CAM API

SDK и API СПРУТКАМ позволяют создавать расширения и внешние приложения, которые управляют CAM-системой через COM, работают с geometry и технологическими данными.

Основные разделы

- Поддерживаемые языки программирования — C#, Delphi и C++.
- Уроки — примеры создания расширений и внешних приложений.
- Системные расширения и entry points — подключение расширений к приложению.
- Отладка расширений — инструменты для C#, Delphi и C++.
- Machining Tools Import — работа с tool libraries, cutting tools и tool holders.
- Внешние приложения — подключение к CAM-системе из отдельного процесса.
- Использование ComWrapper — управление COM-объектами в .NET.
- API reference — типы и методы SDK.

Требования

Для разработки на C# требуется .NET 10.0 SDK или новее.

Поддерживаемые языки программирования

CAMAPI основан на технологии COM, которая позволяет работать с объектом через его интерфейс. Объекты CAMAPI реализуют открытые интерфейсы; их набор образует SDK API.

Поддерживаются следующие языки программирования:

- C#
- C++
- Delphi

Примеры предоставляются для каждого из этих языков.

Поддержка C# в CAMAPI

Factory class

После загрузки managed DLL CAM-система ищет в ней namespace `CAMAPI` с классом `ExtensionFactory`. Класс должен реализовывать интерфейс IExtensionFactory. Через объект этого интерфейса CAM-система взаимодействует с загруженной DLL.

Интерфейс предоставляет метод создания экземпляров расширений, который вызывает CAM-система.

```
// ReSharper disable once CheckNamespace
namespace CAMAPI;

using Extensions;
using ResultStatus;

/// <summary>
/// Factory for creating extensions. Namespace and class name always should
be CAMAPI.ExtensionFactory,
/// so CAMAPI will find it
/// </summary>
public class ExtensionFactory : IExtensionFactory
{
    /// <inheritdoc />
    public void OnLibraryRegistered(IExtensionFactoryContext context, out
TResultStatus ret)
    {
        ret = default;
    }

    /// <inheritdoc />
    public void OnLibraryUnRegistered(IExtensionFactoryContext context, out
TResultStatus ret)
    {
        ret = default;
    }

    /// <summary>
    /// Create new instance of our extension
    /// </summary>
    /// <param name="extensionIdent">
    /// Unique identifier, if out library has more than one extension. Should
accord with
    /// value in settings json, describing this library
```

```
    /// </param>
    /// <param name="ret">Error to return it, because throw exception will
not work</param>
    /// <returns>Instance of out extension</returns>
    public IExtension? Create(string extensionIdent, out TResultStatus ret)
    {
        try
        {
            // write your code to create new instance
        }
        catch (Exception e)
        {
            ret.Code = TResultStatusCode.rsError;
            ret.Description = e.Message;
        }
        return null;
    }
}
```

Подключение SDK

Добавьте ссылку на актуальную версию пакета SDK.

```
<PackageReference Include="SprutTechnology.CAMAPI.SDK.Net" Version="18.2.0" />
```

Поддержка Delphi в CAMAPI

Экспортируемая функция

Native DLL должна экспортировать функцию, которая служит entry point:

```
function CreateInstanceOfExtension(PluginID: WideString): NativeUInt;
begin
    // create instance of your class
    var res: IExtension;
    if SameText(PluginID, TExtensionUtilCommonDll.GetIdent()) then
        res := TExtensionUtilCommonDll.Create() as IExtension

    else
        raise Exception.Create('Unknown ID ' + PluginID);

    // modify as pointer
    res._AddRef;
    result := NativeUInt(res);
end;
```

Логика функции может быть сложнее, однако она должна возвращать указатель на объект с увеличенным reference count.

Подключение SDK

SDK поставляется в пакете `SprutTechnology.CAMAPI.SDK.bpl.x64`. В BuildSystem можно настроить загрузку и распаковку пакета в заданный каталог перед компиляцией. Затем этот каталог необходимо подключить в настройках проекта Delphi.

В файле `build.cs`:

- Добавьте в список `ManagerProps`:

```
new RestorerNugetProps
{
    Name = "RestorerNuget",
    DepsProp = new List<RestorerDepProp>
    {
        new()
        {
            PackageId = "SprutTechnology.CAMAPI.SDK.bpl.x64",
            Version = "18.2.0",
            OutDir = Path.Combine(RootDirectory.Parent, "SDK")
```

```
        }
    }
}
```

- Задайте соответствия:

```
settings.ManagerNames.Add("restorer", "Debug", "RestorerNuget");
settings.ManagerNames.Add("restorer", "Release", "RestorerNuget");
```

- Добавьте восстановление зависимостей перед компиляцией в целевую задачу:

```
private Target Compile => _ => _
    .Executes(() =>
    {
        _buildSpace.Projects.Restore(Variant);
        _buildSpace.Projects.Compile(Variant, true);
    });
```

Таким же способом можно подключить любой другой пакет — аналогично `PackageReference` в проектах C#.

Поддержка C++ в CAMAPI

Экспортируемая функция

Native DLL должна экспортировать функцию, которая служит entry point:

```
extern "C" __declspec(dllexport) void* __stdcall CreateInstanceOfExtension(const
wchar_t* PluginID) {
  // create instance of your class
  IExtensionPtr resultPtr = new ExtensionUtilityExample();

  // modify as pointer
  resultPtr.AddRef();
  void* res = resultPtr.GetInterfacePtr();
  return (res);
}
```

Логика функции может быть сложнее, однако она должна возвращать указатель на объект с увеличенным reference count.

Подключение SDK

SDK поставляется в пакете `SprutTechnology.CAMAPI.SDK.tlb`. В BuildSystem можно настроить загрузку и распаковку пакета в заданный каталог перед компиляцией. Затем этот каталог необходимо подключить в настройках проекта C++.

В файле `build.cs`:

- Добавьте в список `ManagerProps`:

```
new RestorerNugetProps
{
    Name = "RestorerNuget",
    DepsProp = new List<RestorerDepProp>
    {
        new()
        {
            PackageId = "SprutTechnology.CAMAPI.SDK.tlb",
            Version = "18.2.0",
            OutDir = Path.Combine(RootDirectory.Parent, "SDK")
        }
    }
}
```

- Задайте соответствия:

```
settings.ManagerNames.Add("restorer", "Debug", "RestorerNuget");
settings.ManagerNames.Add("restorer", "Release", "RestorerNuget");
```

- Добавьте восстановление зависимостей перед компиляцией в целевую задачу:

```
private Target Compile => _ => _
    .Executes(() =>
    {
        _buildSpace.Projects.Restore(Variant);
        _buildSpace.Projects.Compile(Variant, true);
    });
```

Таким же способом можно подключить любой другой пакет — аналогично `PackageReference` в проектах C#.

Уроки

Уроки помогают познакомиться с CAMAPI на практических примерах:

- Каждый урок знакомит с новой возможностью;
- Сложность постепенно возрастает, последующие уроки используют знания из предыдущих;
- Уроки можно проходить в любом порядке.

Список уроков:

- Урок 1 — введение в разработку расширений на примере C#;
- Урок 2 — подключение SDK к проекту Delphi через систему сборки;
- Урок 3 — подключение SDK к проекту C++ через систему сборки;
- Урок 4 — выгрузка расширения во время работы CAM-системы;
- Урок 5 — создание пользовательской операции;
- Урок 6 — работа с geometry из приложения C#;
- Урок 7 — управление CAM-системой из приложения C#;
- Урок 8 — интеграция с PLM-системой из расширения C#.

Дополнительная информация:

- Использование ComWrapper.

По мере развития документации список уроков пополняется.

Урок 1 — введение в разработку расширений

Цели

- Создать простое работающее расширение на C#;
- Рассмотреть общий порядок разработки; подробности будут разобраны в следующих уроках.

Практическая часть

Порядок действий:

- Откройте репозиторий GitVerse и клонируйте его.
- Найдите каталог `ExtensionEmptyNet` — он послужит основой нового расширения.
- Проверьте `ExtensionEmptyNet/.stbuild/build/stbuild.csproj`: в нём указана версия системы сборки. Для доступа к новым возможностям системы сборки может потребоваться её обновление.
- Откройте проект `ExtensionEmptyNet/project/main/ExtensionEmptyNet.csproj`. Его структура и особенности будут рассмотрены в следующих уроках. Сейчас задача — создать простейшее расширение.
- Проверьте версию пакета `SprutTechnology.CAMAPI.SDK.Net`: для доступа к новым возможностям API может потребоваться её обновление.
- При необходимости измените `ExtensionTest.cs`. Вы можете использовать API без ограничений; пример упрощён, чтобы показать общий процесс создания расширения.
- Для выполнения кода DLL должна содержать класс ExtensionFactory. CAM-система вызовет его метод `Create`, получит экземпляр класса расширения и выполнит его код.
- Измените JSON-файл:
  - Тип расширения должен совпадать с идентификатором entry point;
  - Идентификатор расширения должен соответствовать значению, которое ожидает метод `ExtensionFactory.Create`;

```
{
    "name": "Empty extension on C#",
    "description": "Empty extension on C#",
    "module_path": "${extensionJsonFolder}/ExtensionEmptyNet.dll",
    "extensions": [
        {
            "utility": {
            "name": "Empty extension on C#",
                "caption": "Empty extension on C#",
                "id": "Extension.Utility.Empty.Net"
            }
        }
```

```
    ]
}
```

- Соберите проект, чтобы проверить его работоспособность.
- Создайте файл установки расширения с помощью команды `ExtensionEmptyNet/commands/pack.cmd`;
- При успешной упаковке появится файл `ExtensionEmptyNet/project/main/bin/ExtensionEmptyNet.dext`. Его можно установить на этом или другом компьютере с соответствующей версией CAM-системы.
- Дважды щёлкните файл, чтобы установить расширение.
- Запустите CAM-систему, откройте меню утилит и выберите `Empty extension on C#`. Название пункта должно соответствовать значению `caption` в JSON.
- Убедитесь, что открылся Блокнот со сведениями о проекте.

Урок 2 — подключение SDK к проекту Delphi через систему сборки

Цели

- Создать простое работающее расширение на Delphi;
- Подключить SDK к проекту Delphi через систему сборки.

