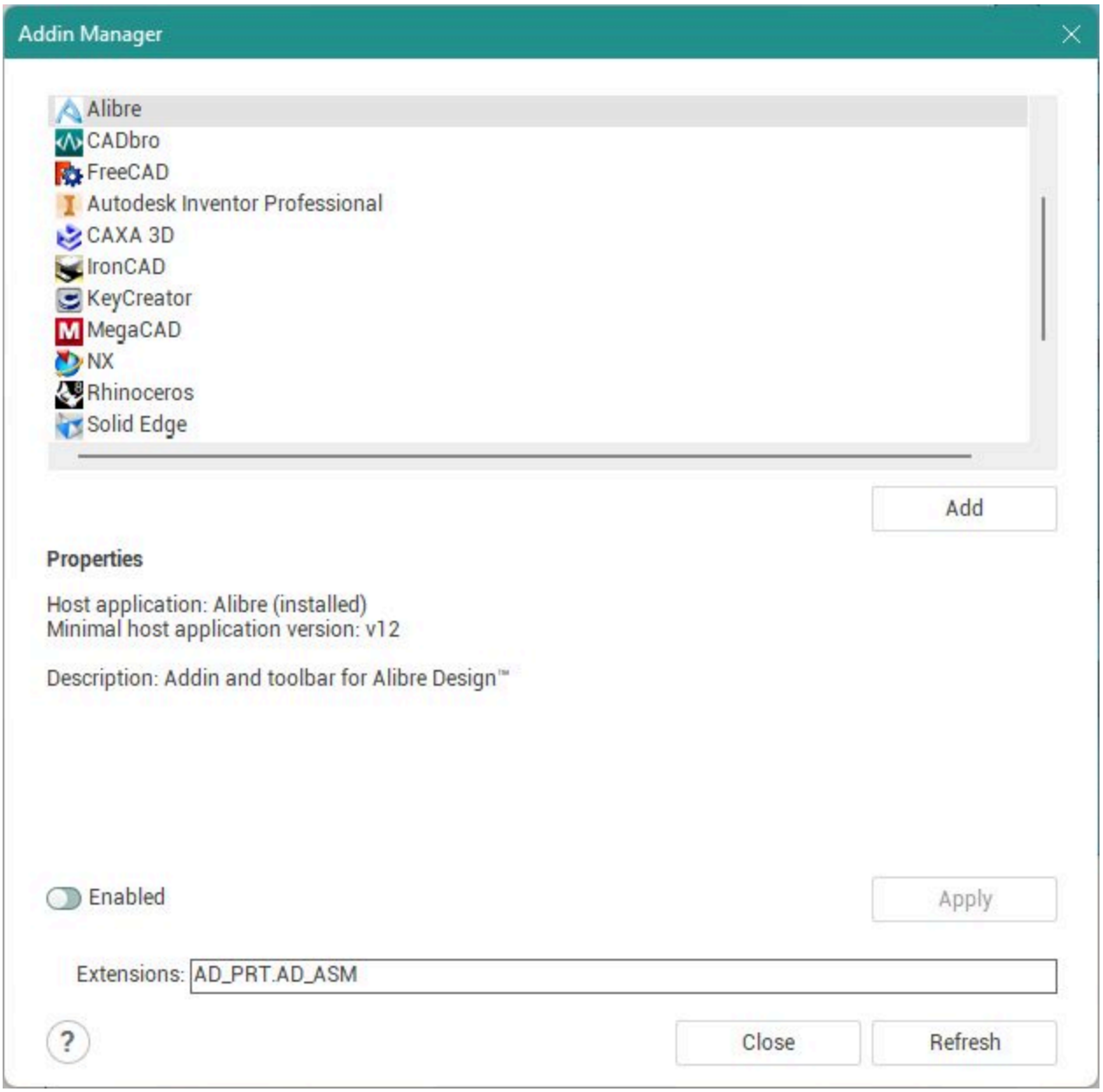
Ниже описаны компоненты Дополнения и его окружение, а также файлы дистрибутива:

- Мастер дополнений — UI-инструмент CAM-системы (`AddinManager.dll`) для управления жизненным циклом Дополнений.
- Файлы дистрибуции — где хранятся Дополнения, шаблон пути и обязательный состав папки.
- Транслятор — единая точка входа `<Translator>.exe`: контракт командной строки и коды возврата.
- Файл XML дополнения — метаданные Дополнения, поддерживаемые форматы и имена команд.
- Файл CAMInfo.ini — текущие настройки CAM-системы, доступные Дополнениям.
- Утилита CheckLockToolbar — вспомогательный код (`LockProcManager`), упрощающий разработку Транслятора.

- Панель инструментов — кнопка в CAD-системе, которая передаёт геометрию в CAM-систему.

Мастер дополнений

UI-инструмент CAM-системы для управления жизненным циклом Дополнений. Реализован библиотекой `AddinManager.dll` в составе CAM-системы.



При выборе Дополнения в верхнем списке вся его дополнительная информация отображается на панели `Свойства`. Обычно это имя и версия приложения, с которым работает Дополнение, описание предоставляемых возможностей, прочая информация.

Поле `Расширения` появляется на экране в случае, если выбранное в списке Дополнение производит работы с файлами, имеющими определённые расширения (например, импорт/экспорт).

Флажок `Включить` меняет параметр установки выбранного Дополнения.

Кнопка `Применить` применяет значение флага `Включить` к выбранному Дополнению.

Кнопка `Закрыть` закрывает Мастер дополнений. При этом если были внесены изменения, то при закрытии будет задан вопрос: «Выйти с сохранением / Выйти без сохранения / Отменить выход».

Кнопка `Обновить` обновляет список Дополнений и все данные о Дополнениях. При этом несохранённые изменения будут утеряны.

Файлы дистрибуции

Все Дополнения хранятся в стандартной директории CAM-системы. Это единственное место, откуда осуществляется управление Дополнениями и взаимодействие с ними — там же лежат файлы, необходимые для работы Панели инструментов.

Шаблон пути.

```
%ProgramData%\<Название_компании>\<Название_продукта>\<Версия_продукта>\AddIns\
<Название_CAD_системы>\
```

Например, для CAD-системы КОМПАС:

```
C:\ProgramData\<Название_компании>\<Название_продукта>\
<Версия_продукта>\AddIns\Kompas\
```

Файлы каждого Дополнения хранятся в отдельной подпапке, названной по CAD-системе. Мастер дополнений сканирует все подпапки `AddIns/` и предлагает их к установке.

Пример файловой структуры Дополнения.

```
AddIns/
  <CAD>/
    <Translator>.exe
    <Translator>.bmp
    <Translator>.xml
    STGeomFile.dll
    CAMInfo.ini
    <ToolbarFiles>
```

Обязательный состав папки. Три файла с одинаковым именем и разными расширениями, без их наличия Мастер дополнений не покажет это дополнение:

Файл	Назначение
`<Translator>.exe`	Исполняемый файл Транслятора (см. Транслятор).
`<Translator>.bmp`	Иконка 16×16, отображается в UI Мастера дополнений.

Файл	Назначение
`<Translator>.xml`	Файл XML дополнения (см. Файл XML дополнения).

Сопровождающие файлы.

Файл	Назначение
`CAMInfo.ini`	Автоматически создаётся Мастером дополнений. Содержит текущие настройки CAM-системы (см. Файл CAMInfo.ini).
Файл Панели инструментов (расширение зависит от CAD)	Файл плагина, который конвертирует геометрию CAD-системы и встраивается в её панель инструментов (см. Панель инструментов).
Ресурсы UI CAD-системы	Иконки кнопок, шрифты, конфигурационные XML/CUIX панелей — формат зависит от CAD. Обычно в подпапке.

Прочие файлы (если они есть) рекомендуется размещать в отдельной подпапке (например, `ToolbarFiles/`) — это упрощает чтение структуры и не мешает Мастеру дополнений находить обязательные файлы.

Транслятор

Единая точка входа для команд Мастера дополнений и для команды импорта от CAM-системы. Standalone-процесс, исполняемый файл `.exe`. Язык реализации произвольный; чаще встречаются Delphi, C++, C# (.NET Framework).

Контракт командной строки. `<Translator>.exe` принимает один из четырёх форматов аргументов:

Аргументы	Семантика
`IsInstalled`	Проверка возможности установки, а именно CAD-система присутствует на машине и все условия для установки выполнены. При успехе вернуть 10. Иначе Мастер дополнений заблокирует возможность установки.
`Install`	Транслятор выполняет необходимые операции, чтобы в CAD-системе появилась Панель инструментов с кнопкой. При успехе вернуть 10. Иначе Мастер дополнений покажет ошибку.
`UnInstall`	Удалить Панель инструментов из CAD-системы. Аналогично команде выше, в итоге в CAD-системе должна исчезнуть кнопка. При успехе вернуть 10.
`$(InFile) $(OutFile)`	Импорт CAD-файла. Транслятор запускает CAD-систему (или подключается к уже запущенной), создаёт новый документ и загружает в него `InFile`, затем выполняет импорт всей геометрии (обычно средствами Панели инструментов) и формирует файл `OutFile`. Подробнее — см. Импорт CAD-файла в CAM-систему.

Имена аргументов команд (`IsInstalled`, `Install`, `UnInstall`, и формат `<In> <Out>`) задаются в Файле XML дополнения через теги `CommandLineIsAppInst`, `CmdLnInstall`, `CmdLnUnInstall`, `CommandLine`. Имена могут отличаться в каждом конкретном Дополнении — Мастер дополнений и CAM-система считывают их из XML.

Коды возврата.

Код	Значение
`0`	False — операция не выполнена (CAM-система покажет окно с ошибкой).
`10`	True — операция выполнена успешно.

Файл XML дополнения

Содержит метаданные Дополнения, описание поддерживаемых форматов и имена команд, которыми Мастер дополнений вызывает `<Translator>.exe`.

Корневая структура.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Collection>
    <CADTranslators>
        <TranslatorName>Alibre</TranslatorName>
        <TranslatorComposition AddinConnected="-1" ToolbarConnected="-1"/>
        <IsExternal>-1</IsExternal>
        <WantAdmin>0</WantAdmin>
        <Enabled>0</Enabled>
        <HideWinow>-1</HideWinow>

<TranslatorPath>C:\ProgramData\Название_компании\Название_продукта\Версия_продукта\A
ddIns\Alibre\AlibreTranslator.exe</TranslatorPath>
        <OutputExtension>IGS</OutputExtension>
        <InputExtensions>AD_PRT.AD_ASM</InputExtensions>
        <CommandLine>$(InFile) $(OutFile)</CommandLine>
        <CommandLineIsAppInst>IsInstalled</CommandLineIsAppInst>
        <CmdLnInstall>Install</CmdLnInstall>
        <CmdLnUnInstall>UnInstall</CmdLnUnInstall>
        <HostApplication>Alibre</HostApplication>
        <DeprecatedHostApplications>
            <Item>GeoMagic</Item>
        </DeprecatedHostApplications>
        <MinimalVersion>v12</MinimalVersion>
        <CADPlatform>xAny</CADPlatform>
        <Description>
            <Item LangCode="1033">Addin and toolbar for Alibre Design™</Item>
            <Item LangCode="1049">Дополнение и панель инструментов для
Alibre Design™</Item>
        </Description>
        <Copyright>Название компании</Copyright>
        <HostCopyright>© Alibre, LLC</HostCopyright>
        <BuildInfo PackageID="" PackageVersion="" BuildNumber=""/>
    </CADTranslators>
</Collection>
```

Описание тегов. Особое внимание следует обратить на `HostApplication` — его значение используется Панелью инструментов при передаче геометрии в CAM-систему.

Имя тега	Описание	Значение
`TranslatorName`	Имя Дополнения, отображаемое в списке Мастера дополнений.	—
`TranslatorComposition`	Атрибуты `AddinConnected` и `ToolbarConnected` показывают какие сценарии импорта будут доступны.	`AddinConnected="-1"` — доступен импорт из CAM-системы (см. Импорт CAD-файла в CAM-систему); `ToolbarConnected="-1"` — доступен импорт из CAD-системы (см. Импорт из CAD-системы в CAM-систему).
`IsExternal`	Показывает, что это внешний импортер.	Должен быть равен `-1`.
`WantAdmin`	Показывает, нужно ли запускать установщик с правами администратора.	`0` — False; `-1` — True.
`WantAdminForInstallation`	Показывает, нужны ли права администратора для установки.	`0` — False; `-1` — True.
`Enabled`	Показывает состояние установки приложения.	По умолчанию `0`. Назначается автоматически.
`HideWinow`	Показывает, нужно ли запускать установщик в скрытом режиме.	`0` — False; `-1` — True.
`TranslatorPath`	Путь к исполняемому файлу Транслятора.	По умолчанию пустой. Назначается автоматически.
`OutputExtension`	Формат файла, в который будет конвертирована геометрия CAD-системы.	Указывается расширение (например, `SGF`, `IGS`, `STEP`).
`InputExtensions`	Форматы файлов, которые умеет импортировать Транслятор. При импорте	Обычно тут указываются родные расширения проектов CAD-системы.

Имя тега	Описание	Значение
	файла CAM-система по этому тегу ищет нужное Дополнение.	Несколько расширений задаются через `.` (например, `CDW.FRW.A3D`).
`CommandLine`	Команда для импорта. Ранее предполагалось, что количество элементов может быть разным, но в итоге остановились на двух.	Должен быть равен `$(InFile) $(OutFile)`.
`CommandLineIsAppInst`	Команда на проверку возможности установки.	Обычно `IsInstalled`.
`CmdLnInstall`	Команда на установку Дополнения.	Обычно `Install`.
`CmdLnUnInstall`	Команда на удаление Дополнения.	Обычно `UnInstall`.
`HostApplication`	Строковый идентификатор CAD-системы. Используется Панелью инструментов при передаче геометрии в CAM-систему — по нему CAM-система находит зарегистрированное Дополнение.	Обычно значение дублируется из тега `TranslatorName`. Главное — должен быть уникальным среди других Дополнений.
`DeprecatedHostApplications`	Старые идентификаторы для обратной совместимости с проектами, созданными в более ранних версиях CAM-системы.	Список `<Item>` с прежними значениями `HostApplication`.
`MinimalVersion`	Минимальная версия CAD-системы, на которую можно установить Дополнение.	Строка свободного формата (например, `v12`, `20+ x64`).
`CADPlatform`	Не используется.	По умолчанию `xAny`.

Имя тега	Описание	Значение
`Description`	Блок с описанием, отображаемый в окне Мастера дополнений. Разбит на подблоки с `LangCode` (locale, номер кодировки) для отображения на разных языках.	`<Item LangCode="1033">` — EN; `<Item LangCode="1049">` — RU.
`Copyright`	Авторские права владельца Дополнения.	—
`HostCopyright`	Авторские права CAD-системы.	—
`BuildInfo`	Метаданные сборки (атрибуты `PackageID`, `PackageVersion`, `BuildNumber`).	Опционально.

Файл CAMInfo.ini

Файл `CAMInfo.ini` содержит текущие настройки CAM-системы, которые используются Дополнениями. Файл автоматически создаётся при сканировании Мастером дополнений в директории каждого Дополнения. При необходимости список может быть дополнен новыми параметрами.

Пример файла CAMInfo.ini:

```
[CAM]
CAMName=<Название_продукта> <Версия_продукта>
CompanyName=<Название_компании>
CompanyURL=<сайт_компании>
CAMPath=C:\Program Files\<Название_компании>\<Название_продукта>\Bin64\<CAM>.exe
CAMLinearMeasure=lmMillimetre
CAMCfgPath=C:\Users\<Пользователь>\AppData\Roaming\<Название_компании>\
<Название_продукта>\<Версия_продукта>
LanguageID=1049
CompanyLocalizedName=<Название_компании>
```

Описание параметров.

Параметр	Описание
`CAMName`	Официальное имя установленного продукта.
`CompanyName`	Официальное название компании. Так как для разных рынков используются разные компании, то лучше использовать `CompanyLocalizedName`.
`CompanyURL`	Ссылка на сайт.
`CAMPath`	Путь к исполняемому файлу CAM-системы.
`CAMLinearMeasure`	Текущие единицы измерений. Возможные значения: `lmMillimetre`, `lmCentimetre`, `lmDecimetre`, `lmMetre`, `lmInch`, `lmFoot`.
`CAMCfgPath`	Путь к файлу настроек.
`LanguageID`	Текущий номер кодировки, используемый в CAM-системе (LCID).

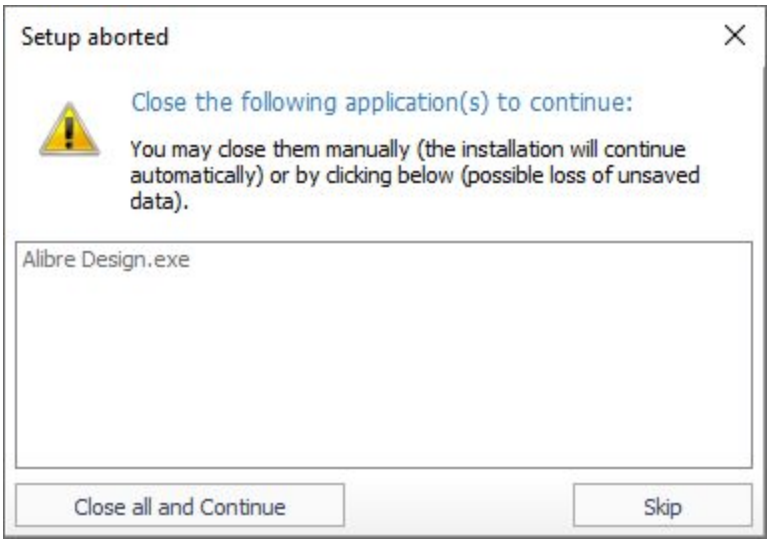
Параметр	Описание
`CompanyLocalizedName`	Локализованное название компании.

Утилита CheckLockToolbar

CheckLockToolbar — репозиторий со вспомогательным кодом, который упрощает разработку Транслятора. К проекту подключается юнит `LockProcManager` (доступны реализации на Delphi и C#); его функции описаны ниже. Также в репозитории собрана утилита `CheckLockToolbar.exe` — её можно вызвать как отдельный процесс, если подключение юнита нежелательно.

`IsTerminateProc(ProcessNames: string): boolean`

Проверяет, не запущены ли указанные процессы в системе. Если находит таковые, то показывает окно:



Данное окно позволяет принудительно завершить указанные процессы или проигнорировать их.

Функция может быть использована во время установки/удаления Дополнения, так как иногда доступ к нужным файлам может быть заблокирован CAD-системой.

Параметры:

Параметр	Описание
`ProcessNames`	Имена процессов через пробел, например `"Alibre Design.exe"`.

Результат:

Значение	Описание
`True`	Указанные процессы не активны или были закрыты.
`False`	Указанные процессы не удалось закрыть, либо пользователь отменил их закрытие.

`RunAs(Filename, Parameters: string; Admin: boolean; WinState: DWORD; Wait: boolean = true): boolean`

Запускает исполняемый файл на выполнение с параметрами.

Параметры:

Параметр	Описание
`Filename`	Имя исполняемого файла.
`Parameters`	Список параметров, разделяются пробелом.
`Admin`	Флаг, запустить с правами администратора.
`WinState`	Параметры запуска процесса (например, `SW_HIDE`, `SW_SHOW`).
`Wait`	Флаг, ожидание завершения процесса.

Результат:

Значение	Описание
`True`	Файл был успешно запущен.
`False`	Ошибка при запуске.

`RunRegsvr32(DllPath, Param: string): boolean`

Регистрирует/разрегистрирует библиотеку с помощью утилиты `regsvr32`, по сути являясь обёрткой для неё.

Параметры:

Параметр	Описание
`DllPath`	Путь к библиотеке.
`Param`	Параметры для утилиты `regsvr32`: `dllregisterserver` — для регистрации, `dllunregisterserver` — для снятия с регистрации.

Результат:

Значение	Описание
`True`	Библиотека успешно зарегистрировалась/разрегистрировалась.
`False`	Ошибка.

Панель инструментов

Панель инструментов — это кнопка в CAD-системе, которая при активации передаёт геометрию в CAM-систему. В зависимости от структуры CAD-системы Панель инструментов может иметь разный вид и различное расположение — в меню, в общей панели, в отдельной панели и так далее.

Все необходимые данные для работы Панели инструментов находятся в двух файлах:

- **`CAMInfo.ini`** (см. Файл CAMInfo.ini) — данные для заполнения названий, подсказок, а также пути к CAM-системе и файлам настроек;
- Файл XML дополнения (см. Файл XML дополнения) — значение тега `HostApplication`, по которому CAM-система связывает переданную геометрию с нужным импортером.

Так как в некоторых случаях приложение Панели инструментов может находиться в другой директории, при установке необходимо предусмотреть доступ к папке с этими файлами.

Дополнительно можно использовать утилиту CheckLockToolbar (см. Утилита CheckLockToolbar).

Алгоритм по нажатию кнопки.

1. Сохранить файл модели в родном формате CAD-системы (далее — файл проекта CAD-системы).
   - Если файла модели не существовало (это новый проект), то нужно показать окно «Сохранить как…». В большинстве случаев это происходит автоматически при вызове соответствующего метода API; если нет — необходимо реализовать эту часть самостоятельно.
   - При отказе пользователя от сохранения необходимо прервать выполнение.
2. Конвертировать файл в один из внутренних форматов CAM-системы. Если есть возможность, предпочтительнее использовать STEP или Parasolid; иначе — сформировать SGF (см. Импорт геометрии в CAM).
3. Запустить CAM-систему с параметрами командной строки (см. пример вызова ниже).

Пример вызова CAM-системы.