Практическая часть

Порядок действий:

- Откройте репозиторий GitVerse и клонируйте его.
- Найдите каталог `ExtensionEmptyDelphi` — он послужит основой нового расширения.
- Обратите внимание на следующие настройки:
  - `ExtensionEmptyDelphi/.stbuild/build/stbuild.csproj` — версия системы сборки. Для доступа к новым возможностям системы сборки может потребоваться её обновление.
  - `ExtensionEmptyDelphi/.stbuild/build/build.cs`:
    - Свойства `RestorerNugetProps`, где указана версия `SprutTechnology.CAMAPI.SDK.bpl.x64`;
    - Свойства `BuilderMsDelphiProps`, где указан путь к `MSBuild.exe`. Используется версия из Microsoft Framework, а не из SDK;
- При необходимости измените `src/Extension.pas`. Вы можете использовать API без ограничений; пример упрощён, чтобы показать общий процесс создания расширения. Проверьте следующую настройку:
  - В `unit scope names` обязательно должен присутствовать элемент `IDL`;
- Для выполнения кода проект должен экспортировать функцию CreateInstanceOfExtension. Она принимает значение `id` из `ExtensionEmptyDelphi.settings.json` и возвращает ссылку на созданный объект.
- Соберите проект, чтобы проверить его работоспособность.
- Создайте файл установки расширения с помощью команды `ExtensionEmptyDelphi/commands/pack.cmd`;
- При успешной упаковке появится файл `ExtensionEmptyDelphi/project/main/Win64/Release/ExtensionEmptyDelphi.dext`. Его можно установить на этом или другом компьютере с соответствующей версией CAM-системы.
- Дважды щёлкните файл, чтобы установить расширение.
- Запустите CAM-систему, откройте меню утилит и выберите `Empty extension on Delphi`. Название пункта должно соответствовать значению `caption` в JSON.
- Убедитесь, что открылся Блокнот со сведениями о проекте.

Урок 3 — подключение SDK к проекту C++ через систему сборки

Цели

- Создать простое работающее расширение на C++;
- Подключить SDK к проекту C++ через систему сборки.

Практическая часть

Порядок действий:

- Откройте репозиторий GitVerse и клонируйте его.
- Найдите каталог `ExtensionEmptyCpp` — он послужит основой нового расширения.
- Обратите внимание на следующие настройки:
  - `ExtensionEmptyCpp/.stbuild/build/stbuild.csproj` — версия системы сборки. Для доступа к новым возможностям системы сборки может потребоваться её обновление.
  - `ExtensionEmptyCpp/.stbuild/build/build.cs`:
    - Свойства `RestorerNugetProps`, где указана версия `SprutTechnology.CAMAPI.SDK.tlb`;
    - Свойства `BuilderMsCppProps`, где указан путь к `MSBuild.exe`. Используется версия из Microsoft Visual Studio;
  - `ExtensionEmptyCpp\project\main\ExtensionEmptyCpp.settings.json` — подпись пункта меню, подсказка и путь к значку относительно DLL;
- При необходимости измените `main.cpp`. Вы можете использовать API без ограничений; пример упрощён, чтобы показать общий процесс создания расширения.
- Для выполнения кода проект должен экспортировать функцию CreateInstanceOfExtension. Она принимает значение `id` из `ExtensionEmptyCpp.settings.json` и возвращает ссылку на созданный объект.
- Соберите проект, чтобы проверить его работоспособность.
- Создайте файл установки расширения с помощью команды `ExtensionEmptyCpp/commands/pack.cmd`;
- При успешной упаковке появится файл `ExtensionEmptyCpp/project/main/x64/Release/ExtensionEmptyCpp.dext`. Его можно установить на этом или другом компьютере с соответствующей версией CAM-системы.
- Дважды щёлкните файл, чтобы установить расширение.
- Запустите CAM-систему, откройте меню утилит и выберите `Empty extension on C++`. Название пункта должно соответствовать значению `caption` в JSON.
- Убедитесь, что открылся Блокнот со сведениями о проекте.

Урок 4 — выгрузка расширения во время работы CAM-системы

Цели

- Создать простое работающее расширение на C#;
- Рассмотреть способы выгрузки расширений.

Практическая часть

Порядок действий:

- Запустите CAM-систему: список утилит будет изменяться во время её работы.
- Откройте репозиторий GitVerse и клонируйте его.
- Найдите каталог `GCodeGeneration\ExtensionUtilityNCMakerNet` — он послужит основой расширения.
- Код расширения уже готов. В этом уроке изменяется только файл конфигурации.
- Откройте и соберите .NET-проект. Должен появиться файл `bin/debug/ExtensionUtilityNcMakerNet.dll`;
- Скопируйте `ExtensionUtilityNcMakerNet.settings.json` в `bin/debug/`;
- Установите расширение в первый раз:
  - Поместите DLL и `settings.json` в архив с расширением `.dext`;
  - Запустите созданный файл `.dext`, чтобы добавить расширение в CAM-систему. Параметр `unload_mode` не задан, поэтому используется значение по умолчанию `umJustAfterExecution`;
- В меню утилит CAM-системы выберите `Generate G code by C#`, затем закройте Блокнот с результатами.
- Не закрывая CAM-систему, установите расширение во второй раз:
  - Измените `bin/debug/ExtensionUtilityNcMakerNet.settings.json`:
    - Задайте режим выгрузки `umManuallyFromUI`;
    - Измените подпись пункта меню;

```
{
  "name": "Extension on C# to generate G code",
  "description": "Extension on C# to generate G code",
  "module_path": "${extensionJsonFolder}/ExtensionUtilityNcMakerNet.dll",
  "unload_mode": "umManuallyFromUI",
  "extensions": [
    {
      "utility": {
        "name": "Generate G code by C#",
        "caption": "Generate G code by C# (version 2)",
        "id": "Extension.Utility.NcMaker.Net"
```

```
      }
    }
  ]
}
```

- Поместите JSON-файл и DLL в архив с расширением `.dext`;
- Запустите файл `.dext`, чтобы установить расширение. DLL будет перезаписана, поскольку она была выгружена. Если библиотека остаётся загруженной, возникнет ошибка;
- Откройте меню утилит CAM-системы. В нём должен появиться пункт `Generate G code by C# (version 2)`. Новая подпись означает, что расширение обновлено;
- Выберите `Generate G code by C# (version 2)` и закройте Блокнот с результатами;
- Повторно запустите созданный ранее файл `.dext`. Появится ошибка `Access to the path 'ExtensionUtilityNcMakerNet.dll' is denied`: расширение осталось в памяти, как и предусмотрено режимом `umManuallyFromUI`;
- Выгрузите расширение вручную через интерфейс:
  - Откройте `Settings` → `Extensions`;
  - Выберите `Generate G code by C#` — значение поля `name` из `settings.json`;
  - Нажмите `unload`. После успешной выгрузки кнопка станет недоступной; при неудаче появится ошибка;
- Снова запустите файл `.dext`. Теперь установка пройдёт успешно, поскольку расширение выгружено вручную;

- Не закрывая CAM-систему, установите расширение в третий раз:
  - Измените `bin/debug/ExtensionUtilityNcMakerNet.settings.json`:
    - Задайте режим выгрузки `umBeforeProgramFinalization`;
    - Измените подпись пункта меню;

```
{
  "name": "Extension on C# to generate G code",
  "description": "Extension on C# to generate G code",
  "module_path": "${extensionJsonFolder}/ExtensionUtilityNcMakerNet.dll",
  "unload_mode": "umBeforeProgramFinalization",
  "extensions": [
    {
      "utility": {
        "name": "Generate G code by C#",
        "caption": "Generate G code by C# (version 3)",
        "id": "Extension.Utility.NcMaker.Net"
      }
    }
  ]
}
```

    - Поместите JSON-файл и DLL в архив с расширением `.dext`;
    - Запустите созданный файл `.dext`, чтобы установить расширение;
    - Откройте меню утилит CAM-системы. В нём должен появиться пункт `Generate G code by C# (version 3)`, подтверждающий обновление подписи;
    - Выполните команду расширения и закройте Блокнот с результатами;
    - Повторно запустите созданный ранее файл `.dext`. Появится ошибка `Access to the path 'ExtensionUtilityNcMakerNet.dll' is denied`;
    - Проверьте возможность ручной выгрузки через интерфейс:
        - Откройте `Settings` → `Extensions`;
        - Выберите `Generate G code by C#` — значение поля `name` из `settings.json`;
        - Убедитесь, что кнопка `unload` недоступна. Режим `umBeforeProgramFinalization` запрещает ручную выгрузку;
- Закройте CAM-систему и запустите файл `.dext`. Библиотека выгружается из памяти при завершении CAM-системы, поэтому теперь её можно обновить.

Урок 5 — создание пользовательской операции

Цели

- Создать новый тип операции с настройками.

Практическая часть

Порядок действий:

- Откройте репозиторий GitVerse и клонируйте его.
- Найдите каталог `Operation\ExtensionOperationSimpleNet` — он послужит основой расширения.
- Обратите внимание на следующие настройки:
  - XML-файл `project\main\OperationSimpleNet.xml` содержит настройки операции:
    - Признак того, что расчёт toolpath выполняет внешнее расширение;
    - Параметры операции, доступные пользователю в интерфейсе;
    - Значение `ContainerID` изменять нельзя;
    - Значение `SolverID` впоследствии передаётся в `ExtensionFactory.Create`;
  - `ExtensionFactory.cs`:
    - Методы `OnLibraryRegistered` и `OnLibraryUnRegistered` изменяют глобальный файл `UserOperationsList.xml`, содержащий пользовательские операции;
  - Toolpath рассчитывается во вспомогательном потоке, а .NET позволяет обращаться к COM-объектам только из потока, в котором они созданы. Поэтому избегайте инициализации COM-свойств `ExtensionOperationSimpleNet` в `RunInitialize`, если затем они используются в другом потоке.
- При необходимости измените `main.cpp`. Вы можете использовать API без ограничений.
- Выполните задачу `Pack`, создающую файл `.dext`, и запустите этот файл. При первой установке это обязательно: регистрация расширения изменяет глобальный `UserOperationsList.xml`, добавляя путь к `bin\Release\OperationSimpleNet.xml`.
- Перезапустите CAM-систему, чтобы обновить список типов операций;
- Добавьте новую операцию из группы, указанной в узле `MultiGroup` XML-файла. Если операция отсутствует, включите отображение недоступных операций (`unavailable operations`);
- Выполните для новой операции команду `generate toolpath`.

Урок 6 — работа с geometry из приложения C#

Цели

- Создать отдельное приложение на C#, работающее с geometry model.

Практическая часть

- Откройте репозиторий GitVerse и клонируйте его.
- Найдите каталог `Geometry`. В нём два примера: работа с импортом и работа с geometry model.
- Откройте проект `ExtensionUtilityGeometryModelNet\project\main\ExtensionUtilityGeometryModelNet.csproj`.
- Соберите проект.
- Отдельно запустите CAM-систему.
- Откройте `Setting` и перейдите на вкладку `Extensions`.
- Нажмите `Install` и выберите файл `ExtensionUtilityGeometryModelNet\project\main\ExtensionUtilityGeometryModelNet.settings.json`.
- Вернитесь в главное окно, нажмите `Import` и выберите модель `Models\Milling_3D\49-1.igs` из примеров.
- После импорта модели откройте меню утилит в левом верхнем углу и запустите одну из утилит:
  - `Export geometry node` — экспортирует текущую geometry model в файл `.osd`;
  - `Transform geometry node` — преобразует текущую geometry model;
  - `Select colored geometry nodes` — изменяет цвет узлов и выделяет их в главном окне.

Урок 7 — управление CAM-системой из приложения C#

Цели

- Создать отдельное приложение на C# для взаимодействия с CAM-системой.

Практическая часть

- Откройте репозиторий GitVerse и клонируйте его.
- Найдите каталог `ApplicationEmptyNet` — он послужит основой нового приложения.
- Откройте проект `ApplicationEmptyNet/project/main/ApplicationEmptyNet.csproj`.
- Обратите внимание на следующие настройки:
  - Singleton `CamHelper`, который подключается к найденному экземпляру CAM-системы;
- При необходимости измените `MainWindow.xaml.cs`. Вы можете использовать API без ограничений; пример упрощён, чтобы показать общий процесс создания отдельного приложения.
- Запустите пример;
- Отдельно запустите CAM-систему;
- Нажмите `Open project` и выберите любой проект из примеров;
- Убедитесь, что выбранный проект открылся в CAM-системе.

Урок 8 — интеграция с PLM-системой из расширения C#

Цели

- Изучить базовую структуру расширения для интеграции с PLM-системой;
- Создать расширение на C# или изменить готовый пример для работы с PLM-системой.

Практическая часть

Чтобы создать расширение:

- Установите актуальную версию NuGet-пакета с базовыми интерфейсами:
  - `SprutTechnology.CAMAPI.SDK.Net`.
- Реализуйте два интерфейса:
  - `IExtension` — общий интерфейс, через который CAM-система загружает расширение. Его единственное свойство `IExtensionInfo Info` содержит основные metadata;
  - `IExtensionPLM` — интерфейс интеграции с PLM-системой: подключение, отправка и получение данных.
- Начните с обязательных методов:
  - `GetParameters` — возвращает объект `IPLMParameters`, с помощью которого CAM-система отображает настройки расширения;
  - `Connect` — получает значения параметров и устанавливает соединение с PLM-системой;
  - `Disconnect` — закрывает соединение и при необходимости очищает внутреннее состояние.
- Реализуйте основные методы работы с элементами:
  - Получение элементов:
    - `GetChilds` — получает дочерние элементы по идентификатору родителя. Если родитель не указан, возвращает элементы корневого уровня;
    - `FindItems` — ищет элементы по имени;
    - `GetItem` — загружает элемент по идентификатору;
    - `GetLinkedItem` — возвращает связанный элемент;
    - `GetItemData` — загружает подробные данные выбранного элемента.
  - Передача элементов:
    - `DownloadItems` — скачивает выбранные элементы, кроме проектов;
    - `UploadItem` — загружает обычные элементы в PLM-систему, кроме проектов;
    - `DownloadProject` / `UploadProject` — выполняют соответствующие действия с проектами.
- Реализуйте методы жизненного цикла:

    - `Install` и `Uninstall` вызываются при регистрации и удалении расширения. Их можно оставить пустыми, если дополнительные действия установки и очистки не нужны.
- Логические свойства `SupportProjectLoad`, `SupportMachineLoad`, `SupportPostprocessorLoad`, `SupportPostprocessorInsideMachineLoad`, `SupportToolLoad`, `SupportToolSave`, `SupportDomainAuth` сообщают CAM-системе, какие функции поддерживает расширение. Если свойство возвращает `false`, соответствующая возможность отключается в интерфейсе.

Полный перечень методов и их описания приведены в API reference.

Можно клонировать готовый пример, использовать его без изменений или доработать для собственной PLM-системы:

- Откройте репозиторий примеров GitVerse и клонируйте его.
- Найдите каталог с примером интеграции с PLM-системой — он послужит основой расширения.
- Откройте проект `PLMExtension/PLMExtensionNet/project/main/PLMExtensionNet.csproj` в выбранной среде разработки или редакторе кода.
- Обратите внимание на следующие настройки:
    - `PLMExtension/PLMExtensionNet/.stbuild/build/stbuild.csproj` — версия системы сборки. Для доступа к новым возможностям системы сборки может потребоваться её обновление;
    - Версия NuGet-пакета `SprutTechnology.CAMAPI.SDK.Net`.
- Адаптируйте пример к своей PLM-системе. Он содержит базовую рабочую реализацию обоих интерфейсов и имитирует работу PLM-системы с помощью каталогов.

Когда расширение готово:

- Соберите решение;
- Отдельно запустите CAM-систему;
- Установите расширение в CAM-систему и настройте подключение к PLM по инструкции;
- Проверьте работу всех функций расширения в CAM-системе.

Системные расширения CAMAPI

В CAMAPI входит базовый набор системных расширений, обеспечивающих его работу. Они подключаются так же, как сторонние расширения, и при необходимости могут быть заменены.

Доступные системные расширения:

- Extension.Util.Common.Dll.

Системное расширение Extension.Util.Common.Dll

Расширение загружает DLL-файлы: native DLL на C++ и Delphi, а также managed DLL на C#.

- Entry point — utility_runner;
- Реализуемый интерфейс — IExtensionUtilityRunner.

Расширение определяет, относится ли DLL к managed DLL. Для native DLL вызывается функция `CreateInstanceOfExtension` с GUID из настроек утилиты в конфигураторе. Функция должна вернуть объект интерфейса `IExtensionUtility`.

В managed DLL расширение ищет namespace `CAMAPI` с классом `ExtensionFactory`, реализующим интерфейс `IExtensionFactory`, и создаёт экземпляр этого класса. Затем вызывается метод `IExtensionFactory.Create` с GUID нужной утилиты, поскольку одна DLL может содержать несколько утилит.

Системное расширение Extension.Util.Common.Exe

Расширение запускает EXE-файлы в отдельном процессе, чтобы не прерывать работу CAM-системы. Если в настройках утилиты указаны параметры, они передаются запускаемой программе.

- Entry point — utility_runner;
- Реализуемый интерфейс — IExtensionUtilityRunner.

Отладка расширений

В этом разделе описана отладка собственных расширений в установленной CAM-системе:

- Отладка .NET-расширения в Visual Studio Code;
- Отладка расширения Delphi в RAD Studio;
- Отладка расширения C++ в Visual Studio.

Отладка .NET-расширения в Visual Studio Code

Отладка в Visual Studio Code настраивается в файле `launch.json`: в нём задаются исполняемая программа и параметры запуска. Готовый `launch.json` есть в примерах репозитория GitVerse:

- `program`: `"C:/Program Files/Sprut Technology/SprutCAM 18/Bin64/sc.exe"` — замените каталог на путь установленной версии;
- `args`: `["/EXTENSION_DEBUG_FOLDER=\"${workspaceFolder}/project/main/bin/Debug\""]`.

При выборе `Run` → `Start Debugging`:

- Visual Studio Code выполнит задачу `preLaunchTask`:
  - Соберёт проект и поместит DLL по пути из файла `.csproj` (`<OutDir>bin\$(Configuration)\</OutDir>`);
  - Скопирует `settings.json` в ту же папку;
- CAM-система запустится с параметром `/EXTENSION_DEBUG_FOLDER="${workspaceFolder}/project/main/bin/Debug"`;
- Помимо зарегистрированных расширений, CAM-система проверит каталог `${workspaceFolder}/project/main/bin/Debug` и загрузит собранную DLL согласно файлу `settings.json`;
- При выполнении кода расширения в CAM-системе отладчик остановится на breakpoints, установленных в Visual Studio Code.

Отладка расширения Delphi в RAD Studio

Чтобы отладить расширение в RAD Studio:

- В свойствах проекта откройте `Options` → `Debugger`:
  - Укажите в `Host application` путь к исполняемому файлу CAM-системы, например `C:/Program Files/Sprut Technology/SprutCAM 18/Bin64/sc.exe`; замените каталог на путь установленной версии;
  - Укажите в `Parameters` значение `/EXTENSION_DEBUG_FOLDER="${workspaceFolder}/project/main/bin/Debug"`. Вместо `${workspaceFolder}` подставьте полный путь к каталогу примера.
- Скопируйте файл `settings.json` в каталог `bin\debug`.

При выборе `Run (F9)`:

- RAD Studio соберёт проект и поместит DLL по пути, указанному в настройках проекта;
- CAM-система запустится с параметром `/EXTENSION_DEBUG_FOLDER="${workspaceFolder}/project/main/bin/Debug"`;
- Помимо зарегистрированных расширений, CAM-система проверит каталог `${workspaceFolder}/project/main/bin/Debug` и загрузит собранную DLL согласно файлу `settings.json`;
- При выполнении кода расширения в CAM-системе отладчик остановится на breakpoints, установленных в RAD Studio.

Отладка расширения C++ в Visual Studio

Чтобы отладить расширение в Visual Studio:

- В свойствах проекта откройте `Options` → `Debugger`:
  - Укажите в `Command` путь к исполняемому файлу CAM-системы, например `C:/Program Files/Sprut Technology/SprutCAM 18/Bin64/sc.exe`; замените каталог на путь установленной версии;
  - Укажите в `Parameters` значение `/EXTENSION_DEBUG_FOLDER="${workspaceFolder}/project/main/bin/Debug"`. Вместо `${workspaceFolder}` подставьте полный путь к каталогу примера.
- Скопируйте файл `settings.json` в каталог `bin\debug`.

При выборе `Start debugging (F5)`:

- Visual Studio соберёт проект и поместит DLL по пути, указанному в настройках проекта;
- CAM-система запустится с параметром `/EXTENSION_DEBUG_FOLDER="${workspaceFolder}/project/main/bin/Debug"`;
- Помимо зарегистрированных расширений, CAM-система проверит каталог `${workspaceFolder}/project/main/bin/Debug` и загрузит собранную DLL согласно файлу `settings.json`;
- При выполнении кода расширения в CAM-системе отладчик остановится на breakpoints, установленных в Visual Studio.

Entry points CAMAPI

CAMAPI предоставляет entry points для подключения расширений сторонних разработчиков. Каждый entry point определяет место интеграции с CAM-системой и позволяет добавлять новые возможности или расширять существующие. Это даёт возможность создавать приложения и инструменты для конкретных задач пользователей.

Доступные entry points:

- Утилита в главном окне;
- Исполнитель утилиты в главном окне;
- Пункт контекстного меню операции.

Настройки расширения

Расширение, подключаемое к CAM-системе, состоит из DLL и JSON-файла с описанием библиотеки.

Описание библиотеки, содержащей несколько расширений

Список полей:

- name;
- description;
- module_path. Эти поля представлены в интерфейсе IExtensionLibraryInfo.

Описание отдельного расширения

Список расширений задаётся массивом `JsonArray`. Каждое расширение описывается отдельным `JsonObject` с единственным ключом — уникальным идентификатором entry point. Значение ключа — объект `JsonObject` со следующими полями:

- id;
- name.

Дополнительно могут задаваться параметры конфигурации, необходимые для выбранного entry point.

Идентификатор должен быть уникальным среди всех расширений. Чтобы избежать совпадений, рекомендуется добавлять к имени собственный префикс.