```
void LaunchCAM(string fileName, string originalFileName)
{
    if (File.Exists(fCAMPath))
    {
        string par =
            " -EXTIMPORT -HOST \"" + fHost + "\"" +
            " -ORIGINALFILEPATH \"" + originalFileName + "\"" +
            " -DELETEIMPORTFILE \"" + fileName + "\"";
```

```
        RunAs(fCAMPath, par, false, SW_SHOW, false);
    }
}
```

В данном примере используются следующие переменные:

Переменная	Описание
`fCAMPath`	Полный путь к исполняемому файлу CAM-системы. Читается из файла `CAMInfo.ini` (параметр `CAMPath`).
`fHost`	Значение тега `HostApplication` из Файла XML дополнения. Это значение вряд ли будет меняться, поэтому его можно записать в исходник — но нужно помнить об этом.
`originalFileName`	Файл проекта CAD-системы.
`fileName`	Конвертированный файл.

Функция `RunAs` — из юнита `LockProcManager` (см. Утилита CheckLockToolbar).

Сценарии взаимодействия

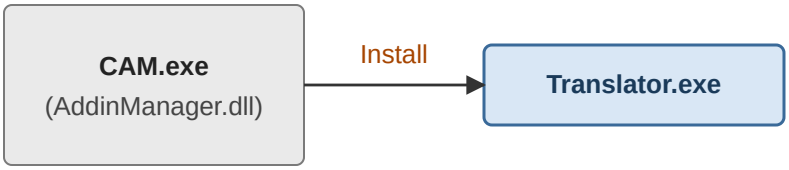
Компоненты Дополнения задействованы в четырёх сценариях, которые делятся на две группы:

- Жизненный цикл Дополнения — установка и удаление; ими управляет Мастер дополнений (`AddinManager.dll`), вызывая Транслятор.
- Перенос геометрии в CAM-систему — по нажатию кнопки Панели инструментов в открытой модели CAD и импорт CAD-файла по запросу CAM-системы.

В каждом сценарии задействован свой набор компонентов и связей (показаны на схемах ниже). CAD-система при этом напрямую не «знает» о CAM-системе — связь обеспечивают Панель инструментов и Транслятор (`<Translator>.exe`).

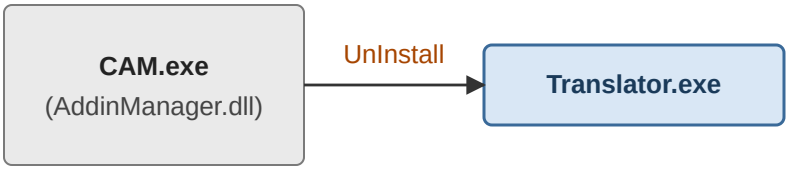
- Установка Дополнения — Мастер дополнений вызывает Транслятор для встраивания Панели инструментов в CAD-систему.
- Удаление Дополнения — Мастер дополнений вызывает Транслятор для удаления Панели инструментов из CAD-системы.
- Импорт из CAD-системы в CAM-систему — пользователь экспортирует открытую модель нажатием кнопки Панели инструментов.
- Импорт CAD-файла в CAM-систему — CAM-система импортирует CAD-файл, вызывая Транслятор на уже загруженной модели.

Установка Дополнения



1. Пользователь открывает Мастер дополнений в CAM-системе.
2. Мастер дополнений сканирует стандартную папку и показывает список найденных Дополнений.
3. Пользователь выбирает нужное Дополнение.
4. Мастер дополнений вызывает `<Translator>.exe IsInstalled`. Если код возврата не равен `10` (False), блокируется кнопка `Применить`.
5. Пользователь устанавливает флажок `Включить` и нажимает кнопку `Применить`.
6. Мастер дополнений вызывает `<Translator>.exe Install`.
7. Транслятор выполняет установку Панели инструментов в CAD-систему. Если код возврата не равен `10` (False), пользователю показывается сообщение об ошибке.

Удаление Дополнения



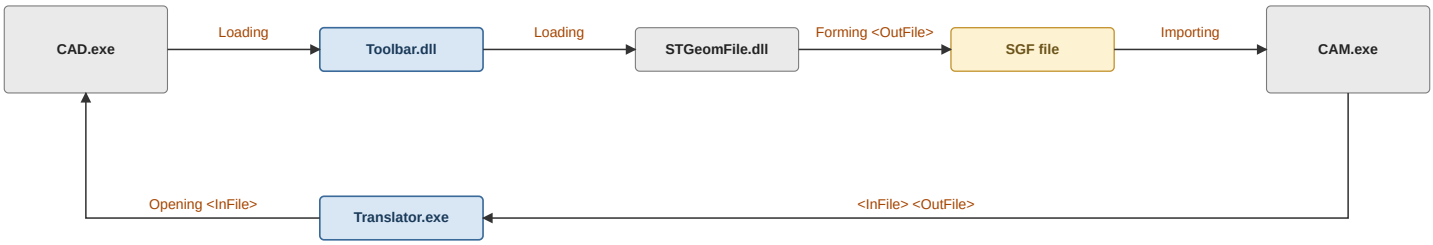
1. Пользователь открывает Мастер дополнений в CAM-системе.
2. Мастер дополнений сканирует стандартную папку и показывает список найденных Дополнений.
3. Пользователь выбирает установленное Дополнение. У установленных Дополнений флажок `Включить` отмечен.
4. Пользователь снимает флажок `Включить` и нажимает кнопку `Применить`.
5. Мастер дополнений вызывает `<Translator>.exe UnInstall`.
6. Транслятор удаляет Панель инструментов из CAD-системы. Если код возврата не равен `10` (False), пользователю показывается сообщение об ошибке.

Импорт из CAD-системы в CAM-систему



1. Пользователь работает в CAD-системе, создаёт или открывает модель.
2. Нажимает кнопку Панели инструментов (которая была добавлена при установке Дополнения).
3. Панель инструментов сохраняет файл в родном формате CAD-системы (для нового файла — диалог «Сохранить как…»; при отмене процесс прерывается).
4. Панель инструментов конвертирует геометрию в один из внутренних форматов CAM-системы (в SGF при прямом импорте либо в любой другой поддерживаемый формат) во временный файл.
5. Панель инструментов запускает CAM-систему с параметрами командной строки (см. Панель инструментов).
6. CAM-система открывается, импортирует переданный файл и удаляет временный файл.

Импорт CAD-файла в CAM-систему



1. Пользователь в CAM-системе импортирует CAD-файл.
2. CAM-система по расширению находит нужное Дополнение, а если их несколько — показывает пользователю окно с выбором.
3. CAM-система вызывает `<Translator>.exe <InFile> <OutFile>` (см. Транслятор).
4. Транслятор запускает экземпляр CAD-системы (по возможности — в скрытом режиме, чтобы не портить пользовательскую сессию).
5. Транслятор создаёт новый документ CAD-системы, загружает в него `InFile` и запускает Панель инструментов на импорт, передав ей `OutFile` как выходной файл.
6. Панель инструментов конвертирует геометрию в один из внутренних форматов CAM-системы (в SGF при прямом импорте либо в любой другой поддерживаемый формат) и сохраняет результат в `OutFile`.
7. Транслятор закрывает документ и открытый экземпляр CAD-системы.
8. CAM-система загружает геометрию из `OutFile` и удаляет его.

Импорт геометрии в CAM

Эта глава описывает ядро Дополнения — перенос геометрии модели из CAD-системы в CAM-систему. Геометрия, доступная через API CAD-системы, преобразуется в файл внутреннего формата CAM-системы, который та затем загружает.

Базовый формат такого переноса — SGF (Structured Geometry File). Он хранит топологию (грани, рёбра, вершины) и геометрию (поверхности, кривые, точки) в виде иерархии связанных сущностей.

Сам файл строит библиотека STGeomFile.dll: она поставляется вместе с CAM-системой и предоставляет API для пошагового создания тел — сначала описываются примитивы геометрии (поверхности, кривые), затем поверх них строится топология (грани и рёбра), и в конце результат сохраняется в файл.

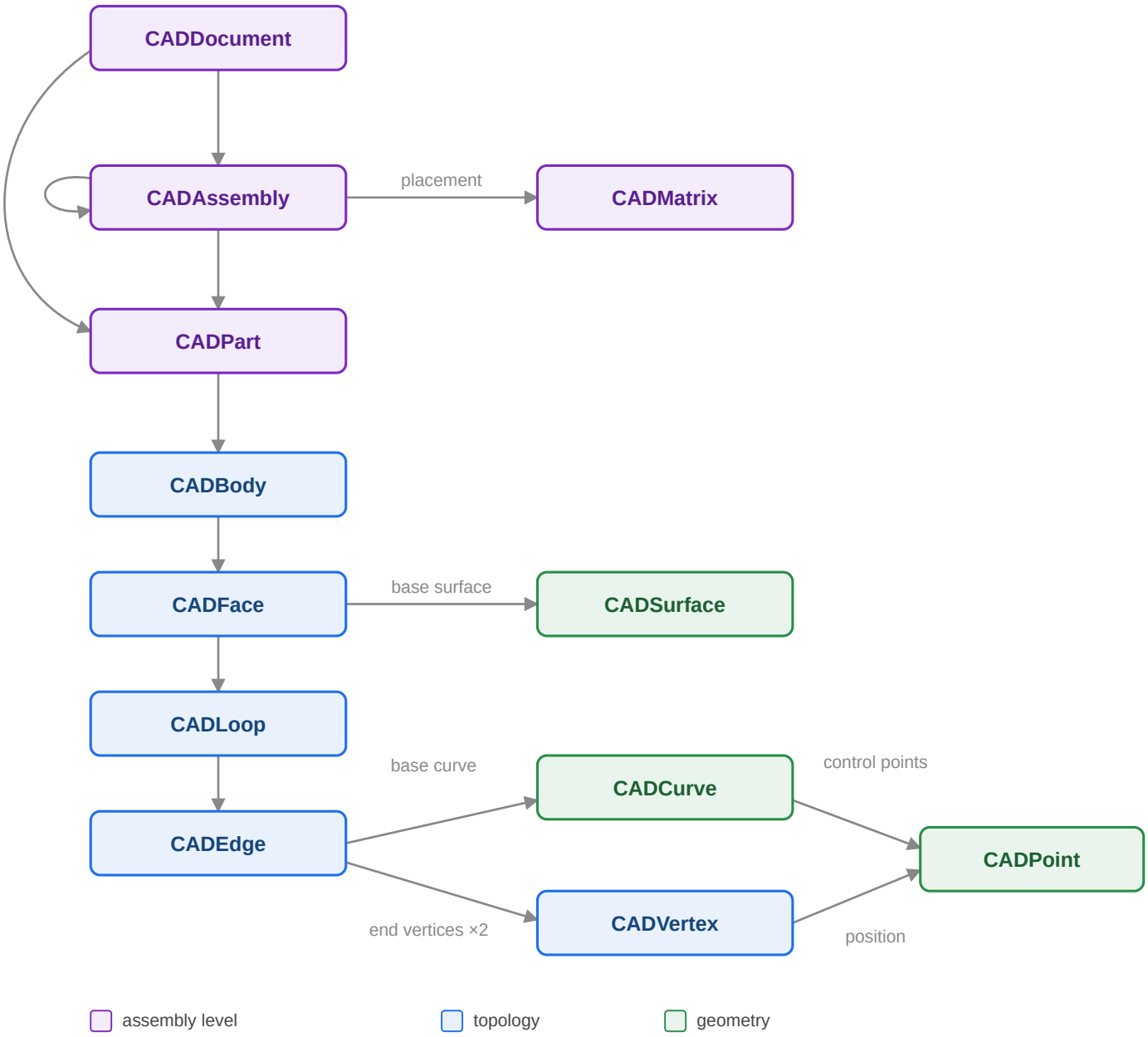
Глава построена по принципу «сначала — *что* импортируем, затем — *как*». Сперва вводится абстрактная структура CAD-модели — иерархия объектов (`CADDocument`, `CADPart`, `CADBody`, …), которую мы в последующих примерах будем переносить в SGF. Затем — два интерфейса импорта (`ISTGeomFiler` и `ISTGeomReceiver`), как подключить библиотеку, чтобы их получить, как сформировать сам файл и базовые структуры данных SGF — типы, на которых написаны все сигнатуры. После этого — работа со сборками: обход дерева компонентов и их размещение, и работа с деталью: сохранение её тел. Далее — импорт геометрии по типам: кривые и точки, грани, поверхности; PMI-аннотации; устаревшие и неиспользуемые методы.

> Язык примеров. Все примеры кода в этой главе приведены на C# — для единообразия и наглядности. Само API к языку не привязано: доступ к нему возможен из любого языка с поддержкой COM (`ISTGeomFiler` / `ISTGeomReceiver`).

Структура CAD-модели

> Это условная (вымышленная) структура. Классы `CADxxx`, описанные ниже, — *не* часть API CAM-системы и не относятся к какой-либо конкретной CAD-системе. Это упрощённая модель данных «гипотетической» CAD, введённая лишь для того, чтобы в примерах кода было видно, *откуда* берутся исходные данные и *как* они отображаются на вызовы SGF. В реальном Дополнении на её месте будут типы вашей CAD-системы (или её API), а логика преобразования останется той же.

CAD-модель — это иерархия связанных объектов: от документа в целом до отдельных точек. Импорт в SGF, по сути, и есть обход этой иерархии с трансляцией каждого объекта в соответствующий вызов `ISTGeomReceiver`.



Сущности модели (сверху вниз — от сборочного уровня к геометрическим примитивам):

Сущность	Что это	Импортируется через
CADDocument	Корень модели — весь CAD-документ. Флаг `IsDetail` определяет содержимое: деталь (`CADPart`) либо сборка (`CADAssembly`).	дерево модели
CADAssembly	Сборка: дерево компонентов. Для каждого компонента флаг `IsDetail` выбирает деталь (`CADPart`) или вложенную сборку (`CADAssembly`); положение задаётся матрицей (`CADMatrix`).	Blocks / инстансы
CADMatrix	Матрица размещения: положение и ориентация компонента сборки в системе координат родителя.	матрица размещения
CADPart	Деталь: одно или несколько тел плюс атрибуты (имя, цвет).	детали и тела
CADBody	Тело: оболочка из граней, описывающая топологию (грани, рёбра, вершины).	ComboSolid
CADFace	Грань: участок несущей поверхности, ограниченный контурами.	обрезанная поверхность
CADSurface	Несущая поверхность грани: плоскость, цилиндр, конус, сфера, тор, NURBS, сетка и др.	поверхности
CADLoop	Контур грани — замкнутая последовательность рёбер.	петли грани

Сущность	Что это	Импортируется через
	Внешний контур задаёт границу, внутренние — отверстия.	
CADEdge	Ребро: участок несущей кривой (`CADCurve`) между двумя вершинами (`CADVertex`).	обрезанная кривая / ребро
CADVertex	Вершина — концевая точка ребра; её положение задаётся точкой (`CADPoint`).	тело (ComboSolid)
CADCurve	Несущая кривая ребра: отрезок, дуга, эллипс, NURBS, кривая пересечения и др.	кривые
CADPoint	Точка в 3D (координаты X/Y/Z) — базовый примитив: положение вершин (`CADVertex`), контрольные и опорные точки кривых и поверхностей.	как параметр прочих вызовов

Связи между сущностями:

- CADDocument → CADPart | CADAssembly — по флагу `IsDetail`: документ — это либо одна деталь, либо сборка;
- CADAssembly → CADPart | CADAssembly — по флагу `IsDetail` компонент сборки — деталь либо вложенная сборка (отсюда рекурсия дерева);
- CADAssembly → CADMatrix — сборка хранит матрицу размещения компонента;
- CADPart → CADBody — деталь состоит из тел;
- CADBody → CADFace — тело ограничено набором граней;
- CADFace → CADSurface (1) + CADLoop (1..*) — у грани одна несущая поверхность и один или несколько контуров;
- CADLoop → CADEdge — контур состоит из рёбер;
- CADEdge → CADCurve (1) + CADVertex (2) — ребро опирается на одну несущую кривую и ограничено двумя вершинами;
- CADVertex → CADPoint — вершина задаётся точкой-положением;
- CADCurve / CADSurface → CADPoint — точки служат контрольными (для NURBS) и опорными.

В последующих главах примеры оформлены как методы `SaveXxx`, которые получают такой `CADxxx`-объект и транслируют его в SGF.

Интерфейсы импорта

Импорт геометрии опирается на два интерфейса одного и того же COM-объекта библиотеки STGeomFile.dll (как его получить — см. Подключение STGeomFile.dll): `ISTGeomFiler` (`sgf`) управляет самим файлом, а `ISTGeomReceiver` (`sgr`) записывает в него геометрию.

ISTGeomFiler

Указатель на интерфейс `ISTGeomFiler` возвращает экспортируемая функция `CreateGeomFiler` (см. Подключение STGeomFile.dll); этот интерфейс управляет файлом:

Метод	Описание
`StartFile(FileName: string)`	Начать формирование SGF-файла (открыть на запись). Вложенный вызов `StartFile` при уже открытом файле недопустим.
`CloseFile()`	Закрыть SGF-файл. Именно при этом вызове файл физически формируется (записывается на диск).
`OpenFile(FileName: string; Receiver: ISTGeomReceiver)`	Открыть SGF-файл на чтение; команды воспроизводятся в указанный `ISTGeomReceiver`. Данный метод используется CAM-системой.

Порядок работы с файлом описан в главе Формирование файла SGF.

ISTGeomReceiver

Основной интерфейс импорта: принимает геометрию в виде последовательности команд. Все методы возвращают `boolean` — статус успешности (в сигнатурах далее он опущен). Объекты создаются под уникальным строковым идентификатором `ID` — по нему на них ссылаются последующие вызовы. Типы параметров (`STFloat`, `TST3DPoint`, `TST2DPoint` и др.) описаны в Структуры данных.

Локальная система координат объекта обычно задаётся тройкой точек: `vT` — положение (начало координат), `vZ` — ось Z, `vX` — ось X. Если ось Y среди параметров не передаётся, то она вычисляется внутри как векторное произведение `vZ × vX`, что автоматически даёт правую систему координат. При этом `vZ` и `vX` должны быть взаимно перпендикулярны, иначе ориентация осей определяется неоднозначно.

Часть методов парные: метод с префиксом `Start…` открывает составной объект, внутри которого вызываются вложенные методы (например, добавление точек, рёбер, узлов), а в конце объект обязательно закрывается одноимённым методом с префиксом `Close…`. Например: `StartNurbsCurve` → `SetNurbsCurveControlPoint` / `SetNurbsCurveKnot` / ⋯ → `CloseNurbsCurve`; `StartFace` → `StartFaceLoop` → `AddFaceEdge2d` → `CloseFaceLoop` → `CloseFace`. Такие пары могут быть вложенными. Если открытый `Start…` не закрыть парным `Close…`, в лучшем случае объект сформируется некорректно, а в худшем — структура файла SGF будет нарушена, а чтение невозможно.

> Опорный объект — раньше зависимого. Независимо от порядка изложения, при записи в SGF каждый объект должен быть создан раньше того, что на него ссылается: кривая — раньше ребра, поверхность — раньше грани, грани — раньше тела. В примерах это обеспечивается рекурсией: метод верхнего уровня (`SaveClosedShell`, `SaveTrimmedFace`, ⋯) сначала сохраняет дочерние объекты, а затем оформляет составной объект поверх них.

Подключение STGeomFile.dll

С помощью `STGeomFile.dll` осуществляется доступ к интерфейсу `ISTGeomFiler`. Библиотека расположена в составе Дополнения (см. Файлы дистрибуции). Также понадобятся два TLB-файла — они описывают интерфейсы (`ISTGeomFiler`, `ISTGeomReceiver`) и перечисления, что даёт типизированный доступ к API:

- `STGeomApiTypes.tlb`;
- `STTypes.tlb`.

Библиотека экспортирует функцию **`CreateGeomFiler`**, которая возвращает ссылку на интерфейс `ISTGeomFiler`.

Пример загрузки библиотеки.