Пример

```
{
    "name": "Library for empty extension",
    "description": "Library for empty extension",
    "module_path": "${extensionJsonFolder}/ExtensionEmpty.dll",
    "extensions": [
        {
            "utility": {
                "id": "Extension.Utility.Empty",
                "name": "Empty extension"
            }
        }
    ]
}
```

Добавление утилиты

Для работы с утилитами CAMAPI предоставляет entry points двух типов. Они позволяют добавить любой файл в список утилит и настроить его выполнение: DLL, исполняемый EXE-файл или скрипт, команды которого обрабатывает другое приложение.

Первый entry point расширяет список утилит. Аналогичный результат можно получить вручную через окно конфигурации, однако расширения можно переносить между компьютерами в файле `.dext`. Поэтому пользователю можно передать готовое расширение вместо инструкции по настройке новой утилиты. Это особенно удобно при большом количестве утилит.

Этот entry point можно использовать вместе со вторым entry point, чтобы подключать утилиты любого формата.

- Идентификатор: `utility`;
- Используемые интерфейсы:
  - IExtensionTypeInfoUtility.
  - IExtensionUtility.
- Доступные параметры конфигурации:
  - `caption` — текст, который увидит пользователь;
  - `hint_text` — текст всплывающей подсказки при наведении указателя мыши;
  - `loader_extension_ident` — идентификатор расширения, которое запускает эту утилиту. Параметр необязателен: если он не задан, используется стандартное расширение Extension.Util.Common.Dll.

Примеры в репозитории GitVerse:

- C# — каталог `ExtensionUtilityNet`;
- Delphi — каталог `ExtensionUtilityDelphi`;
- C++ — каталог `ExtensionUtilityCpp`.

Важно: примеры демонстрируют, как CAM-система отображает ошибку в коде расширения. Для сохранения файла `exported.stcp` в этих примерах CAM-систему необходимо запустить с правами администратора.

Выполнение утилиты

Для этого entry point уже предусмотрены системные расширения:

- Загрузка DLL-файлов;
- Запуск EXE-файлов.

Этот entry point расширяет список типов утилит, которые можно подключить к CAM-системе. Системные расширения поддерживают только DLL и EXE. Создав расширение типа `utility_runner`, можно указать его в параметрах запуска расширения типа `utility`. CAM-система вызовет `utility_runner` и передаст ему сведения о запускаемой утилите.

- Идентификатор: `utility_runner`;
- Используемые интерфейсы:
  - IExtensionTypeInfoUtilityRunner.
  - IExtensionUtilityRunner.
- Доступные параметры конфигурации:
  - `filter_types` — фильтр файлов в диалоге выбора утилиты;
  - `visible_in_editor` — доступность расширения для выбора в редакторе утилит. Параметр позволяет скрыть специализированный исполнитель, предназначенный только для определённой утилиты.

Примеры в репозитории GitVerse:

- C# — каталог `ExtensionUtilityRunnerNet`;
- Delphi — каталог `ExtensionUtilityRunnerDelphi`;
- C++ — каталог `ExtensionUtilityRunnerCpp`.

Добавление пункта в контекстное меню операции

Этот entry point позволяет добавлять собственные пункты в контекстное меню выбранной операции. Для каждого пункта задаётся отдельная процедура выполнения. Одно расширение может добавлять произвольное количество таких функций.

- Идентификатор: `operation_popup`;
- Используемые интерфейсы:
  - IExtensionTypeInfoOperationPopup.
  - IExtensionOperationPopup.
- Параметры конфигурации отсутствуют.

Примеры в репозитории GitVerse:

- C# — каталог `ExtensionOperationPopupNet`;
- Delphi — каталог `ExtensionOperationPopupDelphi`;
- C++ — каталог `ExtensionOperationPopupCpp`.

Machining Tools Import API

В разделе описаны методы CAMAPI для Machining Tools Import в tool libraries CAM-системы. API позволяет создавать turning tools и milling tools, custom axial shape tools и cutting tool libraries. Перед использованием API убедитесь, что доступны следующие DLL:

- Bin64\MachiningToolsImport.dll;
- Bin64\MachinigToolsImportTypes.dll;
- Bin64\STMachiningToolsImportHelper.dll.

Чтобы понять принципы создания tools и tool libraries, изучите раздел о cutting tools в руководстве пользователя.

Системные требования

- Windows 10 или новее;
- .NET 10.0 SDK или новее.

Полезные ссылки

- Примеры Machining Tools Import в GitVerse

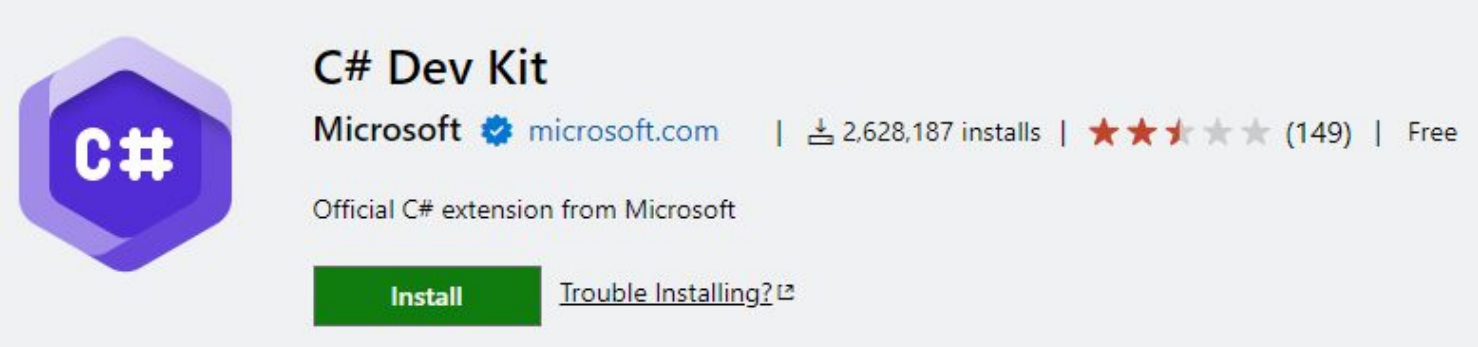
Далее

- Подготовка окружения для Machining Tools Import

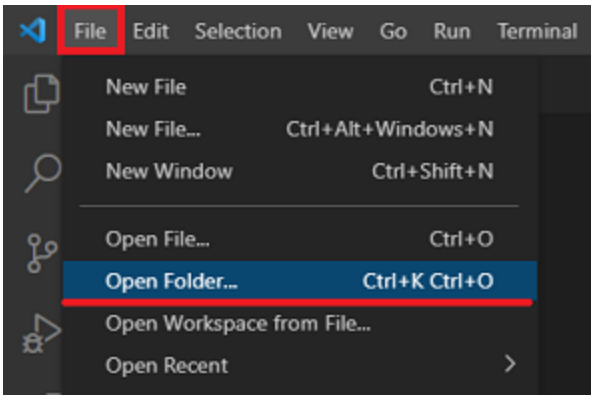
Подготовка окружения

В примере показана подготовка окружения для работы с API Machining Tools Import на основе консольного приложения C# в Visual Studio Code.

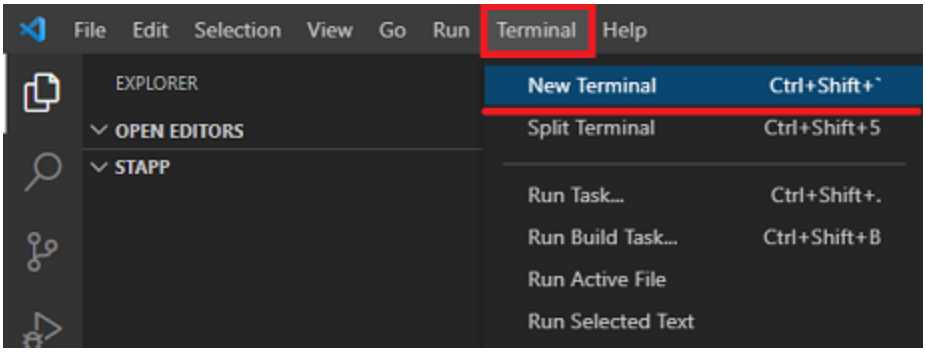
1. Скачайте и установите актуальную версию .NET SDK x64 с официального сайта.
2. Для работы с исходным кодом скачайте и установите актуальную версию Visual Studio Code.
3. Запустите Visual Studio Code и установите расширение C# Dev Kit for Visual Studio Code.



4. Если CAM-система ещё не установлена, скачайте и установите СПРУТКАМ.
5. Откройте в Visual Studio Code существующий каталог проекта или создайте новый. На вопрос `Do you trust the authors of the files in this folder?` ответьте `Yes, I trust the authors`.

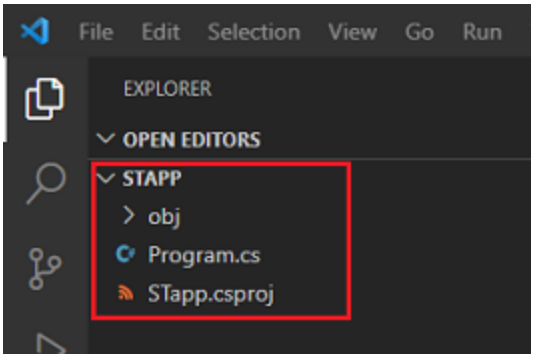


6. Откройте терминал Visual Studio Code.



7. Введите команду dotnet new console --framework net10.0 и нажмите Enter. Если используется другая версия SDK, укажите соответствующую версию вместо net10.0.



- После выполнения команды в рабочем каталоге появятся исходные файлы проекта:
- 

- Имя файла `.csproj` совпадает с именем рабочего каталога.

8. Откройте файл `.csproj` и замените его содержимое приведённым ниже:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net10.0-windows</TargetFramework>
    <OutputType>Exe</OutputType>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="SprutTechnology.CAMAPI.SDK.Net" Version="20.*" />
  </ItemGroup>
</Project>
```

- Укажите пути к DLL для установленной CAM-системы. Если библиотеки отсутствуют, проверьте, установлена ли CAM-система.
- Целевая платформа проекта должна иметь вид `net10.0-windows` или соответствовать более новой версии SDK с суффиксом `-windows`.

9. Откройте файл `Program.cs` и замените его содержимое приведённым ниже:

```
using System;
using System.IO;
using MachinigToolsImportTypes;
using STMachiningToolsImportHelper;
using STTypes;
```

```
namespace STConsoleApp
{
  class Program
  {
    private static string assemblyPath =
      @"C:\Program Files\[CAM_Path]\" + MTIMachiningToolsImportHelper.DllName;

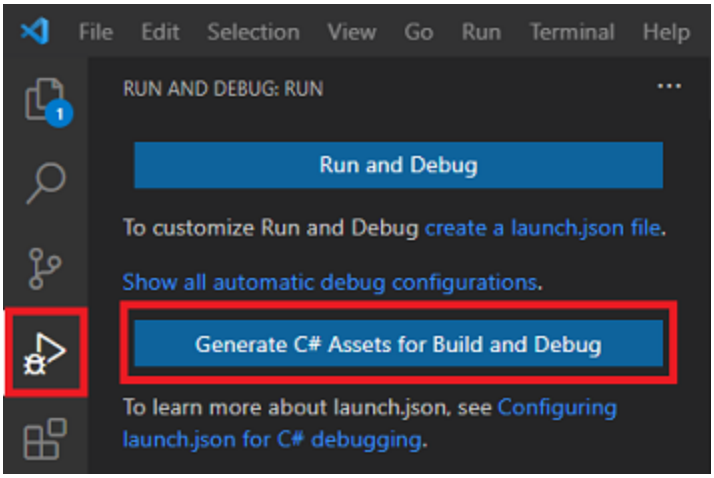
    static void Main(string[] args)
    {
      if (!File.Exists(assemblyPath)) {
        Console.WriteLine("The {0} source file is not found", assemblyPath);
        return;
      }

      try {
        var importer = MTIMachiningToolsImportHelper.CreateImporter(assemblyPath);
        //Use "importer" to create new tool items, to open an existing or create new
machining tools libraries

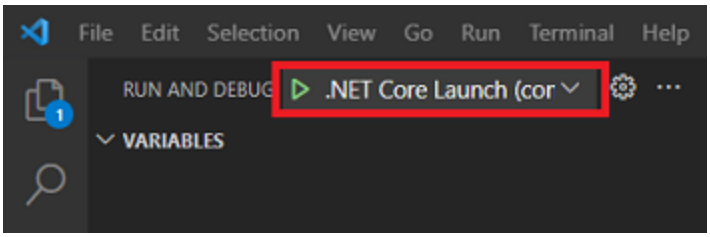
      }
      catch (Exception ex) {
        Console.WriteLine(ex.Message);
      }
      finally {
        MTIMachiningToolsImportHelper.FinalizeImporter();
      }
    }
  }
}
```

Задайте в переменной `assemblyPath` путь к каталогу `Bin64` установленной CAM-системы. Теперь объект `importer` можно использовать для создания tools, открытия и создания tool libraries и других действий. Перед завершением основной программы обязательно вызовите функцию `FinalizeImporter`, как показано в примере.

10. Откройте панель `Run and Debug` и нажмите `Generate C# Assets for Build and Debug`.



- Теперь можно запустить отладку приложения:
-



Если все пути указаны верно, пример должен выполниться без ошибок.

Окружение для разработки настроено.

Полезные ссылки

- Примеры Machining Tools Import в GitVerse

Далее

- Работа с tool library
- Работа с cutting tools

Работа с tool library

Tool libraries хранятся в файлах `.db`. Каждая tool library — независимое хранилище набора tools. Стандартные tool libraries размещаются в каталоге установленной версии продукта: например, в исходном примере это `C:\ProgramData\[CAM_Path]\Version 17\Libraries\Tools`. Пользовательские tool libraries можно создавать в любом каталоге. Для создания новой tool library или открытия существующей tool library используйте методы объекта `IMTI_MachiningToolsImportLibrary` (`importer`):

```
//C#
//create a new tool storage
var tNewStorage = importer.CreateNewToolsStorage(@"D:\teststorage.db");
//open an existing tool storage
var tExstStorage = importer.OpenExistingToolsStorage(@"D:\teststorage.db");
```

Чтобы добавить tool в библиотеку, вызовите следующую функцию:

```
//C#
//the tool must be created and formed before adding it to the tool library
bool result = storage.AddToolItem(tool);
```

Полезные ссылки

- Управление tool libraries в интерфейсе CAM-системы
- Подготовка окружения для Machining Tools Import
- Примеры Machining Tools Import в GitVerse

Далее

- Работа с cutting tools

Работа с cutting tools

Доступны три типа cutting tool: milling tools, turning tools и custom axial shape tools. Для их создания используйте методы объекта `IMTI_MachiningTools`:

- Методы создания milling tools:
  - CreateCylindricalMill
  - CreateSpphericalMill
  - CreateTorusMill
  - CreateDoubleRadialMill
  - CreateLimitedDoubleRadialMill
  - CreateConicalMill
  - CreateLimitedConicalMill
  - CreateEngraver
  - CreateDrill
  - CreateTwoStageDrill
  - CreateSpotDrill
  - CreateCenterDrill
  - CreateReamer
  - CreateCounterSinkDrill
  - CreateCounterBoreDrill
  - CreateBoringBar
  - CreateTap
  - CreateNegativeRadiusMill
  - CreateKnife
  - CreateSawBlade
  - CreateJetCutter
  - CreateFlatTipCountersink
  - CreatePilotCounterbore
  - CreateBackBoringBar
  - CreateThreadMill
  - CreateUndercutDoveMill
  - CreateUndercutSlotMill
  - CreateUndercutLollipopMill
  - CreateUndercutTwoAngleMill
  - CreateUndercutRoundGrooveMill
  - CreateUndercutSharpChamferMill
  - CreateUndercutRoundedChamferMill
  - CreateUndercutIndexableChamferMill
  - CreateUndercutBarrelMill
  - CreateUndercutLensBarrelMill

    - CreateUndercutTaperBarrelMill
    - CreateTaperJetCutter
    - CreatePlazmaJetCutter
    - CreateLaserJetCutter
    - CreateEmptyGripper
    - CreateSingleVacuumGripper
    - CreateFullConeSprayGun
    - CreateHollowConeSprayGun
    - CreateElipticFlatFanSprayGun
    - CreateElipticLinearFanSprayGun
    - CreateRectangularFlatFanSprayGun
    - CreateSphericalProbe
- Методы создания turning tools:
    - CreateTurnToolWithExternalHolder
    - CreateTurnToolWithBoringHolder
    - CreateTurnToolWithExternalThreadingHolder
    - CreateTurnToolWithInternalThreadingHolder
    - CreateTurnToolWithExternalGroovingHolder
    - CreateTurnToolWithInternalGroovingHolder
    - CreateTurnToolWithFaceGroovingHolder
    - CreateTurnToolWithProbingHolder
- Метод создания custom axial shape tool:
    - CreateCustomAxialShapeTool.

Полезные ссылки

- Работа с cutting tools в интерфейсе CAM-системы
- Подготовка окружения для Machining Tools Import
- Примеры Machining Tools Import в GitVerse

Далее

- Работа с milling tools
- Работа с turning tools
- Работа с custom axial shape tools
- Работа с tool holder

Работа с milling tools

Milling tools делятся на подтипы. Геометрия каждого подтипа задаётся собственным набором параметров. Общие атрибуты, например tool name и tool number, одинаковы для всех подтипов cutting tools. Ниже приведён пример создания Cylindrical Mill.

```
//C#
//importer is an instance of IMTI_MachiningToolsImportLibrary
var tool = importer.CreateCylindricalMill();
tool.OverallLength = 100;
tool.CuttingDiameter = 10;
tool.WorkingLength = 30;
tool.ShankDiameter = 12;
tool.ShoulderLength = 60;
tool.ShankTaperAngle = 40;

tool.SetName("My Cylindrical mill");
tool.SetIdentifier("m001");
tool.SetTeethsCount(3);
tool.SetMagazineNumber(1);
tool.SetToolNumber(1);
tool.SetUnits(TMTI_LinearUnits.luMillimeter);
tool.SetOverhang(100);
tool.SetDurability(70);
tool.SetMaxPlungeAngle(90);

var toolings = tool.GetToolingPoints();
toolings.ToolContactPointType = TMTI_AxialToolContactPointType.cpCustomPoint;
toolings.ToolContactPointsCount = 1;

toolings.ToolingPointCount = 1;
toolings.ToolingPointType[0] = TMTI_AxialToolToolingPointType.tpCustomPoint;
toolings.ToolingPointShift[0] = 1.5;
toolings.ToolingPointLengthCorrectorNumber[0] = 1;
toolings.ToolingPointRadiusCorrectorNumber[0] = 1;

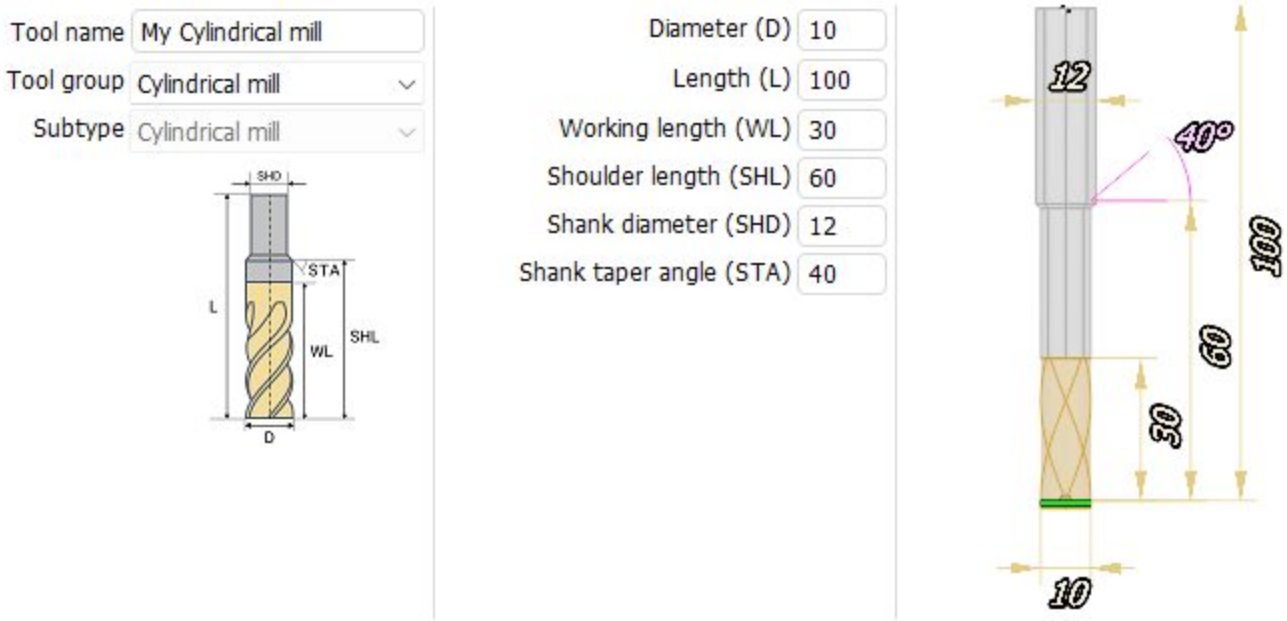
var cond = tool.GetCuttingConditions();
cond.CuttingSpeedMode = TMTI_CuttingSpeedMode.csmRPM;
cond.RotationsPerMinute = 100;
cond.SpindleGearRange = 1;
cond.RotationDirection = TMTI_RotationDirection.rdCW;
cond.FeedUnits = TMTI_FeedUnits.perMinute;
cond.FeedValue = 100;

int tubesCount = cond.Coolant.TubeCount; //20
```

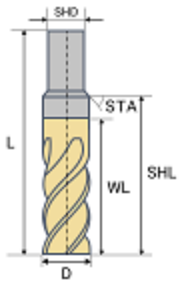
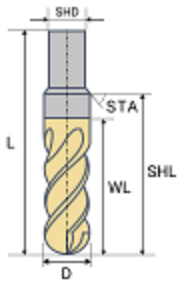
```
cond.Coolant.TubeIsOn[0] = true;
cond.Coolant.TubeIsOn[1] = false;
cond.Coolant.TubeIsOn[2] = true;