```
[DllImport("kernel32.dll")]
public static extern IntPtr LoadLibrary(string dllToLoad);
[DllImport("kernel32.dll")]
public static extern IntPtr GetProcAddress(IntPtr hModule, string procedureName);
[DllImport("kernel32.dll")]
public static extern bool FreeLibrary(IntPtr hModule);

// Сигнатура экспортируемой функции CreateGeomFiler
[UnmanagedFunctionPointer(CallingConvention.StdCall)]
private delegate ISTGeomFiler TCreateGeomFiler();

private IntPtr            hDLL;
protected ISTGeomFiler    sgf;   // формирование файла: StartFile / CloseFile (см. «Формирование файла SGF»)
protected ISTGeomReceiver sgr;   // запись геометрии: методы в разделах геометрии и далее

public bool ConnectToGeomFiler(string DLLPath)
{
    hDLL = IntPtr.Zero;
    sgf  = null;
    sgr  = null;

    // освобождает уже захваченные ресурсы и сигнализирует о неудаче
    bool Fail()
    {
        sgr = null;
        sgf = null;
        if (hDLL != IntPtr.Zero) { FreeLibrary(hDLL); hDLL = IntPtr.Zero; }
        return false;
```

```
    }

    try
    {
        if (!File.Exists(DLLPath))
            return Fail();

        hDLL = LoadLibrary(DLLPath);
        if (hDLL == IntPtr.Zero)
            return Fail();

        IntPtr pCreateGeomFiler = GetProcAddress(hDLL, "CreateGeomFiler");
        if (pCreateGeomFiler == IntPtr.Zero)
            return Fail();

        var CreateGeomFiler =
(TCreateGeomFiler)Marshal.GetDelegateForFunctionPointer(
            pCreateGeomFiler, typeof(TCreateGeomFiler));

        sgf = CreateGeomFiler();
        if (sgf == null)
            return Fail();

        // тот же объект отдаёт интерфейс записи геометрии (в C# QueryInterface —
это приведение типа)
        sgr = sgf as ISTGeomReceiver;
        if (sgr == null)
            return Fail();

        return true;
    }
    catch
    {
        return Fail();
    }
}
```

Таким образом, `sgf` и `sgr` — это один и тот же COM-объект (см. Интерфейсы импорта).

Формирование файла SGF

Это точка входа в импорт геометрии — момент, когда управление передаётся Дополнению извне. Возможны два сценария (см. Сценарии взаимодействия):

- нажатие кнопки Панели инструментов в CAD-системе — пользователь экспортирует открытую модель (Импорт из CAD-системы в CAM-систему);
- вызов транслятором или CAM-системой при импорте CAD-файла — Дополнение запускается на уже загруженной модели (Импорт CAD-файла в CAM-систему).

В обоих случаях на вход поступает одно и то же: текущая модель CAD-системы (что сохранять) и путь к выходному файлу (куда). Дальше Дополнение формирует SGF-файл по единой схеме:

1. подключиться к библиотеке импорта (см. Подключение STGeomFile.dll);
2. обойти модель CAD сверху вниз и записать её геометрию через `sgr` (см. разделы «Кривые и точки»–«PMI»);
3. освободить интерфейсы и выгрузить библиотеку (`FreeLibrary`).

Глобальные настройки импорта

Сразу после `StartFile`, до записи геометрии, задаются глобальные настройки импорта. Они применяются ко всему файлу и устанавливаются один раз через интерфейс `ISTGeomReceiver` (`sgr`, см. Интерфейсы импорта):

Метод	Описание
`SetModelUnits(Units: TST_LinearMeasure; MultScale: STFloat)`	Установить единицы измерения модели (`lm_Millimetre`, `lm_Centimetre`, `lm_Decimetre`, `lm_Metre`, `lm_Inch`, `lm_Foot`). `MultScale` — масштабный коэффициент. По умолчанию `lm_Millimetre` и `MultScale = 1`.
`SetArcToler(Value: STFloat)`	Установить допуск аппроксимации дуг. По умолчанию `0.001` (в единицах модели).
`SetImportOption(OptionType: TSTImportOption; OptionValue: boolean)`	Включить/выключить опцию импорта `OptionType` (значение `OptionValue`). Значения `TSTImportOption` и их назначение — в таблице ниже.

Опции `TSTImportOption` управляют импортом контуров граней — прежде всего для инвертированных (с обратной нормалью) и циклических (замкнутых по параметру) поверхностей. Обычно менять их не требуется: значения по умолчанию подходят для

большинства случаев, и опции трогают только при специфических проблемах с ориентацией или достраиванием контуров. Возможные значения:

`TSTImportOption`	Назначение	По умолчанию
`ioCreateRectLoops`	Достраивать внешний контур для циклической грани, если он не задан явно: по параметрической области поверхности добавляется прямоугольная (внешняя) петля.	выкл.
`ioInverse3dLoopsForInversedFaces`	Инвертировать направление обхода 3D-контуров (петель из пространственных рёбер) у граней с инвертированной поверхностью — чтобы сохранялось правило «тело грани слева».	вкл.
`ioInverse2dLoopsForInversedFaces`	То же, но для 2D-контуров, заданных в UV-параметрах поверхности.	выкл.

Метаданные заголовка SGF

SGF-файл начинается с заголовка: сигнатура, версия формата и JSON-блок метаданных. Заголовок целиком формирует библиотека при `CloseFile`; Дополнение только дополняет JSON-блок сведениями об источнике — из какого файла, какой CAD-системой и какой версией Дополнения сформирован SGF. CAM-система показывает эти сведения пользователю — например, имя исходной модели. Ключи `originalFileName` и `hostName` обязательны, остальные заполняются по возможности.

Метаданные записываются через `sgr.Header.MetaWriter` (интерфейс `ISGFMetaWriter`) строго между `StartFile` и `CloseFile` — вне этого окна `sgr.Header` возвращает `null`. Корневым JSON-объектом владеет библиотека: пары добавляются сразу, без открытия корня, а каждый `BeginObject`/`BeginArray` закрывается парным `EndObject`/`EndArray` (как и геометрические `Start…`/`Close…`). Методы возвращают `boolean` — статус успешности:

Метод	Описание
`AddStrPair(AKey, AValue: string)`	Добавить пару «ключ — строковое значение». Аналогично `AddIntPair` (`integer`), `AddFltPair` (`STFloat`), `AddBolPair` (`boolean`).
`BeginObject(ObjectID: string)` / `EndObject()`	Открыть / закрыть вложенный объект с указанным ключом.
`BeginArray(ArrayID: string)` / `EndArray()`	Открыть / закрыть вложенный массив с указанным ключом.
`AddStrValue(AValue: string)`	Добавить значение в открытый массив. Аналогично `AddIntValue`, `AddFltValue`, `AddBolValue`.

Стандартный набор ключей приведён в таблице. Имена в таблице — это ключи: CAM-система читает значения именно по ним, поэтому записывать их нужно в точности как указано, не переименовывая. Собственные ключи добавлять можно — версия формата от этого не меняется, незнакомые ключи при чтении игнорируются.

Ключ	Содержимое	Обязательность
`originalFileName`	Полный путь к файлу проекта CAD-системы (см. Панель инструментов).	обязательный
`hostName`	Значение тега `HostApplication` — строковый идентификатор CAD-системы, по которому CAM-система находит Дополнение (см. Файл XML дополнения и Глоссарий).	обязательный
`sourceApp`	Вложенный объект: `name` — имя исполняемого файла CAD-системы, `processFileName` — его полный путь, `version` — версия CAD-системы.	необязательный

Ключ	Содержимое	Обязательность
`toolbarLibrary`	Вложенный объект: `name`, `path`, `version` — имя, путь и версия модуля Дополнения.	необязательный

Ключ `_generated` зарезервирован: эту секцию (`timeUTC` — время формирования файла, `library` — версия библиотеки импорта, `processFileName` — путь процесса, сформировавшего файл) библиотека дописывает сама при `CloseFile`.

Пример записи (вызов метода — в примере точки входа импорта) и итоговый JSON-блок:

```
// Метаданные заголовка: исходный файл, CAD-система, модуль Дополнения
void WriteHeaderMeta(CADDocument doc)
{
    var mw = sgr.Header.MetaWriter;
    mw.AddStrPair("originalFileName", doc.FileName);
    mw.AddStrPair("hostName", fHost); // тег HostApplication из Файла XML дополнения
    mw.BeginObject("sourceApp");
        mw.AddStrPair("name", "mycad.exe");
        mw.AddStrPair("processFileName", cadExePath);
        mw.AddStrPair("version", cadVersion);
    mw.EndObject();
    mw.BeginObject("toolbarLibrary");
        mw.AddStrPair("name", "MyCADToolbar.dll");
        mw.AddStrPair("path", toolbarPath);
        mw.AddStrPair("version", toolbarVersion);
    mw.EndObject();
}
```

```
{
  "originalFileName": "C:\\Models\\part.mcad",
  "hostName": "MyCAD",
  "sourceApp":      { "name": "mycad.exe", "processFileName":
"C:\\MyCAD\\mycad.exe", "version": "3.2.1" },
  "toolbarLibrary": { "name": "MyCADToolbar.dll", "path":
"C:\\MyCAD\\Addins\\MyCADToolbar.dll", "version": "1.4.0.12" },
  "_generated":     { "timeUTC": "2026-07-15T10:30:00Z", "library": "STGeomFile
1.0.0.0", "processFileName": "C:\\MyCAD\\mycad.exe" }
}
```

Пример: точка входа импорта

```
// Точка входа импорта. На вход — текущая модель CAD-системы и путь к выходному
SGF-файлу.
bool ExportToSGF(CADDocument doc, string outFilePath)
{
    if (!ConnectToGeomFiler(GetLibraryPath()))
        return false;

    bool ok = false;
    try
    {
        sgf.StartFile(outFilePath);
        try
        {
            // метаданные заголовка (см. «Метаданные заголовка SGF»)
            WriteHeaderMeta(doc);

            // sgr.SetModelUnits(TST_LinearMeasure.lm_Metre, 1.0);
            // sgr.SetArcToler(0.005);
            // sgr.SetImportOption(TSTImportOption.ioCreateRectLoops, true);

            // UnitMatrix3D — единичная матрица (корень без смещения)
            if (!doc.IsDetail)
                SaveAssembly(doc.Assembly, UnitMatrix3D);
            else
                SavePart(doc.Part, UnitMatrix3D);
        }
        finally
        {
            sgf.CloseFile();
        }
        ok = true;
    }
    catch
    {
        ok = false;
    }
    finally
    {
        sgr = null;
        sgf = null;
        if (hDLL != IntPtr.Zero) { FreeLibrary(hDLL); hDLL = IntPtr.Zero; }
    }
    return ok;
}
```

Структуры данных SGF

Базовые типы и структуры объявлены в `STTypes.idl`. На них написаны все сигнатуры методов в последующих разделах.

Тип	Определение	Назначение
`STFloat`	`double`	Вещественное число: координаты, радиусы, углы, параметры, веса.
`TST3DPoint`	`record (X, Y, Z: double)`	Точка или вектор в 3D. В роли вектора (`vZ`, `vX`, `Axis`, `ExtrusionVector`) задаёт направление.
`TST2DPoint`	`record (X, Y: double)`	Точка в 2D — в параметрическом пространстве поверхности (при построении 2D-кривых контура).
`TST3DMatrix`	`record (vX, vY, vZ, vT: TST3DPoint; A, B, C, D: double)`	Матрица преобразования / система координат: базисные векторы `vX`, `vY`, `vZ`, начало координат `vT` и коэффициенты `A`–`D`.

В сигнатурах также используются примитивы: `string` — текстовый идентификатор (`ID`) или путь к файлу, `integer` — целое число, `boolean` — флаг.

Преобразования CAD → SGF в примерах. Типы CAD-системы (`CADPoint`, `CADMatrix`, …) не совпадают с типами SGF, поэтому в примерах используются небольшие функции-преобразования: `ToPoint` приводит CAD-точку к `TST3DPoint`, а `ToMatrix` — матрицу (систему координат) CAD к `TST3DMatrix`. Это покомпонентное копирование полей; реализация тривиальна и в примерах не приводится — показаны только вызовы (с поясняющими комментариями).

Работа со сборкой

Сборка (`CADAssembly`) — узел дерева модели с матрицей размещения (`CADMatrix`). Флаг `IsDetail` задаёт содержимое узла: при `IsDetail = true` внутри лежит деталь (`CADPart`) с геометрией, при `IsDetail = false` — вложенные сборки. Матрица есть только у **`CADAssembly`**, поэтому любой размещаемый компонент — даже одиночная деталь (винт, шайба) — оборачивается в собственную `CADAssembly` (она несёт его положение), а геометрия хранится внутри, в `CADPart`. Один и тот же `CADPart` может переиспользоваться многими сборками (массивы, повторяющийся крепёж). Импорт сборки — рекурсивный обход дерева сверху вниз с накоплением преобразования.

Например, так узел «Ролик в кронштейне» выглядит в дереве модели CAD-системы:

```
Ролик в кронштейне — CADAssembly
├─ Кронштейн — CADAssembly → CADPart
├─ Ось — CADAssembly → CADPart
├─ Ролик — CADAssembly
│  ├─ Втулка — CADAssembly → CADPart
│  └─ Обойма — CADAssembly → CADPart
├─ Винт M6 (1) — CADAssembly → CADPart
├─ Винт M6 (2) — CADAssembly → CADPart
├─ Винт M6 (3) — CADAssembly → CADPart
├─ Винт M6 (4) — CADAssembly → CADPart
├─ Шайба 6 (1) — CADAssembly → CADPart
├─ Шайба 6 (2) — CADAssembly → CADPart
├─ Шайба 6 (3) — CADAssembly → CADPart
└─ Шайба 6 (4) — CADAssembly → CADPart
```

Каждый узел дерева — это `CADAssembly` со своей матрицей; собственно геометрия появляется только в листовых `CADPart` внутри них. «Ролик» — вложенная подсборка (`IsDetail = false`). «Винт M6» и «Шайба 6» присутствуют в четырёх экземплярах — это четыре отдельные `CADAssembly` (четыре матрицы), ссылающиеся на одну и ту же геометрию `CADPart`.

> В примерах ниже узел сборки (`CADAssembly`) имеет поля: `Name` — имя (папка в дереве), `Transform` — матрица размещения (`CADMatrix`), `IsDetail` — флаг содержимого, и само содержимое: при `IsDetail = true` — деталь `Part` (`CADPart`), при `IsDetail = false` — список дочерних узлов `Children`. В Варианте 2 у листовых узлов используется ещё `ComponentId` — идентификатор исходного компонента, общий для всех одинаковых деталей (по нему определяются повторы). Конкретный состав полей зависит от вашей CAD-системы.

Что нужно воспроизвести в SGF. Импорт сборки решает три задачи: повторить структуру дерева (папки), разместить каждый компонент его матрицей и записать геометрию деталей

(`ComboModel` / `ComboSolid`, см. Работа с деталью). Структуру дерева строят группы, а повторяющиеся компоненты можно переиспользовать через блоки. Ниже — оба способа; они равноправны и обычно сочетаются.

Группа. Каждый узел сборки отображается в дереве модели CAM-системы как папка. Пара методов `StartGroupEntity` / `CloseGroupEntity` открывает и закрывает такую папку: все объекты, записанные между вызовами, принадлежат ей. Вызовы вкладываются — ровно так же, как вложены узлы дерева CAD-сборки.

Метод	Описание
`StartGroupEntity(EntityName: string)`	Начать группу — папку в дереве модели с именем `EntityName`. Все объекты, построенные до парного `CloseGroupEntity`, попадают в эту папку. Вызовы можно вкладывать: внутри папки создаются вложенные папки. Нельзя вызывать внутри `StartComboModel` — так как он формирует отдельную закрытую сшитую оболочку.
`CloseGroupEntity()`	Закрыть текущую группу (папку).

Матрица преобразования. Положение компонента задаётся перед сохранением его геометрии. Матрица задаётся методом `SetCurrentTransform2`, и все последующие поверхности — до следующей смены матрицы — умножаются на неё. При обходе сборки в матрицу передаётся накопленное мировое преобразование узла (произведение матриц от корня), поэтому для каждого солида задаётся его финальное положение.

Матрицу вызывают до открытия солида — перед `StartGroupEntity` или `StartComboModel`. Внутри `StartComboModel` менять её нельзя: тело сшивается в собственных координатах и размещается целиком по матрице, заданной до его открытия (см. Работа с деталью).

Метод	Описание
`SetCurrentTransform2(vT, vX, vY, vZ: TST3DPoint)`	Переключает матрицу преобразования для всех последующих объектов; задаётся положением (`vT` — начало координат) и явно заданными осями `vX`, `vY`, `vZ`. Нужно вызывать перед сохранением солида (перед `StartGroupEntity` или `StartComboModel`) — все его поверхности будут умножены на эту матрицу. При сохранении сборок задаётся

Метод	Описание
	финальная матрица для каждого солида. Нельзя вызывать внутри `StartComboModel`.
`SetCurrentTransform(vT, vZ, vX: TST3DPoint)`	Устарело (но в некоторых случаях ещё используется). То же, что `SetCurrentTransform2`, но система координат задаётся положением (`vT`), осью Z (`vZ`) и осью X (`vX`); ось Y вычисляется как векторное произведение `vZ × vX` — это неоднозначно, если `vZ` и `vX` не строго перпендикулярны. По возможности используйте `SetCurrentTransform2`.

Вариант 1 — стандартный обход (группы). Самый простой подход: каждый узел сборки оборачивается в `StartGroupEntity` / `CloseGroupEntity`, геометрия детали записывается внутри. Матрица размещения накапливается по пути от корня: мировое преобразование каждого узла = преобразование родителя × локальная матрица узла.