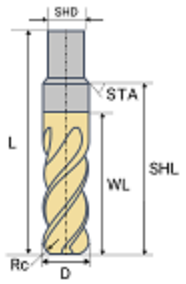
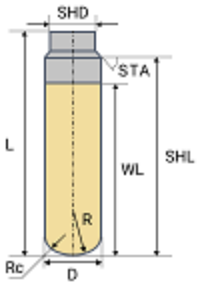
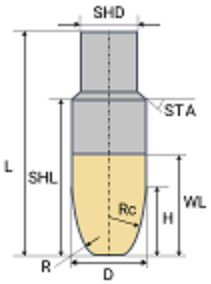
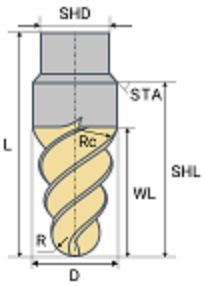
//after setting the properties, add this tool to your tool library instance
```

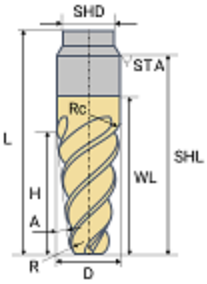
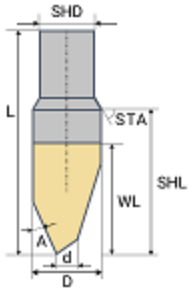
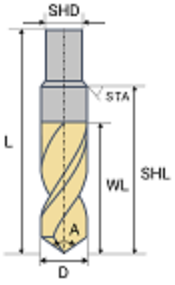
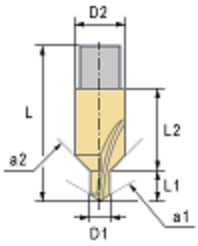
Результат выполнения кода:

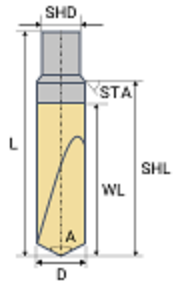
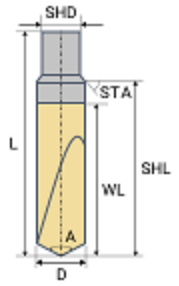
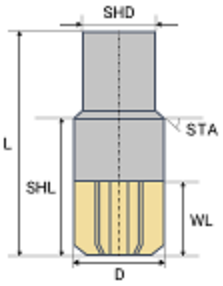
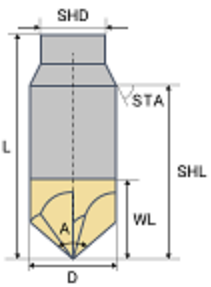


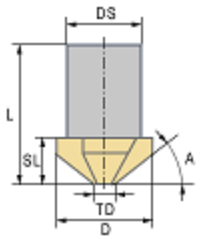
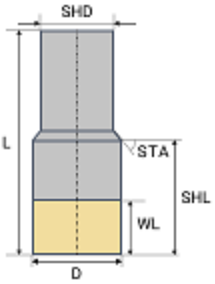
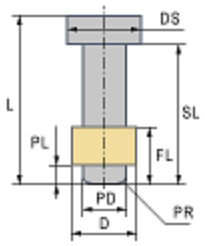
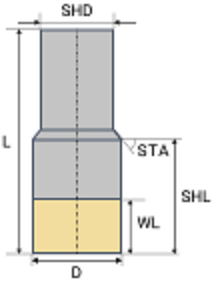
Геометрические свойства milling tools

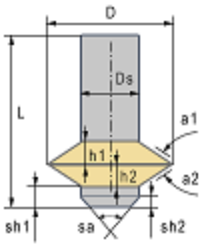
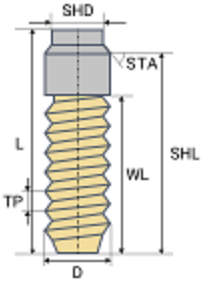
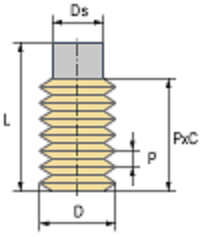
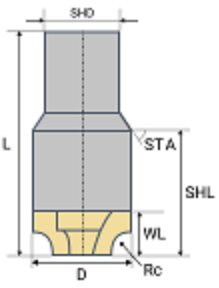
Тип tool и свойства API	Вид
Cylindrical Mill OverallLength (L) CuttingDiameter (D) WorkingLength (WL) ShankDiameter (SHD) ShoulderLength (SHL) ShankTaperAngle (STA)	
Spherical Mill OverallLength (L) CuttingDiameter (D) WorkingLength (WL) ShankDiameter (SHD) ShoulderLength (SHL) ShankTaperAngle (STA)	

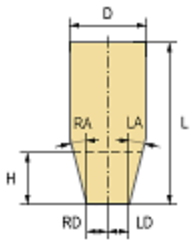
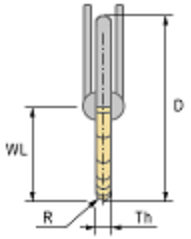
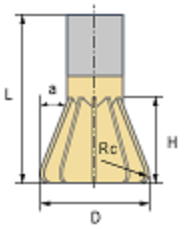
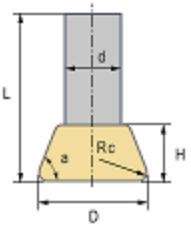
Тип tool и свойства API	Вид
Torus Mill OverallLength (L) CuttingDiameter (D) WorkingLength (WL) ShankDiameter (SHD) ShoulderLength (SHL) ShankTaperAngle (STA) Radius (Rc)	
Double Radial Mill OverallLength (L) CuttingDiameter (D) WorkingLength (WL) ShankDiameter (SHD) ShoulderLength (SHL) ShankTaperAngle (STA) Radius (Rc) TipRadius (R)	
Limited Double Radial Mill OverallLength (L) CuttingDiameter (D) WorkingLength (WL) ShankDiameter (SHD) ShoulderLength (SHL) ShankTaperAngle (STA) Radius (Rc) TipRadius (R) ShapeLength (H)	
Conical Mill OverallLength (L) CuttingDiameter (D) WorkingLength (WL) ShankDiameter (SHD) ShoulderLength (SHL) ShankTaperAngle (STA) Radius (Rc)	

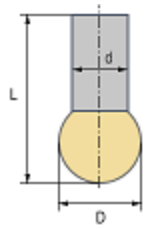
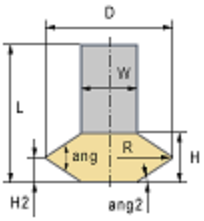
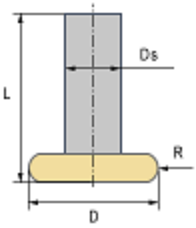
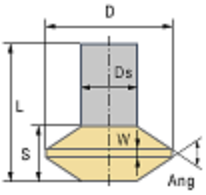
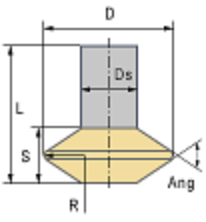
Тип tool и свойства API	Вид
TipRadius (R) Angle (A)	
Limited Conical Mill OverallLength (L) CuttingDiameter (D) WorkingLength (WL) ShankDiameter (SHD) ShoulderLength (SHL) ShankTaperAngle (STA) Radius (Rc) TipRadius (R) Angle (A) ShapeLength (H)	
Engraver OverallLength (L) CuttingDiameter (D) WorkingLength (WL) ShankDiameter (SHD) ShoulderLength (SHL) ShankTaperAngle (STA) Angle (A) EndDiameter (d)	
Drill OverallLength (L) CuttingDiameter (D) WorkingLength (WL) ShankDiameter (SHD) ShoulderLength (SHL) ShankTaperAngle (STA) Angle (A)	
Two Stage Drill OverallLength (L) WorkingLength (WL) Diameter1 (D1) Diameter2 (D2) Length1 (L1)	

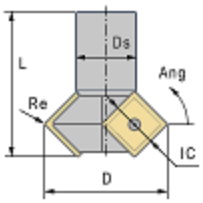
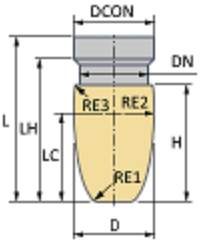
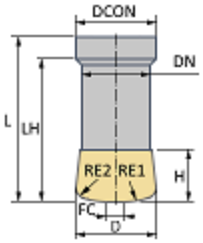
Тип tool и свойства API	Вид
Length2 (L2) Angle1 (a1) Angle2 (a2)	
Spot Drill OverallLength (L) CuttingDiameter (D) WorkingLength (WL) ShankDiameter (SHD) ShoulderLength (SHL) ShankTaperAngle (STA) Angle (A)	
Center Drill OverallLength (L) CuttingDiameter (D) WorkingLength (WL) ShankDiameter (SHD) ShoulderLength (SHL) ShankTaperAngle (STA) Angle (A)	
Reamer OverallLength (L) CuttingDiameter (D) WorkingLength (WL) ShankDiameter (SHD) ShoulderLength (SHL) ShankTaperAngle (STA)	
Counter Sink Drill OverallLength (L) CuttingDiameter (D) WorkingLength (WL) ShankDiameter (SHD) ShoulderLength (SHL) ShankTaperAngle (STA) Angle (A)	

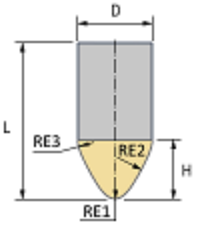
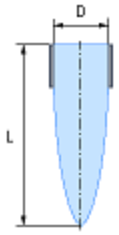
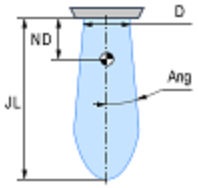
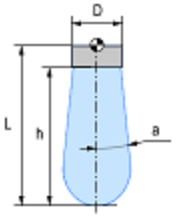
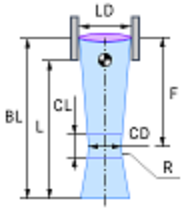
Тип tool и свойства API	Вид
Flat Tip Countersink OverallLength (L) Diameter (D) WorkingLength (WL) ShaftDiameter (DS) TipDiameter (TD) ShoulderLength (SL) TipAngle (A)	
Counter Bore Drill OverallLength (L) CuttingDiameter (D) WorkingLength (WL) ShankDiameter (SHD) ShoulderLength (SHL) ShankTaperAngle (STA) Angle (A)	
Pilot Counterbore OverallLength (L) Diameter (D) WorkingLength (WL) ShaftDiameter (DS) PilotDiameter (PD) ShoulderLength (SL) FluteLength (FL) PilotLength (PL) PilotRadius (PR)	
Boring Bar OverallLength (L) CuttingDiameter (D) WorkingLength (WL) ShankDiameter (SHD) ShoulderLength (SHL) ShankTaperAngle (STA)	

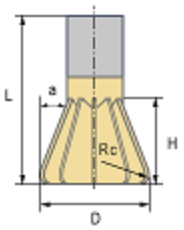
Тип tool и свойства API	Вид
Back Boring Bar OverallLength (L) CuttingDiameter (D) WorkingLength (WL) Height1 (h1) Height2 (h2) Angle1 (a1) Angle2 (a2) StemDiameter (Ds) StemAngle (Sa) StemHeight1 (sh1) StemHeight2 (sh2)	
Tap OverallLength (L) CuttingDiameter (D) WorkingLength (WL) ShankDiameter (SHD) ShoulderLength (SHL) ShankTaperAngle (STA) Angle (A) ThreadPitch (TP)	
Thread Mill OverallLength (L) CuttingDiameter (D) WorkingLength (WL) StemDiameter (Ds) Pitch (P) PitchCount (c) OD_ID (OD/ID)	
Negative Radius Mill OverallLength (L) CuttingDiameter (D) WorkingLength (WL) ShankDiameter (SHD) ShoulderLength (SHL)	

Тип tool и свойства API	Вид
ShankTaperAngle (STA) Radius (Rc)	
Knife OverallLength (L) Diameter (D) FrontDistance (RD) BackDistance (LD) FrontAngle (RA) BackAngle (LA) AutoDepth (LD) InclinationAngle (IA)	
Saw Blade WorkingLength (WL) Diameter (D) Thickness (Th) RoundingRadius (R)	
Undercut Dove Mill OverallLength (L) Diameter (D) WorkingLength (WL) Angle (a) CornerRadius (Rc) ShapeLength (H)	
Undercut Slot Mill OverallLength (L) Diameter (D) WorkingLength (WL) ShaftDiameter (d) SideAngle (a) CornerRadius (Rc) ShapeLength (H)	

Тип tool и свойства API	Вид
Undercut Lollipop Mill OverallLength (L) Diameter (D) WorkingLength (WL) ShaftDiameter (d)	
Undercut Two Angle Mill OverallLength (L) Diameter (D) WorkingLength (WL) Height (H) HeadHeight (H2) Angle1 (ang) Angle2 (ang2) SmoothRadius (R) ShaftDiameter (W)	
Undercut Round Groove Mill OverallLength (L) Diameter (D) WorkingLength (WL) Radius (R) StemDiameter (Ds)	
Undercut Sharp Chamfer Mill OverallLength (L) Diameter (D) WorkingLength (WL) Height (S) Angle (Ang) Width (W) StemDiameter (Ds)	
Undercut Rounded Chamfer Mill OverallLength (L) Diameter (D) WorkingLength (WL) Height (S) Angle (Ang)	

Тип tool и свойства API	Вид
Radius (R) StemDiameter (Ds)	
Undercut Indexable Chamfer Mill OverallLength (L) Diameter (D) WorkingLength (WL) StemDiameter (Ds) InsertAngle (Ang) InsertInnerCircleDiameter (IC) InsertCornersCount (CNC) CornerRadius (Re)	
Undercut Barrel Mill OverallLength (L) Diameter (D) WorkingLength (WL) CenterLength (LC) Height (H) StemLength (LH) EdgeR1 (RE1) EdgeR2 (RE2) EdgeR3 (RE3) StemDiameter (DN) ConnectDiameter (DCON)	
Undercut Lens Barrel Mill OverallLength (L) Diameter (D) WorkingLength (WL) FacetDiameter (FC) EdgeR1 (RE1) EdgeR2 (RE2) Height (H) StemLength (LH) StemDiameter (DN) ConnectDiameter (DCON) Angle (a)	

Тип tool и свойства API	Вид
Undercut Taper Barrel Mill OverallLength (L) Diameter (D) WorkingLength (WL) Height (H) EdgeR1 (RE1) EdgeR2 (RE2) EdgeR3 (RE3)	
Jet Cutter OverallLength (L) Diameter (D) WorkingLength (WL) ShankDiameter (SHD) ShoulderLength (SHL) ShankTaperAngle (STA)	
Taper Jet Cutter OverallLength (L) Diameter (D) WorkingLength (WL) JetLength (JL) NozzleDistance (ND) TaperAngle (Ang)	
Plazma Jet Cutter OverallLength (L) Diameter (D) WorkingLength (WL) Length (h) Angle (a)	
Laser Beam Cutter OverallLength (L) WorkingLength (WL) LensDiameter (LD) FocalDistance (F) CylinderZoneDiameter (CD) CylinderZoneLength (CL)	

Тип tool и свойства API	Вид
CurvatureRadius (R) LaserBeamLength (BL)	
Single Vacuum Gripper OverallLength (L) WorkingLength (WL) Diameter (D) ShaftDiameter (shd) CornerRadius (Rc) ShapeLength (H)	

Полезные ссылки

- Работа с milling tools в интерфейсе CAM-системы
- Подготовка окружения для Machining Tools Import
- Примеры Machining Tools Import в GitVerse

Далее

- Работа с tool holder

Работа с turning tools

Turning tool состоит из insert и tool holder. Сначала выберите группу turning tools, затем совместимые типы tool holder и insert, а также tool hand direction. Общие атрибуты, например tool name и tool number, одинаковы для всех подтипов cutting tools. Ниже приведён пример создания turning tool с external tool holder:

```
//C#
using THDT = MachinigToolsImportTypes.TMTI_TurnToolHolderDimensionTypes;
using TIDT = MachinigToolsImportTypes.TMTI_TurnToolInsertDimensionTypes;
------------------------------------------------------------
//importer is an instance of IMTI_MachiningToolsImportLibrary
var tool = importer.CreateTurnToolWithExternalHolder();
tool.SetName("My Turn tool");
tool.SetIdentifier("t001");
tool.SetMagazineNumber(1);
tool.SetToolNumber(1);
tool.SetUnits(TMTI_LinearUnits.luMillimeter);
tool.SetDurability(70);

tool.GetHandType().Hand = TMTI_Hand.hRight;

bool isCompatable = tool.SetHolderAndInsertType(
    TMTI_ExternalToolHolderTypes.htA_90deg,
    TMTI_ExternalToolHolderInsertTypes.itC_80degRhombic);
if (isCompatable) {
    var tdim = tool.GetDimensions();
    tdim.Holder[THDT.hdL1] = 130;
    tdim.Holder[THDT.hdB] = 20;
    tdim.Holder[THDT.hdL3] = 30;
    tdim.Holder[THDT.hdF1] = 30;

    tdim.Insert[TIDT.idL] = 17;
    tdim.Insert[TIDT.idTi] = 2;
    tdim.Insert[TIDT.idRe] = 0.1;
    //if you have changed this property, the holder type is automatically set
to undefined
    //tdim.Insert[TIDT.idKr] = 90;
}

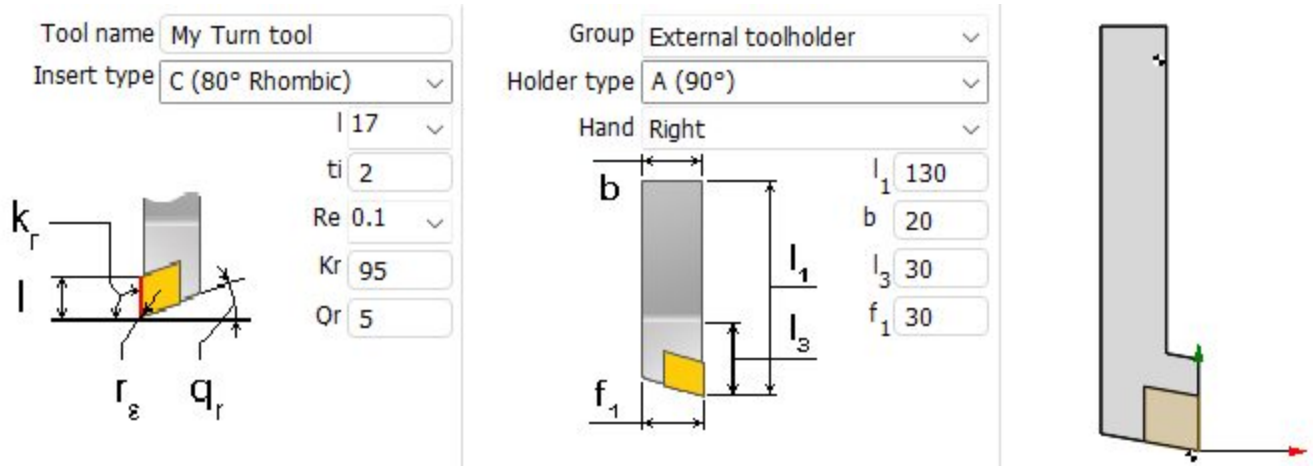
var tdir = tool.GetDirections();
tdir.FixingDirection = TMTI_FixingDirection.fdDirect;
tdir.CuttingDirection[TMTI_CuttingDirections.cdS] = true;
tdir.CuttingDirection[TMTI_CuttingDirections.cdE] = true;
tdir.CuttingDirection[TMTI_CuttingDirections.cdSE] = true;
```

```
var overh = tool.GetOverhang();
overh.isAutoCalc = false;
overh.Axial = -10;
overh.Radial = 121;
overh.Matching = 1.5;

var tpoints = tool.GetToolingPoints();
tpoints.Corrector[0] = 1;
tpoints.Corrector[1] = 99;
tpoints.TPointX[0] = -2.22;
tpoints.TPointX[1] = 1.11;
tpoints.TPointY[0] = -1.55;
tpoints.TPointY[1] = -0.29;

var cond = tool.GetCuttingConditions();
cond.RotationDirection = TMTI_RotationDirection.rdCCW;
cond.CuttingSpeedMode = TMTI_CuttingSpeedMode.csmCSS;
cond.Coolant.TubeIsOn[0] = false;
cond.Coolant.TubeIsOn[1] = true;
cond.Coolant.TubeIsOn[2] = true;
cond.FeedValue = 2;
//after setting the properties, add this tool to your tool library instance
```

Результат выполнения кода:



Совместимые варианты частей tool и их геометрические размеры можно посмотреть в интерфейсе CAM-системы.

Полезные ссылки

- Работа с turning tools в интерфейсе CAM-системы
- Подготовка окружения для Machining Tools Import
- Примеры Machining Tools Import в GitVerse

Далее

- Работа с tool holder

Работа с custom axial shape tools

Custom axial shape tool создаётся по кривой, построенной по точкам на плоскости. Общие атрибуты, например tool name и tool number, одинаковы для всех подтипов cutting tools. Ниже приведён пример создания custom axial shape tool в группе Spherical Mills.