```
// Рекурсивный обход сборки. parentTransform — накопленное преобразование родителя.
void SaveAssembly(CADAssembly asm, TST3DMatrix parentTransform)
{
    // мировое преобразование узла = преобразование родителя × локальная
матрица сборки
    TST3DMatrix world = Multiply(parentTransform, ToMatrix(asm.Transform));

    sgr.StartGroupEntity(asm.Name);
    try
    {
        if (asm.IsDetail)
            SavePart(asm.Part, world);                // лист: деталь с геометрией
        else
            foreach (CADAssembly child in asm.Children)
                SaveAssembly(child, world);           // подсборка: спускаемся в
дочерние узлы
    }
    finally
    {
        sgr.CloseGroupEntity();
    }
}
```

Этот вариант подходит для большинства случаев. Недостаток: если один и тот же компонент встречается в сборке многократно (например, четыре одинаковых болта), его геометрия записывается в файл четыре раза. Для таких случаев предназначен Вариант 2.

Вариант 2 — блоки (Blocks). Блок — это одна модель с топологией (`ComboModel`) или без, записанная в SGF один раз под уникальным именем и вставляемая в дерево любое количество раз через `InsertBlock`. Геометрия хранится единожды, каждый экземпляр задаёт только матрицу размещения. Для компонентов, встречающихся больше одного раза, это сокращает размер файла и ускоряет загрузку.

- подсборка (`IsDetail = false`) — обычная папка `StartGroupEntity` / `CloseGroupEntity` (как в Варианте 1) с рекурсивным обходом дочерних узлов;
- деталь (`IsDetail = true`) — сохраняется как блок: при первом вхождении компонента его геометрия записывается в блок (`StartBlock` / `CloseBlock`) и тут же размещается `InsertBlock`; при последующих вхождениях вызывается только `InsertBlock`. Уже записанные блоки отслеживает множество `savedBlocks` (в импортёре STEP — флаг `fShellSaved`), предварительный подсчёт повторов не нужен.

`InsertBlock` сам создаёт узел в дереве модели (его аргумент `NewBlockName` — имя этого узла), поэтому оборачивать узел в дополнительный `StartGroupEntity` не нужно.

Методы блоков.

Метод	Описание
`StartBlock(BlockName: string)`	Начать блок — зафиксировать под именем `BlockName` одну модель с топологией (`ComboModel`) или без для последующего переиспользования. Геометрию внутри блока записывают те же методы, что и для одиночной детали (см. Работа с деталью). Блок помещается в буфер.
`CloseBlock()`	Завершить формирование блока (он остаётся в буфере).
`InsertBlock(BlockID, NewBlockName: string)`	Скопировать блок из буфера в дерево модели. `BlockID` — имя блока из `StartBlock`; `NewBlockName` — имя создаваемого узла (группы) в дереве. Матрица размещения задаётся предшествующим вызовом `SetCurrentTransform2`.

```
// savedBlocks — ComponentId уже записанных блоков (поле импортёра; аналог
fShellSaved в STEP).
readonly HashSet<string> savedBlocks = new HashSet<string>();

void SaveAssemblyWithBlocks(CADAssembly asm, TST3DMatrix parentTransform)
{
    TST3DMatrix world = Multiply(parentTransform, ToMatrix(asm.Transform));

    if (!asm.IsDetail)
    {
        sgr.StartGroupEntity(asm.Name);
        try
        {
            foreach (CADAssembly child in asm.Children)
                SaveAssemblyWithBlocks(child, world);
        }
        finally
        {
            sgr.CloseGroupEntity();
        }
        return;
    }

    // лист (деталь): при первом вхождении компонента создаём блок, далее —
только вставляем
    if (savedBlocks.Add(asm.ComponentId))
    {
        sgr.StartBlock(asm.ComponentId);
        try
        {
            SavePart(asm.Part, UnitMatrix3D);
        }   // геометрия в локальных координатах детали
        finally
        {
            sgr.CloseBlock();
        }
    }
    // размещаем экземпляр по его матрице (в том числе при первом вхождении)
    sgr.SetCurrentTransform2(world.vT, world.vX, world.vY, world.vZ);
    sgr.InsertBlock(asm.ComponentId, asm.Name);    // InsertBlock сам создаёт
узел дерева
}
```

> Оба варианта равноправны и могут сочетаться в одном импортёре. Вариант 1 проще в реализации и отладке; Вариант 2 даёт меньший размер SGF-файла при сборках с

> повторяющимися деталями и загружается быстрее. В примере выше детали оформляются блоками (повторная геометрия пишется один раз и переиспользуется), а подсборки и сборочные узлы остаются папками-группами.

Работа с деталью

Деталь (`CADPart`) состоит из тел (`CADBody`) и может также содержать отдельные кривые, точки и PMI. Тело — это оболочка из граней. Замкнутое тело (солид) сохраняется отдельно через `ComboSolid`, который обеспечивает сшивку: грани, вершины и рёбра оболочки записываются внутрь пары `ComboModel` → `ComboSolid`, и при загрузке CAM-система собирает из них единый твёрдотельный солид. Незамкнутые оболочки, а также отдельные кривые и точки в `ComboSolid` не попадают — они добавляются в дерево модели напрямую через `AddEntity` (см. ниже).

Зачем сшивать модель. Из CAD-модели приходит вся топология тела — и поверхности, и рёбра, и вершины вместе со связями между ними. `ComboSolid` переносит эти связи в SGF без потерь: CAM получает ту же связную топологию (B-rep) с понятными «внутри/снаружи» и соседством граней, что была в CAD, и ему не нужно заново искать и угадывать стыки.

Грани добавляются в тело уже после того, как они созданы через `StartFace` (Грань (обрезанная поверхность)); связи по общим рёбрам и вершинам и задают сшивку.

Метод	Описание
`StartComboModel(GeomFile: string)`	Начать модель с топологией (будет «сшита»). `GeomFile` — уникальное имя модели.
`StartComboSolid(ID: string)`	Начать тело внутри `ComboModel`.
`AddComboSolidVertex(p: TST3DPoint; VertexID: string)`	Добавить вершину тела `p` под идентификатором `VertexID` (на него ссылаются рёбра).
`AddComboSolidEdge(EdgeID, CurveID, StartVertexID, EndVertexID, LeftFaceID, RightFaceID: string)`	Ребро тела между вершинами `StartVertexID`⋯`EndVertexID`, разделяющее левую (`LeftFaceID`) и правую (`RightFaceID`) грани — именно эта связь задаёт сшивку. `CurveID` обычно лучше передавать пустым: собственная геометрия топологическому ребру не особо нужна — она и так локализуется стыком левой и правой поверхностей. Если передать `CurveID`, при сшивке движок попытается вставить геометрию указанной кривой, а она не всегда учитывает, что ребро при сшивке может быть разбито на несколько.

Метод	Описание
`AddComboSolidFace(FaceID, FaceName: string)`	Добавить ранее созданную (через `StartFace`) грань.
`CloseComboSolid()`	Закрыть тело.
`CloseComboModel()`	Закрыть модель.

Уникальные идентификаторы и ассоциативность. Для топологических элементов — грани, ребра и вершины — критично задавать уникальные идентификаторы. По ним элементы ссылаются друг на друга (ребро — на две свои грани и две вершины, и т.д.), и это обеспечивает ассоциативность: после сшивки и последующих пересчётов CAM может однозначно сопоставить операцию с конкретной гранью, а не «потерять» её. Поэтому идентификаторы должны быть уникальными и стабильными.

Сами уникальные идентификаторы обеспечивает CAD-система: у каждого топологического элемента в её модели есть собственный постоянный идентификатор, который сохраняется между перестроениями. Транслятору не нужно генерировать идентификаторы самостоятельно — достаточно взять штатный идентификатор элемента из API CAD-системы и передать его в SGF. Это гарантирует и уникальность, и стабильность: при повторном импорте изменённой модели неизменные элементы сохранят те же идентификаторы, и привязки операций не разрушатся.

Пример: сохранение детали. Метод `SavePart`, вызываемый при обходе сборки (Работа со сборкой), получает деталь и её мировую матрицу, задаёт текущее преобразование и сохраняет её содержимое — тела, отдельные кривые, точки и PMI. Саму деталь удобно обернуть в папку дерева модели (`StartGroupEntity`, Работа со сборкой) — здесь это организационная папка, а не сшивка.

```
void SavePart(CADPart part, TST3DMatrix transform)
{
    sgr.SetCurrentTransform2(transform.vT, transform.vX,
transform.vY, transform.vZ);

    sgr.StartGroupEntity(part.Name);
    try
    {
        SaveBodies(part.Bodies);   // замкнутые тела — через ComboSolid, открытые —
через AddEntity
        SaveCurves(part.Curves);   // кривые, см. «Кривые и точки»
        SavePoints(part.Points);   // точки, см. «Кривые и точки»
        SavePMI(part.PMIs);        // PMI (резьбы и т.п.), см. «PMI»
    }
```

```
    finally
    {
        sgr.CloseGroupEntity();
    }
}

void SaveBodies(CADBody[] bodies)
{
    foreach (CADBody body in bodies)
    {
        if (body.IsClosed)
            SaveClosedShell(body);   // замкнутое тело → ComboSolid (сшивка)
        else
            SaveOpenShell(body);     // открытая оболочка → AddEntity
    }
}

// Открытая оболочка — не тело: грани добавляются в дерево напрямую через AddEntity,
вне ComboModel.
void SaveOpenShell(CADBody body)
{
    sgr.StartGroupEntity(body.Name);   // папка оболочки в дереве модели
    try
    {
        foreach (CADFace face in body.Faces)
            if (SaveTrimmedFace(face))      // создаёт грань под face.Id
(см. «Грань»)
                sgr.AddEntity(face.Id, face.Name);
    }
    finally
    {
        sgr.CloseGroupEntity();
    }
}

// Замкнутая оболочка → один ComboSolid внутри ComboModel; сшивку обеспечивает
сам ComboSolid.
void SaveClosedShell(CADBody body)
{
    sgr.StartComboModel(body.Id);
    try
    {
        sgr.StartComboSolid(body.Id);
        try
        {
            // 1) грани оболочки: сохраняем каждую и добавляем в тело
```

```
            foreach (CADFace face in body.Faces)
                if (SaveTrimmedFace(face))
                    sgr.AddComboSolidFace(face.Id, face.Name);

            // 2) вершины тела (на них по Id ссылаются рёбра); координаты приводим
к TST3DPoint
            foreach (CADVertex vertex in body.Vertices)
                sgr.AddComboSolidVertex(ToPoint(vertex.Point), vertex.Id);

            // 3) рёбра тела: какие грани (левая/правая) сходятся на общем ребре —
это и есть сшивка
            foreach (CADEdge edge in body.Edges)
                sgr.AddComboSolidEdge(edge.Id, "",
                                      edge.Start.Id, edge.End.Id,
                                      edge.LeftFaceId, edge.RightFaceId);
        }
        finally
        {
            sgr.CloseComboSolid();
        }
    }
    finally
    {
        sgr.CloseComboModel();
    }
}
```

> `body.Vertices` и `body.Edges` — это топология тела: вершины с координатами (`Point`) и идентификаторами (`Id`); каждое ребро (`CADEdge`) хранит концевые вершины (`Start`, `End`) и идентификаторы смежных граней (`LeftFaceId`, `RightFaceId`). Вершины добавляются раньше рёбер — ребро ссылается на них по `Id`. Для `CurveID` передаётся пустая строка: как и в импортёре STEP, геометрию ребра несут грани.

Открытые оболочки, кривые и точки. Не всё в детали — замкнутые тела. Незамкнутая оболочка (`body.IsClosed == false`), отдельные 3D-кривые и точки в `ComboSolid` не попадают: класть открытую оболочку в `ComboSolid` бессмысленно — сшивать нечего, замкнутого тела не выйдет, — а с кривыми и точками он не работает вовсе. Такие объекты создают вне `StartComboModel` и добавляют в дерево модели напрямую вызовом `AddEntity(ID, EntityName)` (Работа со сборкой) — по тому же `ID`, под которым объект был создан (см. `SaveOpenShell` / `SaveCurves` / `SavePoints` в примере выше). Размещение, если нужно, задаётся матрицей `SetCurrentTransform2` перед `AddEntity`.

Метод	Описание
`AddEntity(ID: string; EntityName: string)`	Добавить в дерево модели созданную ранее сущность по её `ID` (грань, кривую, точку) под именем `EntityName`. Её локальные координаты умножаются на матрицу, заданную через `SetCurrentTransform2`. Метод можно вызывать и внутри `ComboModel` — сущность попадёт в дерево самостоятельным объектом, вне сшивки (в сшивку тела грани добавляет `AddComboSolidFace`).

Кривые и точки

Методы построения геометрии кривых: аналитические примитивы (точка, отрезок, окружность, дуга, эллипс, спираль, NURBS, составные кривые), производные кривые (на поверхности, обрезанные, инвертированные, пересечения) и их параметризация.

Цвет. Текущий цвет задаётся вызовом `SetCurrentColor(Value: integer)`. Это состояние: заданный цвет применяется ко всем создаваемым после него объектам — поверхностям, кривым и точкам (добавляемым через `AddEntity` или `AddComboSolidFace`) и сохраняется, пока не будет изменён следующим вызовом. Поэтому `SetCurrentColor` вызывают перед каждым объектом, которому нужен свой цвет. Если `SetCurrentColor` не вызывался, цвет не переопределяется и объект наследует цвет родительского узла дерева. Формат значения — упакованное целое `Blue shl 16 or Green shl 8 or Red` (по 8 бит на канал).

Пример: сохранение кривых и точек.

```
void SaveCurves(CADCurve[] curves)
{
    foreach (CADCurve curve in curves)
        if (SaveCurve(curve))
        {
            sgr.SetCurrentColor(curve.Color);  // цвет действует на следующий
AddEntity (и далее, пока не сменится)
            sgr.AddEntity(curve.Id, curve.Name);
        }
}

void SavePoints(CADPoint[] points)
{
    foreach (CADPoint point in points)
    {
        sgr.CreatePoint(point.Id, ToPoint(point));   // координаты CAD-точки
приводим к TST3DPoint
        sgr.AddEntity(point.Id, point.Name);         // SetCurrentColor не вызывали
— точка наследует текущий/родительский цвет
    }
}
```

Методы построения кривых сгруппированы в трёх разделах:

- Точка и аналитические кривые — точка, система координат, отрезок, окружность, дуга, эллипс, спираль, NURBS-кривая, составные кривые.

- Производные кривые — кривая на поверхности, обрезанная, инвертированная, кривая пересечения и граница скругления.
- Параметризация кривых — изменение диапазона и перепараметризация уже созданной кривой.

Точка и аналитические кривые

Кривые строятся в локальной плоскости XY: `vT` — центр, `vZ` — нормаль (ось из плоскости), `vX` — ось X локальной системы, ось Y вычисляется как векторное произведение Z и X (поэтому её не нужно передавать). Построение кривой начинается от направления `vX` (это начальная точка её параметризации) и идёт против часовой стрелки вокруг `vZ`; в том же направлении откладываются и углы. Созданные объекты можно либо добавить в геометрическое дерево через `AddEntity`, либо использовать как строительные блоки для других объектов — например, в качестве кривых петли для обрезки поверхностей.

Примитивы точки и аналитических кривых:

- 3D-точка — `CreatePoint`.
- Система координат — `CreateCoordinateSystem`.
- Отрезок — `CreateLineSeg`.
- Окружность — `CreateCircle`.
- Дуга окружности — `CreateArc`.
- Эллипс — `CreateEllipse`.
- Дуга эллипса — `CreateEllipseArc`.
- Коническая спираль — `CreateConicalSpiral`.
- NURBS-кривая — `StartNurbsCurve` ⋯ `CloseNurbsCurve`.
- Составная кривая — плоская (2D) и пространственная (3D).

Пример сохранения аналитической кривой

`SaveCurve` — диспетчер: получает кривую CAD-системы и, в зависимости от её типа, вызывает соответствующий обработчик (`SaveLine`, `SaveCircle`, ⋯). Каждый обработчик пишет геометрию в SGF подходящим вызовом из Кривые и точки и возвращает флаг успеха. Ниже показаны сам диспетчер и один из обработчиков (`SaveLine`); остальные устроены аналогично.

```
// Сохраняет кривую любого типа; возвращает признак успеха (Id уже задан в CADCurve)
bool SaveCurve(CADCurve curve)
{
    switch (curve.Type)
    {
        case "Line":    return SaveLine(curve.Id, curve.Param);
        case "Circle":  return SaveCircle(curve.Id, curve.Param);      //
обработчики остальных
        case "Arc":     return SaveArc(curve.Id, curve.Param);         // типов
устроены так же
        case "Ellipse": return SaveEllipse(curve.Id, curve.Param);
        case "Nurbs":   return SaveNurbsCurve(curve.Id, curve.Param);
        // … прочие типы кривых (см. «Кривые и точки»)
```

```
        default:        return false;                   // тип не поддерживается
    }
}

// Один из обработчиков — сохранение отрезка
bool SaveLine(string id, CADLineParam param)
{
    // координаты CAD-точек приводим к TST3DPoint
    return sgr.CreateLineSeg(id, ToPoint(param.Start), ToPoint(param.End));
}
```

3D-точка

`CreatePoint(ID: string; P: TST3DPoint)` — создать 3D-точку.

- `ID` — идентификатор создаваемой сущности;
- `P` — координаты точки.

Система координат

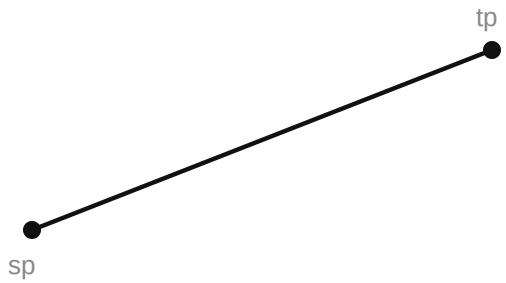
`CreateCoordinateSystem(CSID: string; vT, vZ, vX: TST3DPoint)` — создать систему координат. Задаётся в абсолютных координатах.

- `CSID` — идентификатор системы координат;
- `vT` — положение (начало координат);
- `vZ` — ось Z;
- `vX` — ось X.

Отрезок

`CreateLineSeg(ID: string; sp, tp: TST3DPoint)` — создать отрезок по двум точкам.

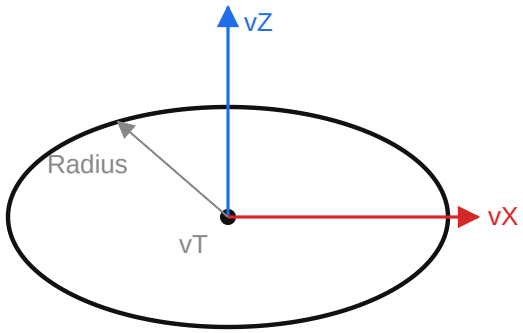
- `ID` — идентификатор сущности;
- `sp` — начальная точка;
- `tp` — конечная точка.



Окружность

`CreateCircle(ID: string; Radius: STFloat; vT, vZ, vX: TST3DPoint)` — создать окружность по точке и радиусу. Построение начинается от `vX` против часовой стрелки.

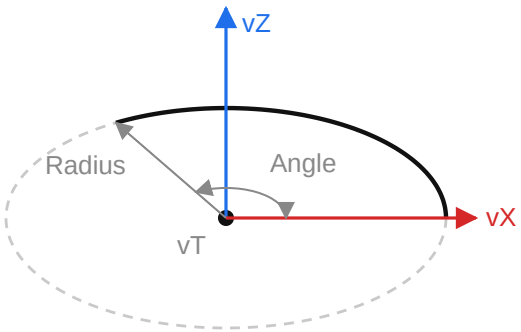
- `ID` — идентификатор сущности;
- `Radius` — радиус;
- `vT` — точка центра;
- `vZ` — ось Z;
- `vX` — ось X.



Дуга окружности

`CreateArc(ID: string; Radius, Angle: STFloat; vT, vZ, vX: TST3DPoint)` — создать дугу окружности. Окружность задаётся по точке и радиусу. Построение начинается от `vX` против часовой стрелки и разворачивается на угол `Angle`.

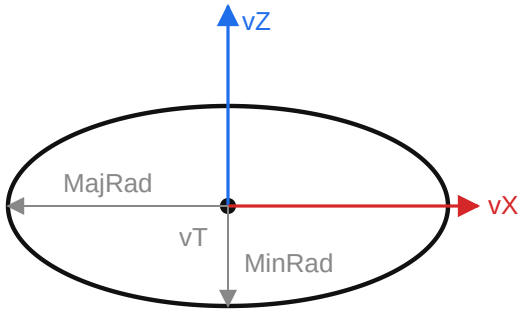
- `ID` — идентификатор сущности;
- `Radius` — радиус;
- `Angle` — угол раскрытия дуги (отсчитывается от `vX`);
- `vT` — точка центра;
- `vZ` — ось Z;
- `vX` — ось X.



Эллипс

`CreateEllipse(ID: string; MajRad, MinRad: STFloat; vT, vZ, vX: TST3DPoint)` — создать эллипс.

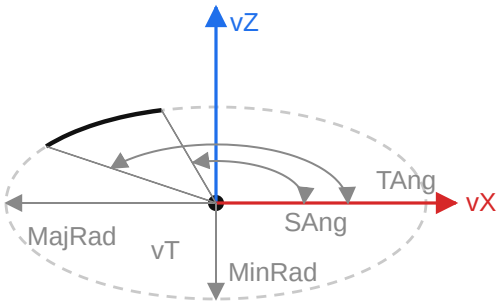
- `ID` — идентификатор сущности;
- `MajRad` — большая полуось (вдоль оси X);
- `MinRad` — малая полуось (вдоль оси Y);
- `vT` — точка центра;
- `vZ` — ось Z;
- `vX` — ось X.



Дуга эллипса

`CreateEllipseArc(ID: string; MajRad, MinRad, SAng, TAng: STFloat; vT, vZ, vX: TST3DPoint)` — создать дугу эллипса по двум углам. Все углы считаются от `vX` против часовой стрелки.

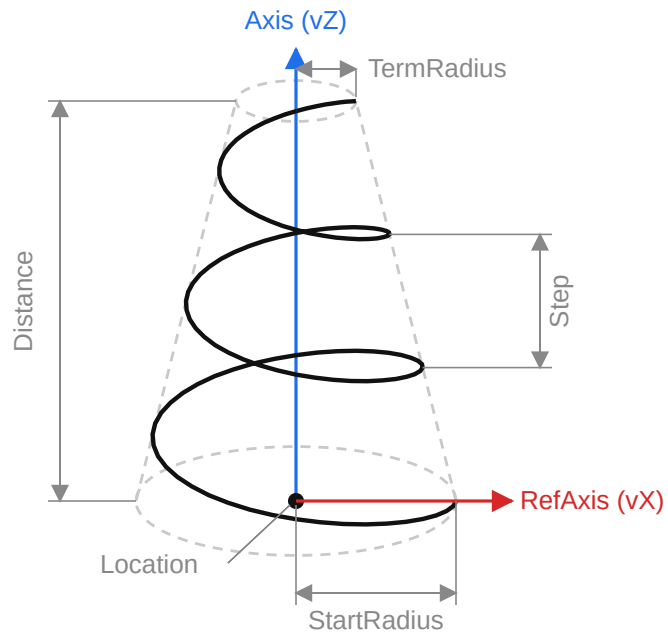
- `ID` — идентификатор сущности;
- `MajRad` — большая полуось (вдоль оси X);
- `MinRad` — малая полуось (вдоль оси Y);
- `SAng` — начальный угол дуги;
- `TAng` — конечный угол дуги;
- `vT` — точка центра;
- `vZ` — ось Z;
- `vX` — ось X.



Коническая спираль

`CreateConicalSpiral(ID: string; Location, Axis, RefAxis: TST3DPoint; StartRadius, TermRadius, Step, Distance: STFloat; CCW: boolean)` — создать коническую спираль вокруг оси `Axis`.

- `ID` — идентификатор сущности;
- `Location` — центр основания (положение, аналог `vT`);
- `Axis` — ось спирали (аналог `vZ`);
- `RefAxis` — направление отсчёта (аналог `vX`);
- `StartRadius` — радиус в начале;
- `TermRadius` — радиус в конце;
- `Step` — шаг (подъём за оборот);
- `Distance` — длина вдоль оси;
- `CCW` — направление вращения (True = против часовой стрелки).



NURBS-кривая

NURBS-кривая формируется последовательностью вызовов: открыть (`StartNurbsCurve`), задать контрольные точки, веса и узлы, закрыть (`CloseNurbsCurve`).

`StartNurbsCurve(ID: string; Degree: integer; IsPeriodic, IsRational, IsClosed: boolean; KnotCount, CPCount: integer)` — начать формирование NURBS-кривой.

- `ID` — идентификатор сущности;
- `Degree` — степень кривой;
- `IsPeriodic` — признак периодичности;
- `IsRational` — признак рациональности;
- `IsClosed` — признак замкнутости;
- `KnotCount` — число узлов;
- `CPCount` — число контрольных точек.

`SetNurbsCurveControlPoint(Index: integer; P: TST3DPoint)` — задать контрольную точку по индексу.

- `Index` — индекс контрольной точки;
- `P` — координаты точки.

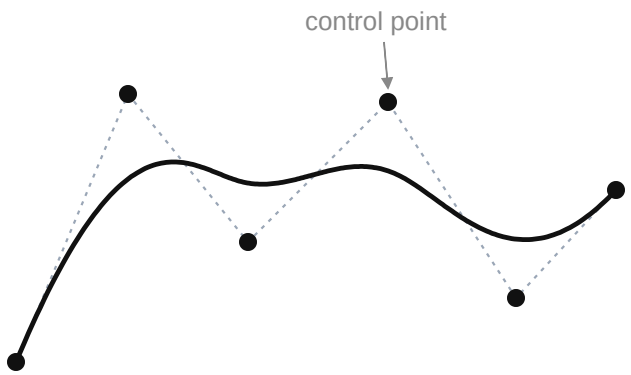
`SetNurbsCurveWeight(Index: integer; w: STFloat)` — задать вес контрольной точки.

- `Index` — индекс контрольной точки;
- `w` — вес.

`SetNurbsCurveKnot(Index: integer; knot: STFloat)` — задать узел.

- `Index` — индекс узла;
- `knot` — значение узла.

`CloseNurbsCurve()` — завершить формирование NURBS-кривой.



Пример импорта NURBS-кривой.

```
// Параметры NURBS-кривой CAD-системы
class CADNurbsParam
{
    public int         Degree;        // степень кривой
    public bool        Periodic;      // периодичность
    public bool        Rational;      // рациональность
    public bool        Closed;        // замкнутость
    public double[]    Knots;         // полный узловой вектор
    public CADPoint[]  ControlPoints; // контрольные точки (вершины CAD)
    public double[]    Weights;       // веса (по одному на контрольную точку)
}

bool SaveNurbsCurve(string id, CADNurbsParam param)
{
    sgr.StartNurbsCurve(id, param.Degree, param.Periodic,
param.Rational, param.Closed,
                        param.Knots.Length, param.ControlPoints.Length);
    try
    {
        // узловой вектор
        for (int i = 0; i < param.Knots.Length; i++)
            sgr.SetNurbsCurveKnot(i, param.Knots[i]);

        // контрольные точки: вес + координаты (приводим к TST3DPoint)
        for (int i = 0; i < param.ControlPoints.Length; i++)
        {
            double w = param.Weights[i];
            TST3DPoint p = ToPoint(param.ControlPoints[i]);

            // для рациональной кривой контрольные точки задаются в однородном виде
(× вес)
            if (param.Rational)
            {
                p.X *= w;
                p.Y *= w;
                p.Z *= w;
            }

            sgr.SetNurbsCurveWeight(i, w);
            sgr.SetNurbsCurveControlPoint(i, p);
        }
    }
    finally
    {
        sgr.CloseNurbsCurve();
    }
```

```
        return true;
    }
```

Составная кривая

Составная кривая собирается последовательностью вызовов: открывается от начальной точки, продолжается участками и завершается вызовом закрытия. Бывает плоской (в локальной плоскости) и пространственной.

Плоская (2D). Строится в локальной плоскости и продолжается отрезками, дугами и кривыми Безье.

`StartCurve2d(ID: string; p: TST2DPoint; vT, vZ, vX: TST3DPoint)` — начать 2D-кривую от точки `p`.

- `ID` — идентификатор сущности;
- `p` — начальная точка (в координатах плоскости);
- `vT` — положение плоскости;
- `vZ` — нормаль;
- `vX` — ось X.

`CutTo2d(p: TST2DPoint)` — продолжить отрезком до точки `p`.

- `p` — конечная точка отрезка.

`ArcTo2d(pc, p: TST2DPoint; rad: STFloat)` — продолжить дугой до точки `p`.

- `pc` — центр дуги;
- `p` — конечная точка;
- `rad` — радиус дуги.

`BezierTo2d(p1, p2, p3: TST2DPoint)` — продолжить кривой Безье 3-го порядка.

- `p1`, `p2`, `p3` — контрольные точки кривой Безье.

`CloseCurve2d(Close: boolean)` — завершить кривую.

- `Close` — при `true` кривая замыкается.



Пространственная (3D). Строится в пространстве и продолжается отрезками.

`StartCurve3d(ID: string; p: TST3DPoint)` — начать 3D-кривую от точки `p`.

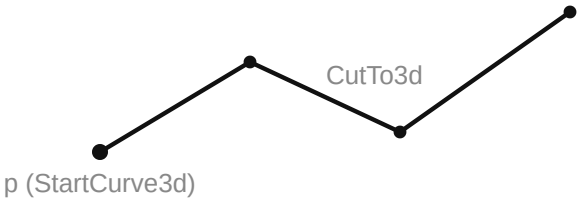
- `ID` — идентификатор сущности;
- `p` — начальная точка.

`CutTo3d(p: TST3DPoint)` — продолжить отрезком до точки `p`.

- `p` — конечная точка отрезка.

`CloseCurve3d(IsClosed: boolean)` — завершить кривую.

- `IsClosed` — при `true` кривая замыкается.



Производные кривые

Производные кривые строятся на основе уже созданных объектов: исходная кривая (а для кривой на поверхности — ещё и поверхность) задаётся своим идентификатором, а результат получает новый `ID`. Такие кривые можно обрезать, инвертировать, перепараметризовать или построить на поверхности. Как и аналитические кривые, результат можно добавить в дерево через `AddEntity` либо использовать как строительный блок (например, кривую петли для обрезки поверхностей). При создании производных объектов исходные объекты не меняются, но при использовании одинаковых `ID` новый объект перезапишет старый.

Виды производных кривых:

- Кривая на поверхности — `CreateCurveOnSurface`.
- Обрезанная кривая — `CreateTrimmedCurve` / `CreateTrimmedCurve2`.
- Инвертированная кривая — `CreateInversedCurve`.
- Кривая пересечения — `StartIntersectionCurve` ⋯ `CloseIntersectionCurve`.
- Граница скругления — `CreateBlendBound`.

Пример сохранения производной кривой

Для сохранения производной кривой сначала необходимо сохранить все опорные объекты и только потом — её саму.

```
bool SaveTrimmedEdge(string edgeId, CADEdge edge)
{
    // сначала сохраняем опорную кривую
    if (!SaveCurve(edge.Curve))
        return false;

    // затем — саму обрезанную кривую по опорной (по двум граничным точкам;
    // координаты CAD-вершин приводим к TST3DPoint)
    return sgr.CreateTrimmedCurve2(edgeId, edge.Curve.Id,
                                   ToPoint(edge.Start.Point),
ToPoint(edge.End.Point));
}
```

Кривая на поверхности

`CreateCurveOnSurface(ID: string; CurveID, SurfaceID: string)` — создать 3D-кривую, лежащую на поверхности. Исходная кривая задаётся в параметрическом пространстве поверхности и проецируется на неё.

- `ID` — идентификатор создаваемой сущности;
- `CurveID` — идентификатор исходной кривой;
- `SurfaceID` — идентификатор поверхности-носителя.

Обрезанная кривая

Часть исходной кривой между двумя границами. Границы задаются одним из двух способов — значениями параметра или точками:

`CreateTrimmedCurve(ID: string; SourceCurveID: string; t1, t2: STFloat)` — обрезать по значениям параметра.

- `ID` — идентификатор создаваемой сущности;
- `SourceCurveID` — идентификатор исходной кривой;
- `t1`, `t2` — начальное и конечное значения параметра.

`CreateTrimmedCurve2(ID: string; SourceCurveID: string; p1, p2: TST3DPoint)` — обрезать по двум точкам.

- `ID` — идентификатор создаваемой сущности;
- `SourceCurveID` — идентификатор исходной кривой;
- `p1`, `p2` — начальная и конечная точки.

Инвертированная кривая

`CreateInversedCurve(ID: string; SourceCurveID: string)` — создать кривую с обратной (инвертированной) параметризацией: направление обхода исходной кривой меняется на противоположное.

- `ID` — идентификатор создаваемой сущности;
- `SourceCurveID` — идентификатор исходной кривой.

Кривая пересечения

Кривая пересечения — это пространственная кривая, лежащая одновременно на двух поверхностях, вдоль линии их пересечения. Обе поверхности должны быть созданы заранее и передаются по идентификаторам.

`StartIntersectionCurve(ID: string; FirstSurfID, SecondSurfID: string; L1, L2: TSTLimitType; BaseParam, BaseScale: STFloat)` — начать формирование кривой пересечения двух поверхностей.

- `ID` — идентификатор создаваемой сущности;
- `FirstSurfID` — идентификатор первой поверхности;
- `SecondSurfID` — идентификатор второй поверхности;
- `L1` — тип начальной границы (см. `TSTLimitType`);
- `L2` — тип конечной границы (см. `TSTLimitType`);
- `BaseParam` — параметр первой точки, лежащей на кривой пересечения;
- `BaseScale` — масштаб параметризации.

`AddIntersectionCurvePoint(Position: TST3DPoint)` — добавить точку пересечения.

- `Position` — координаты точки.

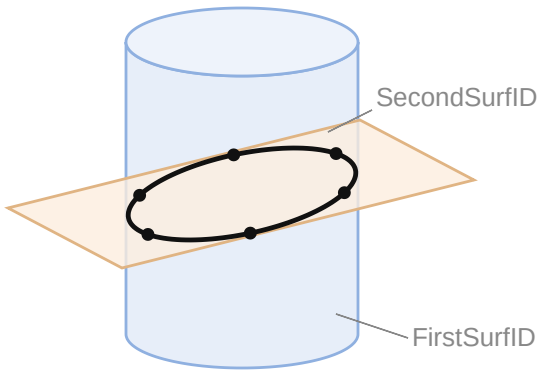
`CloseIntersectionCurve()` — завершить формирование кривой пересечения.

Перечисление `TSTLimitType` задаёт тип границы на концах кривой пересечения (`L1` — начало, `L2` — конец) — как именно кривая ограничивается с каждой стороны:

Значение	Тип границы	Описание
`H`	вспомогательный вектор	Конец задаётся вспомогательным («help») вектором направления.
`T`	терминатор	Кривая обрывается в явно заданной точке-терминаторе; для такой границы в кривую добавляют граничную точку методом `AddIntersectionCurvePoint`.
`L`	произвольная граница	Произвольное ограничение конца кривой.

Значение	Тип границы	Описание
`B`	граница направляющей	Конец ограничен краем области направляющей (spine).
`U`	не задана	Граница не определена (ограничение отсутствует).

Эти же типы границ используются и при создании поверхности скругления (`CreateBlendedSurface`, см. Поверхность скругления).



Пример импорта кривой пересечения. Предполагается, что обе поверхности уже сохранены, а точки кривой и параметры границ получены из CAD. Кривая пишется как последовательность точек; для границ типа `T` (терминаторов) дополнительно добавляются явные граничные точки.

```
// Кривая пересечения CAD-системы: две опорные поверхности + точки
class CADIntersectionCurve
{
    public string       Id;         // идентификатор кривой в CAD
    public string       Surf1Id;    // идентификатор первой поверхности
(уже сохранена)
    public string       Surf2Id;    // идентификатор второй поверхности
(уже сохранена)
    public TSTLimitType StartLim;   // тип начальной границы
    public TSTLimitType EndLim;     // тип конечной границы
    public double       BaseParam;  // параметр первой точки на кривой
    public double       BaseScale;  // масштаб параметризации
    public CADPoint[]   Points;     // точки кривой пересечения
    public CADPoint     StartPoint; // граничная точка (для границы типа T)
    public CADPoint     EndPoint;   // граничная точка (для границы типа T)
}

bool SaveIntersectionCurve(CADIntersectionCurve curve)
```

```
{
    sgr.StartIntersectionCurve(curve.Id, curve.Surf1Id, curve.Surf2Id,
                               curve.StartLim, curve.EndLim,
curve.BaseParam, curve.BaseScale);
    try
    {
        // координаты CAD-точек приводим к TST3DPoint
        if (curve.StartLim == TSTLimitType.T)
            sgr.AddIntersectionCurvePoint(ToPoint(curve.StartPoint));

        foreach (CADPoint v in curve.Points)
            sgr.AddIntersectionCurvePoint(ToPoint(v));

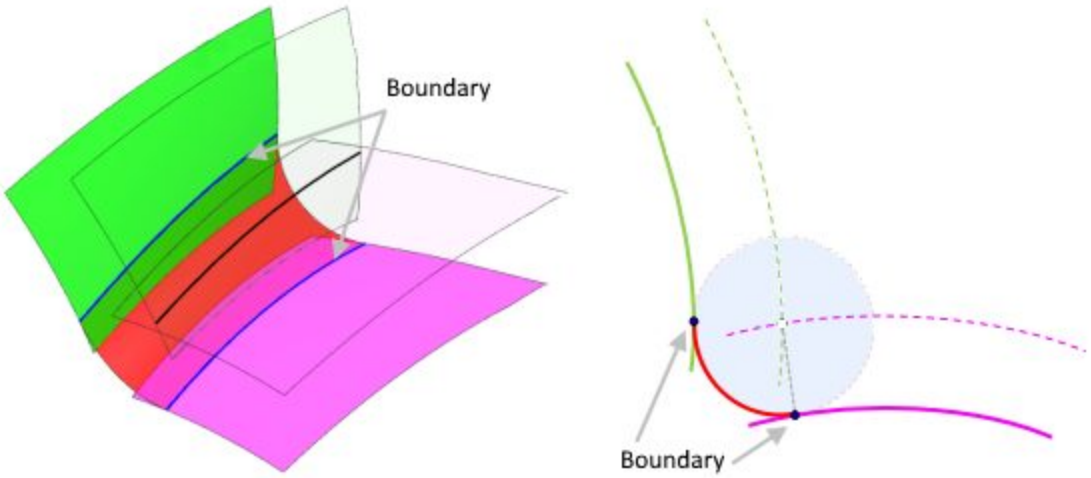
        if (curve.EndLim == TSTLimitType.T)
            sgr.AddIntersectionCurvePoint(ToPoint(curve.EndPoint));
    }
    finally
    {
        sgr.CloseIntersectionCurve();
    }
    return true;
}
```

Граница скругления

В формате Parasolid XT это вспомогательная (конструкционная) поверхность `blend_bound`, заданная неявно так, что она пересекает соответствующую опорную поверхность скругления вдоль искомой граничной линии и ортогональна и скруглению, и этой поверхности вдоль неё. Собственной параметризации она не имеет и на практике представляется кривой пересечения, поэтому в нашей модели создаётся как кривая. Опорная поверхность, которой соответствует граница, выбирается индексом `Boundary` — это `surface[1 - Boundary]` из пары опорных поверхностей скругления `Blend`.

`CreateBlendBound(ID, Blend: string; Boundary: integer)` — граница скругления: кривая, вдоль которой поверхность скругления (см. Поверхность скругления) касается одной из своих опорных поверхностей.

- `ID` — идентификатор создаваемой сущности;
- `Blend` — идентификатор поверхности скругления (`ID` из `CreateBlendedSurface`, см. Поверхность скругления);
- `Boundary` — индекс опорной поверхности скругления (`0` или `1`).



Параметризация кривых

Методы изменяют параметризацию уже созданной кривой (аналитической или производной) по её идентификатору; новый объект при этом не создаётся.

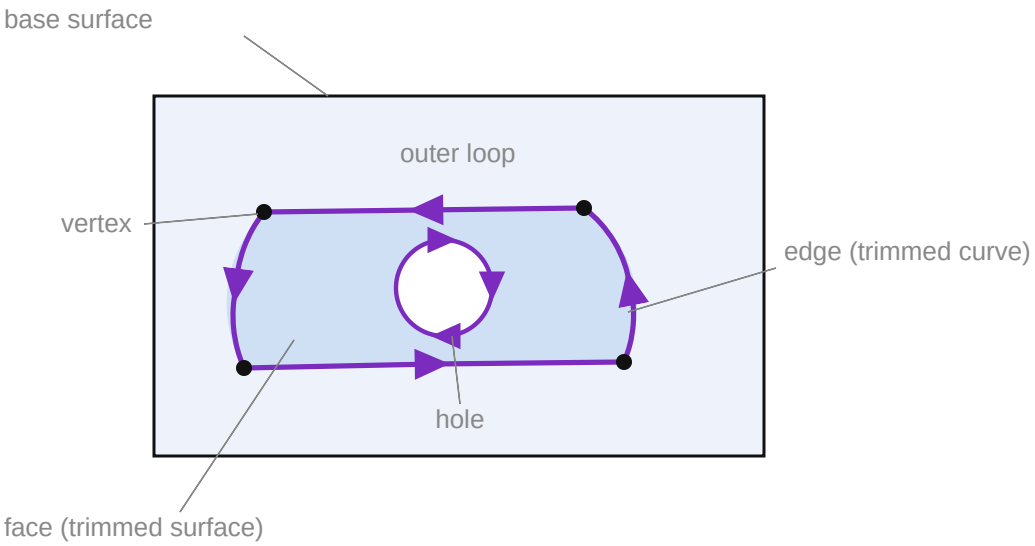
Метод	Описание
`SetCurveParametrization(CurveID: string; TMin, TMax: STFloat)`	Задать новые границы диапазона параметра существующей кривой. `CurveID` — идентификатор кривой; `TMin`, `TMax` — новые минимальное и максимальное значения параметра.
`SetCurveDomain(ID: string; SrcTmin, SrcTmax, PrxTmin, PrxTmax: STFloat)`	Перепараметризовать кривую: задать соответствие между старым (`Src*`) и новым (`Prx*`) диапазонами параметра. `ID` — идентификатор кривой; `SrcTmin`, `SrcTmax` — старые границы диапазона; `PrxTmin`, `PrxTmax` — новые границы диапазона.

Грань (обрезанная поверхность)

Из поверхности можно вырезать ограниченный участок — грань (обрезанную поверхность) — и добавить его в геометрическое дерево как самостоятельный объект, либо как часть оболочки. Грань состоит из *базовой поверхности* и одного или нескольких *контуров (петель)*, которые ограничивают вырезаемую область. Тем самым грань связывает два вида геометрии: несущую поверхность (её строят методами главы Поверхности) и кривые рёбер контура вместе с их вершинами (глава Кривые и точки).

Как это устроено.

1. Базовая поверхность. Грань строится поверх ранее сохранённой поверхности (любой из Поверхности). Она открывается вызовом `StartFace(ID, SurfaceID, NormalOrientation)`, где `SurfaceID` — идентификатор этой поверхности, а `NormalOrientation` задаёт, совпадает ли нормаль грани с нормалью поверхности.
2. Контур (петля). Это замкнутая последовательность рёбер, идущая по границе вырезаемой области; открывается `StartFaceLoop` и закрывается `CloseFaceLoop`. Контуров может быть несколько: первый, как правило, — внешняя граница, остальные — отверстия внутри грани.
3. Направление обхода петли. Оставляемая область (тело грани) должна находиться слева от направления обхода (если смотреть навстречу нормали грани). Поэтому внешний контур обходится *против часовой стрелки*, а контуры-отверстия — *по часовой*. Направление задаётся как порядком добавления рёбер, так и флагом `Orientation` у каждого ребра (совпадает ли направление обхода с направлением опорной кривой).
4. Ребро = обрезанная кривая между двумя вершинами. Концы ребра — *вершины* — создаются как 3D-точки вызовом `CreatePoint(ID, P)` (на них `AddFaceEdge3d` ссылается по идентификатору). Геометрия самого ребра — обрезанная кривая: берётся опорная 3D-кривая (отрезок, дуга, NURBS, кривая пересечения и т.п., уже сохранённая в SGF) и усекается между концевыми точками ребра вызовом `CreateTrimmedCurve2(ID, SourceCurveID, p1, p2)` (для прямого ребра можно сразу `CreateLineSeg`). Если ребро направлено против опорной кривой, кривую предварительно разворачивают `CreateInversedCurve` либо передают `Orientation = false`. Готовое ребро добавляется в текущую петлю: `AddFaceEdge3d` — для пространственной кривой, `AddFaceEdge2d` — для кривой, заданной в параметрах UV поверхности (см. `CreateCurveOnSurface`).
5. Завершение и добавление в дерево. Грань закрывается `CloseFace` (возвращает признак успешного «замыкания»). Чтобы обрезанная поверхность появилась в геометрическом дереве как самостоятельный объект, её регистрируют вызовом `AddEntity(ID, EntityName)` — где `ID` совпадает с идентификатором грани из `StartFace`.



arrows: loop direction (face interior on the left)

`StartFace(ID, SurfaceID: string; NormalOrientation: boolean)` — начать грань на основе ранее сохранённой поверхности.

- `ID` — идентификатор создаваемой грани;
- `SurfaceID` — идентификатор опорной (базовой) поверхности;
- `NormalOrientation` — совпадает ли нормаль грани с нормалью базовой поверхности (`true` — совпадает, `false` — противоположна).

`StartFaceLoop()` — начать контур (петлю) грани.

`AddFaceEdge2d(CurveID: string; Orientation: boolean; svID, tvID: string)` — добавить ребро в текущий контур по кривой, заданной в параметрах UV поверхности (см. Кривая на поверхности).

- `CurveID` — идентификатор опорной кривой ребра;
- `Orientation` — совпадает ли направление обхода контура с направлением кривой (`false` — кривая проходится в обратную сторону);
- `svID` — идентификатор начальной вершины ребра;
- `tvID` — идентификатор конечной вершины ребра.

`AddFaceEdge3d(CurveID: string; Orientation: boolean; svID, tvID: string)` — то же, что `AddFaceEdge2d`, но ребро задано пространственной (3D) кривой; `svID` и `tvID` ссылаются на вершины — 3D-точки, созданные вызовом `CreatePoint`.

- `CurveID` — идентификатор опорной 3D-кривой ребра;
- `Orientation` — совпадает ли направление обхода контура с направлением кривой;
- `svID` — идентификатор начальной вершины ребра;

- `tvID` — идентификатор конечной вершины ребра.

`CloseFaceLoop()` — завершить текущий контур.

`CloseFace()` — завершить грань (возвращает признак успешного замыкания).

Цвет грани. Цвет грани задаётся так же, как для прочих объектов — вызовом `SetCurrentColor` перед `StartFace`; о действии текущего цвета (состояние, наследование, формат значения) см. Кривые и точки.

Пример: импорт грани CAD-модели в SGF. Метод `SaveTrimmedFace` получает грань `CADFace`, сохраняет её несущую поверхность, а затем открывает грань поверх неё и обходит её контуры.

```
bool SaveTrimmedFace(CADFace face)
{
    if (!SaveSurface(face.Surface))
        return false;

    // цвет грани задаётся перед StartFace — по аналогии с кривыми (см. «Кривые
и точки»)
    sgr.SetCurrentColor(face.Color);

    sgr.StartFace(face.Id, face.Surface.Id, face.SameSense);
    try
    {
        for (int i = 0; i < face.Loops.Length; i++)
        {
            CADLoop loop = face.Loops[i];
            sgr.StartFaceLoop();
            try
            {
                // порядок рёбер задаёт направление обхода:
                // внешний контур — против часовой, отверстия — по часовой
                for (int j = 0; j < loop.Edges.Length; j++)
                {
                    // составной идентификатор ребра: грань.луп.ребро
                    string edgeId = $"{face.Id}.{i}.{j}";
                    SaveFaceEdge(edgeId, loop.Edges[j]);
                }
            }
            finally
            {
                sgr.CloseFaceLoop();
            }
```

```
        }
    }
    finally
    {
        sgr.CloseFace();
    }
    return true;
}

// Сохранение одного ребра контура
bool SaveFaceEdge(string edgeId, CADEdge edge)
{
    if (!SaveCurve(edge.Curve))
        return false;

    // геометрия ребра — опорная кривая, обрезанная между вершинами (координаты
приводим к TST3DPoint)
    sgr.CreateTrimmedCurve2(edgeId, edge.Curve.Id,
                            ToPoint(edge.Start.Point), ToPoint(edge.End.Point));

    // ребро в петлю (3D); SameSense — совпадает ли обход с направлением кривой
    return sgr.AddFaceEdge3d(edgeId, edge.SameSense, edge.Start.Id, edge.End.Id);
}
```

Вспомогательные методы `SaveSurface(CADSurface)` и `SaveCurve(CADCurve)` выбирают вызов по типу геометрии (плоскость, цилиндр, NURBS, отрезок, дуга и т.д. — см. Поверхности–Кривые и точки), записывают её в SGF и возвращают признак успеха; идентификатор берётся из самого объекта (поле `Id`).

Поверхности

Поверхность — это несущая геометрия грани: из полной (нередко бесконечной) поверхности грань вырезает ограниченный участок своими контурами (см. Грань (обрезанная поверхность)). В этой главе собраны методы построения поверхностей: аналитические примитивы (плоскость, цилиндр, конус, сфера, тор), NURBS-поверхность и полигональная сетка, производные поверхности (вращения, выдавливания, скругления, эквидистанта и др.) и их параметризация.

Методы построения поверхностей сгруппированы в трёх разделах:

- Аналитические поверхности — плоскость, цилиндр, конус, сфера, тор, NURBS-поверхность и полигональная сетка.
- Производные поверхности — поверхности вращения, выдавливания, транспонированная UV, поверхность скругления и эквидистанта.
- Параметризация поверхностей — изменение параметрического домена уже созданной поверхности (`SetSurfaceUVDomain`).

Аналитические поверхности

Поверхности создаются под уникальным `ID`. У аналитических примитивов (плоскость, цилиндр, конус, сфера, тор) расположение задаётся одинаково тройкой: `Location` — положение (`vT`), `Axis` — ось Z (`vZ`), `RefAxis` — ось X (`vX`); ось Y вычисляется как векторное произведение `Axis × RefAxis`. Помимо аналитических примитивов в эту группу входят NURBS-поверхность и полигональная сетка (Mesh).

Часть аналитических поверхностей математически бесконечны — плоскость безгранична, а цилиндр и конус не ограничены по оси. Поэтому при создании их обязательно нужно ограничивать: у плоскости область задаётся прямоугольником `xMin…xMax` × `yMin…yMax`, у цилиндра и конуса — границами по оси `Hmin…Hmax` (и при необходимости угловым сектором `Amin…Amax`). Если CAD-система не предоставляет явных границ, разумно ограничить поверхность габаритами всей модели. (Сфера и тор замкнуты и в дополнительном ограничении не нуждаются.)

Примитивы аналитических поверхностей:

- Плоскость — `CreatePlaneP`.
- Цилиндр — `CreateCylinderP`.
- Конус — `CreateConeP`.
- Сфера — `CreateSphereP`.
- Тор — `CreateTorusP`.
- NURBS-поверхность — `StartNurbsSurface` ⋯ `CloseNurbsSurface`.
- Полигональная сетка — `StartMesh` ⋯ `CloseMesh`.

Пример сохранения аналитической поверхности

`SaveSurface` — диспетчер: получает поверхность CAD-системы и, в зависимости от её типа, вызывает соответствующий обработчик (`SavePlane`, `SaveCylinder`, ⋯). Каждый обработчик пишет геометрию в SGF подходящим вызовом из Поверхности и возвращает флаг успеха. Ниже показаны сам диспетчер и один из обработчиков (`SavePlane`); остальные устроены аналогично.

```
// Сохраняет поверхность любого типа; возвращает признак успеха (Id уже задан
в CADSurface)
bool SaveSurface(CADSurface surf)
{
    switch (surf.Type)
    {
        case "Plane":    return SavePlane(surf.Id, surf.Param);
        case "Cylinder": return SaveCylinder(surf.Id, surf.Param);    //
обработчики остальных
```

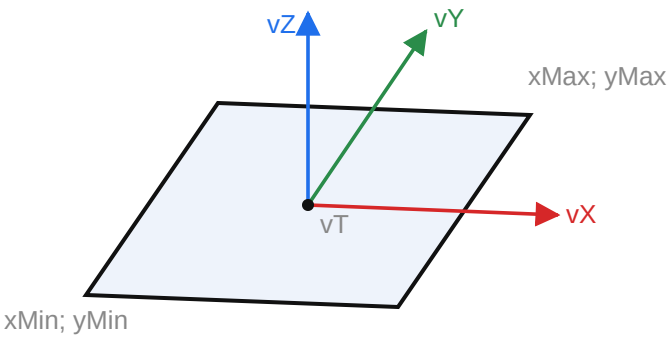
```
        case "Cone":     return SaveCone(surf.Id, surf.Param);        // типов
устроены так же
        case "Sphere":   return SaveSphere(surf.Id, surf.Param);
        case "Torus":    return SaveTorus(surf.Id, surf.Param);
        case "Nurbs":    return SaveNurbsSurface(surf.Id, surf.Param);
        case "Mesh":     return SaveMesh(surf.Id, surf.Param);
        // … прочие типы поверхностей (см. «Поверхности»)
        default:         return false;                  // тип не поддерживается
    }
}

// Один из обработчиков — сохранение плоскости
bool SavePlane(string id, CADPlaneParam param)
{
    // точки-направления CAD приводим к TST3DPoint (Xmin/Ymin/Xmax/Ymax — числа)
    return sgr.CreatePlaneP(id, ToPoint(param.Location), ToPoint(param.Axis),
ToPoint(param.RefAxis),
                            param.Xmin, param.Ymin, param.Xmax, param.Ymax);
}
```

Плоскость

`CreatePlaneP(ID: string; Location, Axis, RefAxis: TST3DPoint; xMin, yMin, xMax, yMax: STFloat)` — создать плоскость, ограниченную прямоугольной областью.

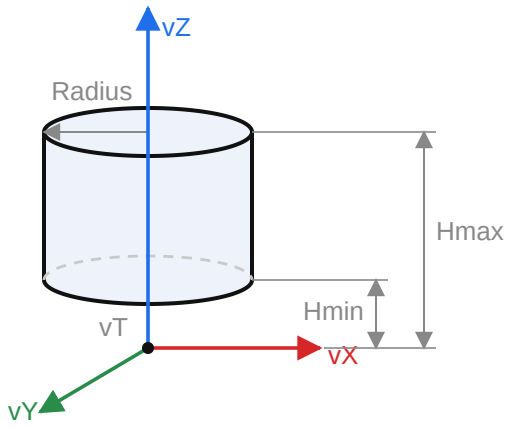
- `ID` — идентификатор сущности;
- `Location` — положение (`vT`);
- `Axis` — ось Z (`vZ`);
- `RefAxis` — ось X (`vX`);
- `xMin`, `yMin` — минимальные X и Y габаритного прямоугольника;
- `xMax`, `yMax` — максимальные X и Y габаритного прямоугольника.



Цилиндр

`CreateCylinderP(ID: string; Location, Axis, RefAxis: TST3DPoint; Radius, Hmin, Hmax, Amin, Amax: STFloat)` — создать цилиндрическую поверхность. Ось идёт вдоль вектора `Axis`.

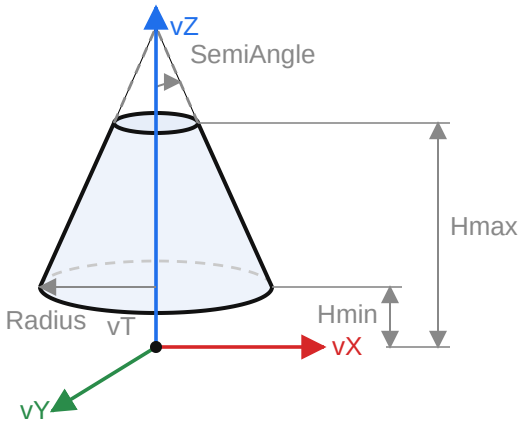
- `ID` — идентификатор сущности;
- `Location` — положение (`vT`);
- `Axis` — ось Z (`vZ`);
- `RefAxis` — ось X (`vX`);
- `Radius` — радиус;
- `Hmin`, `Hmax` — границы по высоте (вдоль оси);
- `Amin`, `Amax` — границы углового сектора (полный цилиндр — от `0` до `2π`).



Конус

`CreateConeP(ID: string; Location, Axis, RefAxis: TST3DPoint; Radius, SemiAngle, Hmin, Hmax, Amin, Amax: STFloat)` — создать коническую поверхность.

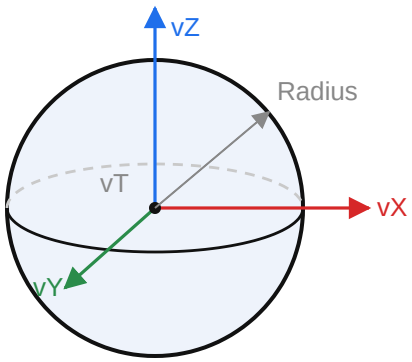
- `ID` — идентификатор сущности;
- `Location` — положение (`vT`);
- `Axis` — ось Z (`vZ`);
- `RefAxis` — ось X (`vX`);
- `Radius` — радиус (в плоскости `Location`);
- `SemiAngle` — половинный угол при вершине конуса;
- `Hmin`, `Hmax` — границы по высоте (вдоль оси);
- `Amin`, `Amax` — границы углового сектора (полный конус — от `0` до `2π`).



Сфера

`CreateSphereP(ID: string; Location, Axis, RefAxis: TST3DPoint; Radius: STFloat)` — создать сферическую поверхность.

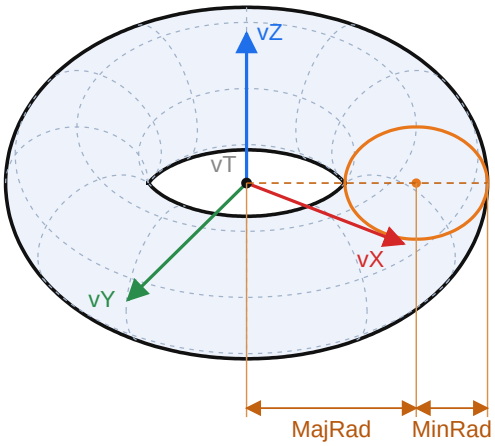
- `ID` — идентификатор сущности;
- `Location` — центр (`vT`);
- `Axis` — ось Z (`vZ`);
- `RefAxis` — ось X (`vX`);
- `Radius` — радиус.



Top

`CreateTorusP(ID: string; Location, Axis, RefAxis: TST3DPoint; MajRad, MinRad: STFloat)` — создать тороидальную поверхность.

- `ID` — идентификатор сущности;
- `Location` — центр (`vT`);
- `Axis` — ось Z (`vZ`), ось тора;
- `RefAxis` — ось X (`vX`);
- `MajRad` — большой радиус (от оси до центра трубы);
- `MinRad` — малый радиус (радиус трубы).



NURBS-поверхность

NURBS-поверхность формируется последовательностью вызовов: открыть (`StartNurbsSurface`), задать узлы по U и V, контрольные точки и веса в сетке `i×j`, закрыть (`CloseNurbsSurface` или `CloseNurbsSurfaceB`).

`StartNurbsSurface(ID: string; K1, K2, UDegree, VDegree: integer; UClosed, VClosed, UPeriodic, VPeriodic, Rational: boolean)` — начать формирование NURBS-поверхности.

- `ID` — идентификатор сущности;
- `K1` — показатель числа контрольных точек по U: `K1 = UKnotCount − UDegree − 2`;
- `K2` — показатель числа контрольных точек по V: `K2 = VKnotCount − VDegree − 2`;
- `UDegree`, `VDegree` — степени по U и V;
- `UClosed`, `VClosed` — признаки замкнутости по U и V;
- `UPeriodic`, `VPeriodic` — признаки периодичности по U и V;
- `Rational` — признак рациональности.

`SetNurbsSurfaceUKnot(i: integer; Value: STFloat)` — задать U-узел.

- `i` — номер узла;
- `Value` — значение узла.

`SetNurbsSurfaceVKnot(i: integer; Value: STFloat)` — задать V-узел.

- `i` — номер узла;
- `Value` — значение узла.

`SetNurbsSurfaceControlPoint(i, j: integer; P: TST3DPoint)` — задать контрольную точку в сетке.

- `i`, `j` — позиция контрольной точки в сетке;
- `P` — координаты точки.

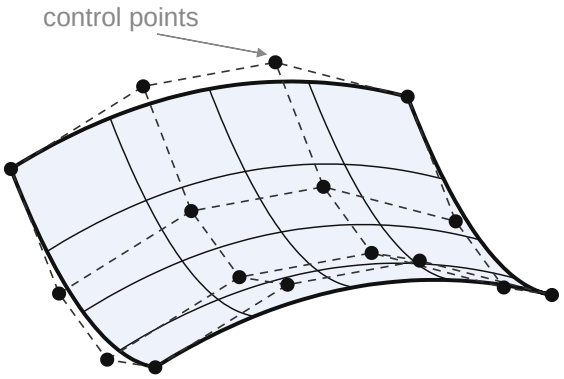
`SetNurbsSurfaceWeight(i, j: integer; Value: STFloat)` — задать вес контрольной точки.

- `i`, `j` — позиция контрольной точки в сетке;
- `Value` — вес.

`CloseNurbsSurface()` — завершить формирование поверхности.

`CloseNurbsSurfaceB(UMin, UMax, VMin, VMax: STFloat)` — завершить формирование с явным заданием диапазонов параметров.

- `UMin`, `UMax` — границы параметра по U;
- `VMin`, `VMax` — границы параметра по V.



Пример импорта NURBS-поверхности.

```
// Параметры NURBS-поверхности CAD-системы
class CADNurbsSurfParam
{
    public int         UDegree, VDegree;      // степени по U и V
    public bool        UClosed, VClosed;      // замкнутость по U и V
    public bool        UPeriodic, VPeriodic;  // периодичность по U и V
    public bool        Rational;              // рациональность
    public double[]    UKnots, VKnots;        // полные узловые векторы по U и V
    public CADPoint[,] ControlPoints;         // сетка контрольных точек
    public double[,]   Weights;               // веса (по одному на
контрольную точку)
}

bool SaveNurbsSurface(string id, CADNurbsSurfParam param)
{
    // показатели числа контрольных точек: K = KnotCount - Degree - 2
    int k1 = param.UKnots.Length - param.UDegree - 2;
    int k2 = param.VKnots.Length - param.VDegree - 2;

    sgr.StartNurbsSurface(id, k1, k2, param.UDegree, param.VDegree,
                          param.UClosed, param.VClosed, param.UPeriodic,
param.VPeriodic, param.Rational);
    try
    {
        // узловые векторы по U и V
        for (int i = 0; i < param.UKnots.Length; i++)
            sgr.SetNurbsSurfaceUKnot(i, param.UKnots[i]);
        for (int i = 0; i < param.VKnots.Length; i++)
            sgr.SetNurbsSurfaceVKnot(i, param.VKnots[i]);

        // контрольные точки и веса в сетке (k1+1) × (k2+1)
```

```
            for (int i = 0; i <= k1; i++)
                for (int j = 0; j <= k2; j++)
                {
                    double w = param.Weights[i, j];
                    TST3DPoint p = ToPoint(param.ControlPoints[i, j]);

                    // для рациональной поверхности — в однородном виде (× вес)
                    if (param.Rational)
                    {
                        p.X *= w;
                        p.Y *= w;
                        p.Z *= w;
                    }

                    sgr.SetNurbsSurfaceWeight(i, j, w);
                    sgr.SetNurbsSurfaceControlPoint(i, j, p);
                }
        }
        finally
        {
            sgr.CloseNurbsSurface();
        }
        return true;
    }
```

Полигональная сетка

Полигональная сетка формируется множеством треугольников, каждый из которых состоит из трёх вершин.

`StartMesh(ID: string)` — начать формирование сетки.

- `ID` — идентификатор сущности.

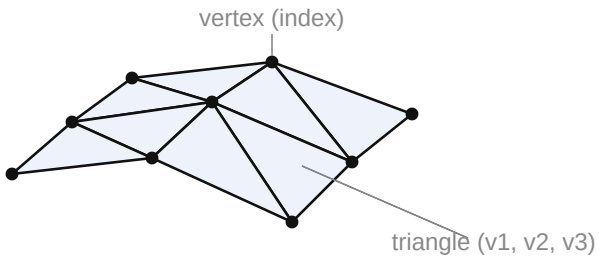
`AddMeshVertex(index: integer; p: TST3DPoint)` — добавить вершину.

- `index` — индекс вершины (используется в `AddMeshTriangle`);
- `p` — координаты вершины.

`AddMeshTriangle(v1, v2, v3: integer)` — добавить треугольник.

- `v1`, `v2`, `v3` — индексы трёх вершин.

`CloseMesh()` — завершить формирование сетки.



Пример импорта полигональной сетки. Каждая вершина добавляется один раз — по своему индексу (повторные индексы из разных треугольников пропускаются), затем по этим индексам добавляются треугольники.

```
// Параметры полигональной сетки CAD-системы
class CADMeshParam
{
    public CADPoint[] Vertices;  // вершины (индекс = позиция в массиве)
    public int[][]    Triangles; // треугольники: тройки индексов вершин
}

bool SaveMesh(string id, CADMeshParam meshParam)
{
    sgr.StartMesh(id);
    try
    {
        var added = new HashSet<int>();
        foreach (int[] tri in meshParam.Triangles)
```

```
        {
            // каждую вершину добавляем один раз — по её индексу (приводим
к TST3DPoint)
            foreach (int vIndex in tri)
                if (added.Add(vIndex))   // вершину приводим к TST3DPoint
                    sgr.AddMeshVertex(vIndex, ToPoint(meshParam.Vertices[vIndex]));

            sgr.AddMeshTriangle(tri[0], tri[1], tri[2]);
        }
    }
    finally
    {
        sgr.CloseMesh();
    }
    return true;
}
```

Производные поверхности

Производные поверхности строятся на основе уже созданных объектов — кривых (образующих, спайнов) и других поверхностей: исходные объекты задаются своими идентификаторами, а результат получает новый `ID`.

Виды производных поверхностей:

- Поверхность вращения — `CreateRevSurface120`.
- Поверхность выдавливания — `CreateTabSurface122`.
- Транспонированная UV-поверхность — `CreateTransposedUVSurface`.
- Поверхность скругления — `CreateBlendedSurface`.
- Эквидистантная поверхность — `CreateOffsetSurf`.

Пример сохранения производной поверхности

Для сохранения производной поверхности сначала необходимо сохранить все опорные объекты и только потом — её саму.

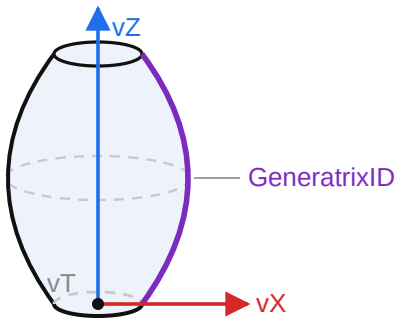
```
bool SaveOffsetSurface(string id, CADOffsetParam param)
{
    if (!SaveSurface(param.Surface))
        return false;

    return sgr.CreateOffsetSurf(id, param.Surface.Id, param.Offset);
}
```

Поверхность вращения

`CreateRevSurface120(ID: string; Location, Axis: TST3DPoint; GeneratrixID: string; sa, ta: STFloat)` — поверхность вращения образующей кривой вокруг оси. Построение ведётся от оси X против часовой стрелки вокруг `Axis`: от начального угла `sa` до конечного `ta`.

- `ID` — идентификатор создаваемой сущности;
- `Location` — точка на оси вращения (`vT`);
- `Axis` — направление оси вращения (`vZ`);
- `GeneratrixID` — идентификатор образующей кривой;
- `sa` — начальный угол (обычно `0`);
- `ta` — конечный угол (полный оборот — `2π`).



Пример сохранения поверхности вращения.

```
// Параметры поверхности вращения CAD-системы
class CADRevolutionParam
{
    public CADPoint Location;    // точка на оси вращения (vT)
    public CADPoint Axis;        // направление оси вращения (vZ)
    public CADCurve Curve;       // образующая кривая
    public double   StartAngle;  // начальный угол (sa)
    public double   EndAngle;    // конечный угол (ta); полный оборот — 2π
}

bool SaveRevolution(string id, CADRevolutionParam param)
{
    if (!SaveCurve(param.Curve))
        return false;

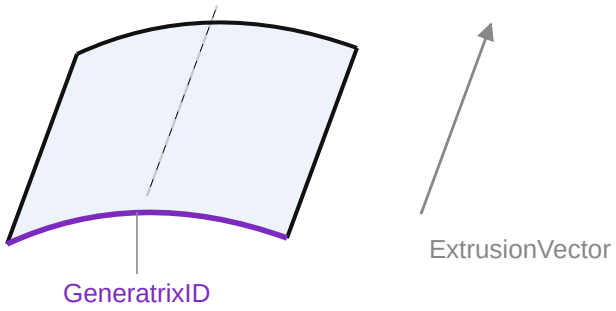
    return sgr.CreateRevSurface120(id, ToPoint(param.Location), ToPoint(param.Axis),
                                   param.Curve.Id, param.StartAngle,
```

```
param.EndAngle);
}
```

Поверхность выдавливания

`CreateTabSurface122(ID: string; GeneratrixID: string; ExtrusionVector: TST3DPoint)` — линейчатая поверхность выдавливания: образующая кривая смещается вдоль вектора.

- `ID` — идентификатор создаваемой сущности;
- `GeneratrixID` — идентификатор образующей кривой;
- `ExtrusionVector` — вектор выдавливания (направление и длина).



Пример сохранения поверхности выдавливания.

```
// Параметры поверхности выдавливания CAD-системы
class CADExtrusionParam
{
    public CADCurve Curve;            // образующая кривая
    public CADPoint ExtrusionVector;  // вектор выдавливания (направление и длина)
}

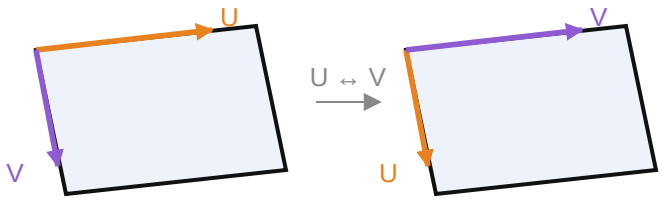
bool SaveExtrusion(string id, CADExtrusionParam param)
{
    if (!SaveCurve(param.Curve))
        return false;

    return sgr.CreateTabSurface122(id, param.Curve.Id,
ToPoint(param.ExtrusionVector));
}
```

Транспонированная UV-поверхность

`CreateTransposedUVSurface(ID: string; SourceSurface: string)` — поверхность с переставленными местами параметрами U и V.

- `ID` — идентификатор создаваемой сущности;
- `SourceSurface` — идентификатор исходной поверхности.



Поверхность скругления

Поверхность точного скругления перекатом шара (rolling ball blend) между двумя поверхностями. Соответствует поверхности `blended_edge` формата Parasolid XT и обычно возникает при скруглении ребра между двумя гранями.

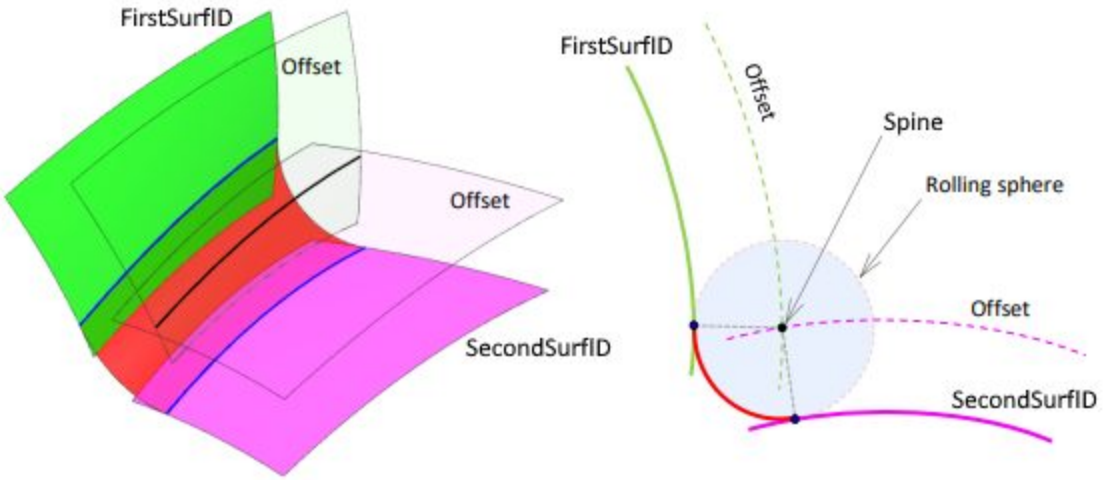
Спайн строится как кривая пересечения двух эквидистант — опорных поверхностей, смещённых на величины `Range1` и `Range2`. Смещение откладывается вдоль единичной нормали поверхности, умноженной на `Range` (и на −1, если ориентация поверхности отрицательна); значения `Range` могут быть и отрицательными. Для скругления постоянного радиуса обе величины равны радиусу шара.

> По возможности скругления моделируются более простыми поверхностями — цилиндрической, тороидальной или сферической, а сложные (переменный радиус, некруговое сечение) — NURBS-поверхностями. Отдельная поверхность скругления (`blended_edge`) применяется только тогда, когда более простое представление невозможно.

`CreateBlendedSurface(ID, FirstSurfID, SecondSurfID, Spine: string; Range1, Range2: STFloat; BlendedType: TSTBlendedType; p1: TST3DPoint; L1: TSTLimitType; p2: TST3DPoint; L2: TSTLimitType)` — создать поверхность скругления.

- `ID` — идентификатор создаваемой сущности;
- `FirstSurfID`, `SecondSurfID` — две опорные поверхности (смежные с исходным ребром), между которыми строится скругление;
- `Spine` — спайн: осевая кривая скругления, путь центра катящегося шара; строится как кривая пересечения двух эквидистант;
- `Range1`, `Range2` — смещения (offsets), применяемые к первой и второй поверхностям соответственно; на рисунке задают эквидистанты «Offset» (пунктирные), пересечение которых и даёт спайн; для постоянного радиуса равны радиусу шара;
- `BlendedType` — тип скругления (`TSTBlendedType`): `R` — перекатом шара (rolling ball); шар на рисунке подписан «Rolling sphere», его радиус равен `Range`, а центр катится по спайну; `E` — «обрыв» (cliff edge), когда одна из поверхностей сама является скруглением с диапазоном `[0, 0]`, а спайн совпадает с кромкой обрыва;
- `p1`, `L1` — начальная точка и тип её границы (`TSTLimitType`, см. Кривая пересечения);
- `p2`, `L2` — конечная точка и тип её границы.

Поля `p1`/`L1` и `p2`/`L2` задаются (тип границы `L`) только в вырожденных случаях — когда спайн периодический, а скругляемое ребро имеет концы (например, спайн эллиптический, а скругление вырождается). В обычном случае границы не ограничены (тип `U`).



`TSTBlendedType` и `TSTLimitType` — это типы SGF, а не CAD-системы. Поэтому в классе параметров используются собственные перечисления Дополнения, а перед вызовом `CreateBlendedSurface` они приводятся к типам SGF (так же, как точки приводятся к `TST3DPoint`).

Пример сохранения поверхности скругления.

```
// собственные перечисления Дополнения (CAD-сторона, без типов SGF)
enum CADBlendKind  { RollingBall, Cliff }              // соответствует R / E
enum CADBorderKind { NotSet, Terminator, ArbLimit }    // соответствует U / T / L

// Параметры поверхности скругления CAD-системы
class CADBlendParam
{
    public CADSurface    Surface1, Surface2;  // две опорные поверхности
    public CADCurve      Spine;               // спайн (осевая кривая скругления)
    public double        Range1, Range2;      // смещения эквидистант (для
постоянного радиуса = радиусу шара)
    public CADBlendKind  Kind;                // тип скругления (CAD-сторона)
    public CADPoint      P1, P2;              // граничные точки (обычно
не используются)
    public CADBorderKind L1, L2;              // типы границ (в обычном
случае NotSet)
}

bool SaveBlendedSurface(string id, CADBlendParam param)
{
    // сначала сохраняем опорные объекты: обе поверхности и спайн
    if (!SaveSurface(param.Surface1) || !SaveSurface(param.Surface2)
|| !SaveCurve(param.Spine))
        return false;
```

```
    // приведение CAD-перечислений к типам SGF через switch
    TSTBlendedType blendedType;
    switch (param.Kind)
    {
        case CADBlendKind.Cliff: blendedType = TSTBlendedType.E; break;   // «обрыв»
        default:                 blendedType = TSTBlendedType.R; break;   //
перекатом шара
    }

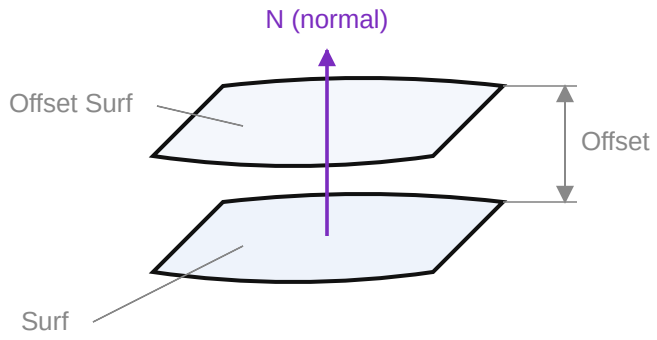
    // поверхность скругления (точки приводятся к TST3DPoint)
    return sgr.CreateBlendedSurface(id, param.Surface1.Id,
param.Surface2.Id, param.Spine.Id,
                                    param.Range1, param.Range2, blendedType,
                                    ToPoint(param.P1), ToLimit(param.L1),
                                    ToPoint(param.P2), ToLimit(param.L2));
}

// функция приведения типа границы (используется для L1 и L2)
TSTLimitType ToLimit(CADBorderKind k)
{
    switch (k)
    {
        case CADBorderKind.Terminator: return TSTLimitType.T;
        case CADBorderKind.ArbLimit:   return TSTLimitType.L;
        default:                       return TSTLimitType.U;   // не задано
    }
}
```

Эквидистантная поверхность

`CreateOffsetSurf(ID, Surf: string; Offset: STFloat)` — создать эквидистантную (смещённую) поверхность. Она строится сдвигом каждой точки исходной поверхности `Surf` вдоль её единичной нормали на расстояние `Offset`. Направление смещения задаётся знаком `Offset` и ориентацией (sense) исходной поверхности.

- `ID` — идентификатор создаваемой сущности;
- `Surf` — идентификатор исходной поверхности;
- `Offset` — знаковое расстояние смещения вдоль нормали (`> 0` — вдоль нормали `N`, `< 0` — против).



Параметризация поверхностей

`SetSurfaceUVDomain(ID: string; SrcUmin, SrcUmax, SrcVmin, SrcVmax, PrxUmin, PrxUmax, PrxVmin, PrxVmax: STFloat; TransposeUV: boolean)` — изменить параметризацию уже созданной поверхности по её идентификатору. Новый объект не создаётся: поверхность оборачивается в прокси, который задаёт линейный пересчёт между двумя диапазонами параметров U/V.

- `ID` — идентификатор поверхности;
- `SrcUmin`, `SrcUmax` / `SrcVmin`, `SrcVmax` — диапазон Src: параметрическое пространство, в котором поверхность «видна» снаружи — в нём заданы координаты контуров грани, ссылающейся на эту поверхность;
- `PrxUmin`, `PrxUmax` / `PrxVmin`, `PrxVmax` — диапазон Prx: пространство, в которое внешний параметр пересчитывается для вычисления самой поверхности (для NURBS — диапазон узлов, *knot-span*);
- `TransposeUV` — при `true` оси U и V меняются местами (например, когда порядок U/V в CAD-системе не совпадает с принятым в SGF — так бывает для тора).

Зачем это нужно. Некоторые CAD-системы хранят контур (loop) грани в одном параметрическом пространстве, а саму несущую поверхность определяют в другом. Если их не согласовать, CAM-система спроецирует контур не туда, и грань построится неверно. `SetSurfaceUVDomain` задаёт линейное соответствие Src↔Prx, и контур проецируется правильно.

Частный случай — продление поверхности: когда параметрический домен грани шире естественного домена поверхности. Так делает импорт из КОМПАС для NURBS-граней: реальный диапазон грани (`MathSurface.ParamU/V`) выходит за пределы узлового диапазона NURBS. Прокси «растягивает» поверхность на весь домен грани, и контур грани проецируется на NURBS корректно. Если же домен грани целиком лежит внутри узлового (обычная «обрезка внутри NURBS», типичный fillet) — перепараметризация не нужна, иначе она лишь исказит геометрию.

Пример: выравнивание параметризации поверхности.

```
// Выровнять параметризацию грани и несущей NURBS-поверхности (по образцу
импорта КОМПАС).
void AlignSurfaceUVDomain(string surfaceId, CADFace face)
{
    CADSurface surf = face.Surface;

    // Src — UV-пространство, в котором заданы контуры грани
    double srcUmin = face.ParamUMin, srcUmax = face.ParamUMax;
    double srcVmin = face.ParamVMin, srcVmax = face.ParamVMax;
```

```
    // Prx — естественный домен поверхности (для NURBS — диапазон узлов)
    double prxUmin = surf.KnotUMin, prxUmax = surf.KnotUMax;
    double prxVmin = surf.KnotVMin, prxVmax = surf.KnotVMax;

    // Если домен грани целиком внутри домена поверхности — это обычная
    // «обрезка внутри NURBS», перепараметризация только исказит геометрию.
    const double eps = 1e-9;
    bool insideU = srcUmin >= prxUmin - eps && srcUmax <= prxUmax + eps;
    bool insideV = srcVmin >= prxVmin - eps && srcVmax <= prxVmax + eps;
    if (insideU && insideV)
        return;

    // Иначе домен грани шире — прокси продлевает поверхность на весь её диапазон
    // (TransposeUV = false: оси U/V не меняем).
    sgr.SetSurfaceUVDomain(surfaceId,
                           srcUmin, srcUmax, srcVmin, srcVmax,
                           prxUmin, prxUmax, prxVmin, prxVmax,
                           false);
}
```

PMI (Product Manufacturing Information)

PMI (Product Manufacturing Information) — аннотации производственной информации: размеры, допуски, резьбы и т.п.

Объект PMI создаётся вызовом `BeginObject_*`, заполняется через свой интерфейс и записывается в SGF вызовом `EndObject`.

Метод	Описание
`BeginObject_PMIThread(CADID: string): IPMIGeomThread`	Создать объект резьбы и вернуть интерфейс `IPMIGeomThread` (см. IPMIGeomThread) для его заполнения. `CADID` — уникальный идентификатор объекта в CAD.
`EndObject(obj: IUnknown): boolean`	Записать заполненный объект (например, полученный из `BeginObject_PMIThread`) в SGF-файл. Возвращает признак успешной записи.

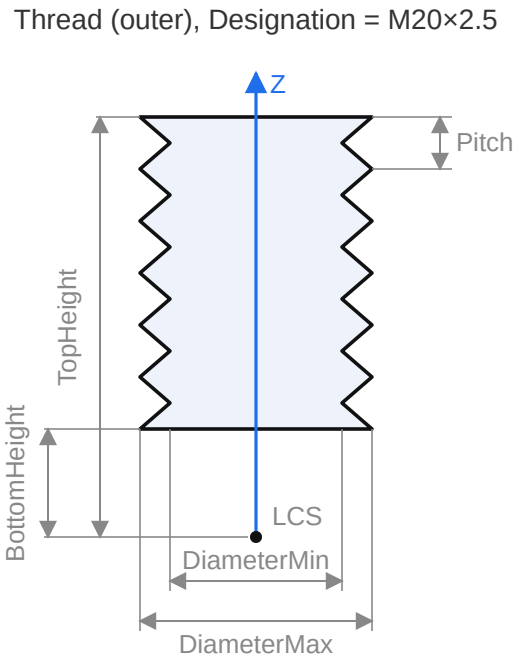
Описание объектов PMI:

- IPMIGeomThread — объект резьбы: параметры, локальная система координат, ассоциативные ссылки на геометрию и пример сохранения.

IPMIGeomThread

Описывает объект резьбы (thread) для PMI. Экземпляр возвращается методом `ISTGeomReceiver.BeginObject_PMIThread` (см. PMI (Product Manufacturing Information)); после заполнения объект записывается в SGF через `EndObject`.

Каждое свойство доступно через пару методов `Get<Имя>` / `Set<Имя>` (например, `GetPitch` / `SetPitch`).



Свойство	Тип	Назначение
`Designation`	`string`	Обозначение резьбы.
`LCS`	`TST3DMatrix`	Локальная система координат резьбы.
`TopHeight`	`STFloat`	Верхняя граница (высота).
`BottomHeight`	`STFloat`	Нижняя граница.
`Pitch`	`STFloat`	Шаг резьбы.
`StartCount`	`integer`	Число заходов.
`DiameterMin`	`STFloat`	Минимальный диаметр.
`DiameterMax`	`STFloat`	Максимальный диаметр.

Свойство	Тип	Назначение
`ConeAngle`	`STFloat`	Угол конуса (для конической резьбы).
`DirectionIsLeft`	`boolean`	Левая резьба.
`IsOuterThread`	`boolean`	Наружная резьба.

Ассоциативные ссылки на геометрию (необязательные). Резьба может ссылаться на исходную геометрию модели, по которой она построена, — по спискам строковых CAD-идентификаторов. Эти ссылки необязательны: резьба корректно сохраняется и без них; они лишь задают ассоциативную связь с геометрией.

Метод	Описание
`AddBaseObjectCADID(Value: string)`	*(необязательно)* Ассоциативная ссылка на базовый объект — несущую поверхность, по которой создана резьба.
`AddInitialBorderCADID(Value: string)`	*(необязательно)* Ассоциативная ссылка на геометрию начальной границы резьбы.
`AddFinalBorderCADID(Value: string)`	*(необязательно)* Ассоциативная ссылка на геометрию конечной границы резьбы.

Порядок заполнения. Объект резьбы создаётся вызовом `BeginObject_PMIThread` (он возвращает интерфейс `IPMIGeomThread`), заполняется сеттерами параметров и записывается в SGF вызовом `EndObject`. Положение резьбы задаёт локальная система координат `LCS` (`TST3DMatrix`), а её протяжённость вдоль оси — `TopHeight` / `BottomHeight`. Ассоциативные ссылки на исходную геометрию (`AddBaseObjectCADID`, `AddInitialBorderCADID`, `AddFinalBorderCADID`) необязательны — их можно не задавать.

Пример сохранения резьбы (PMI).

```
// Сохранить резьбу (PMI) по данным CAD-системы
bool SaveThread(CADThread thread)
{
    // открыть объект резьбы и получить интерфейс для заполнения
    IPMIGeomThread pmi = sgr.BeginObject_PMIThread(thread.Id);
    if (pmi == null)
        return false;
```

```
    // параметры резьбы
    pmi.SetDesignation(thread.Designation);     // обозначение, напр. "M12x1.5"
    pmi.SetPitch(thread.Pitch);                 // шаг
    pmi.SetStartCount(thread.StartCount);       // число заходов
    pmi.SetDiameterMax(thread.OuterDiameter);   // наружный диаметр
    pmi.SetDiameterMin(thread.InnerDiameter);   // внутренний диаметр
    pmi.SetConeAngle(thread.ConeAngle);         // угол конуса (0
для цилиндрической)
    pmi.SetDirectionIsLeft(thread.IsLeft);      // левая резьба
    pmi.SetIsOuterThread(thread.IsOuter);       // наружная резьба

    // положение и протяжённость
    pmi.SetLCS(ToMatrix(thread.Placement));     // локальная система координат
→ TST3DMatrix
    pmi.SetTopHeight(thread.TopHeight);
    pmi.SetBottomHeight(thread.BottomHeight);

    // необязательные ассоциативные ссылки на исходную геометрию — только если
они заданы
    if (thread.BaseSurfaceId != null) pmi.AddBaseObjectCADID(thread.BaseSurfaceId);
// несущая поверхность
    if (thread.StartBorderId != null)
pmi.AddInitialBorderCADID(thread.StartBorderId);  // начальная граница
    if (thread.EndBorderId   != null) pmi.AddFinalBorderCADID(thread.EndBorderId);
// конечная граница

    // записать заполненный объект в SGF
    return sgr.EndObject(pmi);
}
```

Устаревшие и неиспользуемые методы

> Устарело / не используется. Методы ниже не применяются и оставлены только для справки и обратной совместимости. В новых Дополнениях вызывать их не нужно.

Сгруппированы по областям — в том же порядке, что и действующие главы: файл → тело → линии → поверхности → PMI.

Инициализация парметров модели

Пара методов, которые вызывали инициализацию параметров импортера перед чтением SGF. Функциональность перенесена в `StartFile`.

Метод	Описание
`StartModel()`	Проинициализировать параметры импортера (после `StartFile`).
`CloseModel()`	Сбросить параметры (перед `CloseFile`).

Сборка тела

Старый способ построения тела без `ComboSolid` (Работа с деталью): вершины и рёбра тела объявлялись напрямую, а группа граней помечалась как замкнутая оболочка для сшивки силами CAM-системы. Заменён на `ComboSolid`.

Метод	Описание
`CreateSolidVertex(ID: string; P: TST3DPoint)`	Вершина тела (концевая точка ребра).
`CreateSolidEdge(ID, SourceCurveID, svID, tvID: string)`	Именованное ребро между вершинами `svID`⋯`tvID` на основе кривой `SourceCurveID`.
`SetGroupEntityIsSolid(Value: boolean)`	Флаг сшивки: помечает поверхности внутри текущей папки (`StartGroupEntity`) как замкнутую оболочку.

Свойства линии

Тип и толщина линии для последующих объектов. В SGF не используются.

Метод	Описание
`SetCurrentLineType(Value: TSTLineType)`	Тип линии (см. `TSTLineType`).
`SetCurrentLineWidth(Value: integer)`	Толщина линии.

Перечисление `TSTLineType`: `ltSolid`, `ltDash`, `ltDashDot`, `ltDot`.

Игнорирование выхода за диапазон параметров

Управлял тем, вычисляется ли поверхность за пределами её области определения (диапазона параметров U/V). Вызывать не нужно: при корректных исходных данных параметры точек не выходят за диапазон поверхности, поэтому флаг не требуется.

Метод	Описание
`SetIgnoreOutOfRange(Value: boolean)`	Одноразовый флаг: задаётся до создания поверхности, действует только на одну ближайшую и затем сбрасывается. `true` — выход за диапазон игнорируется (поверхность экстраполируется за границы); `false` — диапазон соблюдается строго.

PMI-аннотации

Старый механизм построения PMI-видов и аннотаций. Актуальный механизм PMI — в PMI (Product Manufacturing Information).

Метод	Описание
`CreateView(vT, vZ, vX: TST3DPoint; Scale: STFloat; id: string)`	Создать PMI-вид.
`AddViewEntity(ViewID, EntityPath: string)`	Привязать сущность к виду.
`BeginPmiEntity(id: string; vT, vZ, vX: TST3DPoint; PmiType: TSTPmiType)`	Начать PMI-аннотацию (см. `TSTPmiType`).
`PmiAddArrow(vT: TST3DPoint; Angle: STFloat; ArrowType: TSTPmiArrowType)`	Добавить стрелку (см. `TSTPmiArrowType`).
`BeginPmiText(vT: TST3DPoint; Angle, Height, Length: STFloat; NumberLines:`	Начать блок текста (см. `TSTPmiTextType`).

Метод	Описание
`integer; TextType: TSTPmiTextType)`	
`AddPmiText(Text: string)`	Добавить строку текста.
`EndPmiText()`	Завершить блок текста.
`PmiAddCurve(CurveID: string; CurveType: TSTPmiCurveType)`	Добавить кривую (см. `TSTPmiCurveType`).
`PmiAddMesh(ID: string)`	Добавить mesh в аннотацию.
`SetPmiPreferenceI(Path: string; Value: integer)`	Целочисленный параметр аннотации.
`SetPmiPreferenceD(Path: string; Value: STFloat)`	Параметр-double аннотации.
`SetPmiAssociativity(i: integer; ObjId: string)`	Привязать i-й объект-якорь к аннотации.
`EndPmiEntity()`	Завершить PMI-аннотацию.

Перечисления PMI:

Enum	Значения / назначение
`TSTPmiType`	Типы PMI-аннотаций (28 значений: размеры, выноски, заметки и т.д.).
`TSTPmiCurveType`	Типы PMI-кривых (15 значений).
`TSTPmiArrowType`	Типы стрелок PMI (22 значения).
`TSTPmiTextType`	Типы PMI-текста (14 значений).