```
//C#
//importer is an instance of IMTI_MachiningToolsImportLibrary
var tool = importer.CreateCustomAxialShapeTool();
tool.ToolGroup = TMTI_AxialToolGroup.tgSpherical;
tool.OverallLength = 100;
tool.SetName("My Custom Spherical Mill Tool");
tool.SetIdentifier("s001");
tool.SetTeethsCount(2);
tool.SetMagazineNumber(1);
tool.SetToolNumber(1);
tool.SetUnits(TMTI_LinearUnits.luMillimeter);
tool.SetDurability(70);
tool.SetMaxPlungeAngle(90);

var rcv = tool.BeginGeneratrix();
TST2DPoint p = new TST2DPoint() {X = 1, Y = 1};
TST2DPoint pc = new TST2DPoint() {X = 0.0359, Y = 7.3301};
rcv.SetSpanType(TMTI_CurveSpanType.csfWorking);
rcv.StartCurve(p);
p.X = 6; p.Y = 5;
rcv.ArcTo(pc, p, 6.4031);
p.X = 6; p.Y = 25;
rcv.CutTo(p);
rcv.SetSpanType(TMTI_CurveSpanType.csfNonWorking);
p.X = 6; p.Y = 30;
rcv.CutTo(p);
p.X = 7.5; p.Y = 40;
rcv.CutTo(p);

var toolings = tool.GetToolingPoints();
toolings.ToolContactPointType = TMTI_AxialToolContactPointType.cpCustomPoint;
toolings.ToolContactPointsCount = 1;

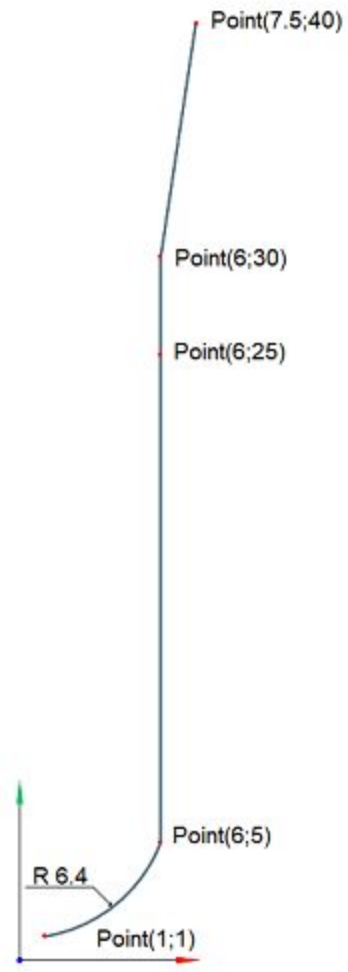
toolings.ToolingPointCount = 1;
toolings.ToolingPointType[0] = TMTI_AxialToolToolingPointType.tpCustomPoint;
toolings.ToolingPointShift[0] = 1.5;
toolings.ToolingPointLengthCorrectorNumber[0] = 1;
toolings.ToolingPointRadiusCorrectorNumber[0] = 1;
```

```
var cond = tool.GetCuttingConditions();
cond.CuttingSpeedMode = TMTI_CuttingSpeedMode.csmRPM;
cond.RotationsPerMinute = 100;
cond.SpindleGearRange = 1;
cond.RotationDirection = TMTI_RotationDirection.rdCW;
cond.FeedUnits = TMTI_FeedUnits.perMinute;
cond.FeedValue = 100;

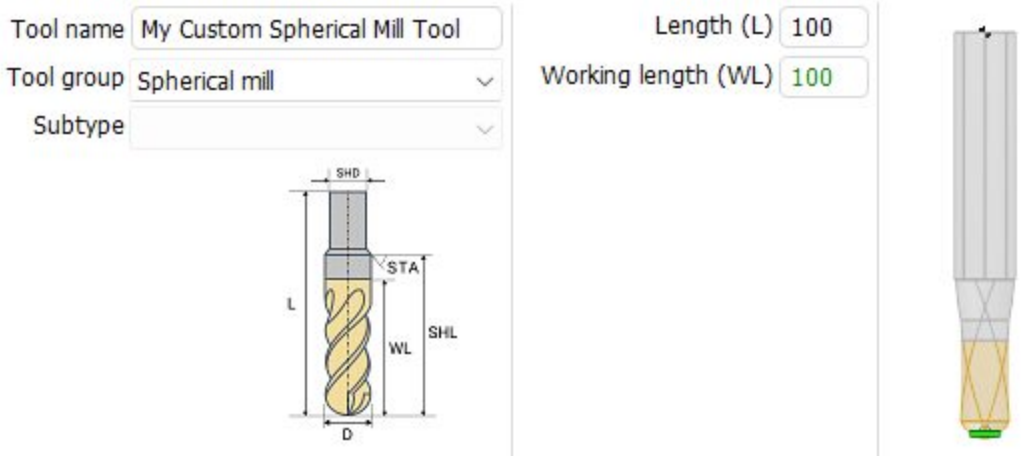
int tubesCount = cond.Coolant.TubeCount; //20
cond.Coolant.TubeIsOn[0] = true;
cond.Coolant.TubeIsOn[1] = false;
cond.Coolant.TubeIsOn[2] = true;

//after setting the properties, add this tool to your tool library instance
```

Полученная кривая:



Результат выполнения кода:



Полезные ссылки

- Работа с custom axial shape tools в интерфейсе CAM-системы
- Подготовка окружения для Machining Tools Import
- Примеры Machining Tools Import в GitVerse

Далее

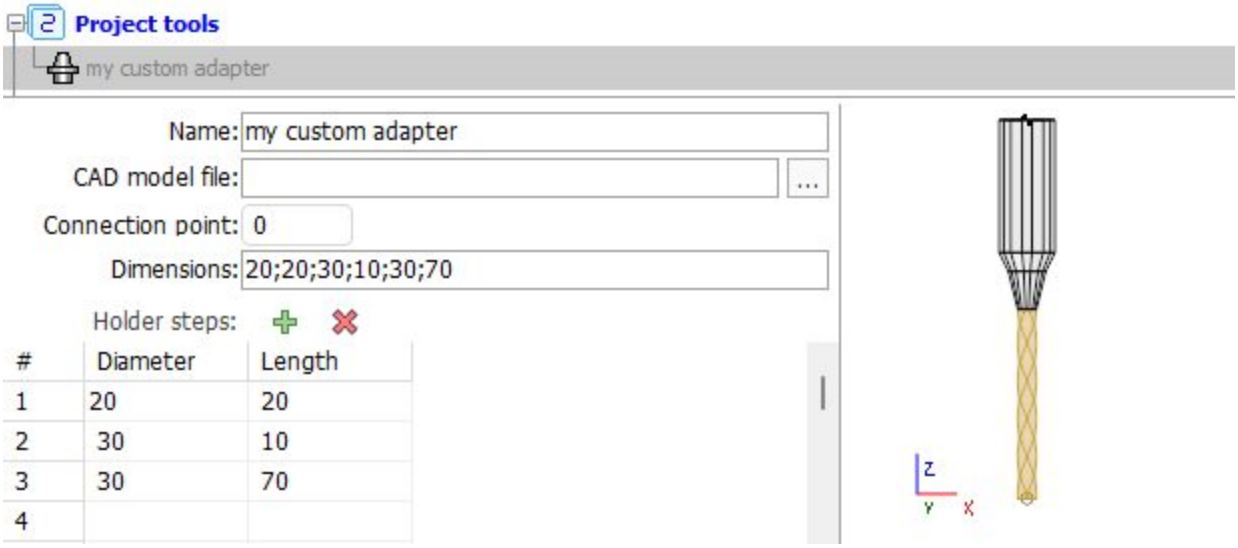
- Работа с tool holder

Работа с tool holder

Tool holder можно задать двумя способами: загрузить CAD model из файла или создать средствами CAM-системы. Поддерживаются CAD models в форматах `.stl` и `.osd`. В конструкторе можно создать ступенчатый tool holder, задав диаметр и длину каждой ступени. Ниже приведён пример создания tool holder с помощью конструктора CAM-системы:

```
//C#
//importer is an instance of IMTI_MachiningToolsImportLibrary
var tool = importer.CreateCylindricalMill();
tool.SetName("My Cylindrical Mill Tool with custom tool holder");
var adapter = tool.GetStepsAdapter();
adapter.HolderName = "my custom adapter";
adapter.HolderStepCount=3;
adapter.HolderStepDiameter[0] = 20;
adapter.HolderStepHeight[0] = 20;
adapter.HolderStepDiameter[1] = 30;
adapter.HolderStepHeight[1] = 10;
adapter.HolderStepDiameter[2] = 30;
adapter.HolderStepHeight[2] = 70;
```

Результат выполнения кода:



Для turning tools задание tool holder возможно только через CAD model.

Полезные ссылки

- Работа с tool holder в интерфейсе CAM-системы
- Подготовка окружения для Machining Tools Import
- Примеры Machining Tools Import в GitVerse

На главную страницу раздела

Взаимодействие с внешними приложениями

С помощью CAMIPC можно создать внешнее приложение, которое:

- Запускает CAM-систему и подключается к ней;
- Управляет CAM-системой: открывает проекты, рассчитывает NC program, редактирует список операций и выполняет другие действия;
- Запускает CAM-систему без окон, что ускоряет её работу;
- Запускает несколько экземпляров CAM-системы и считывает данные из каждого;
- Воспроизводит действия пользователя, например нажатие кнопок.

Список доступных возможностей расширяется.

Подключение

Для подключения к CAM-системе используется библиотека `CAMIPC.Helper.Cam.dll`. Варианты подключения:

- In C#;
- In Delphi;
- In C++.

Подключение к CAM-системе из приложения C#

В состав установленной CAM-системы входят:

- `CAMIPC.Helper.Cam.dll` с экспортируемой функцией `CreateHelper`. Она возвращает объект интерфейса IIpcHelper, методы которого позволяют подключиться к работающему экземпляру CAM-системы или запустить новый;
- `NativeLibUtils.dll` — библиотека, упрощающая загрузку и выгрузку native DLL.

В файл `.csproj` необходимо добавить:

- Ссылку на `NativeLibUtils.dll` с указанием пути;
- Ссылку на пакет CAMIPC SDK;

```
<ItemGroup>
    <PackageReference Include="SprutTechnology.CAMIPC.SDK.Net" Version="18.2.0" />
    <Reference Include="C:\Program Files\Sprut Technology\SprutCAM
18\Bin64\NativeLibUtils.dll" />
</ItemGroup>
```

Вспомогательный singleton

Открытый метод `GetApplication()` этого класса подключается к работающему экземпляру CAM-системы.

```
using System;
using System.IO;
using System.Runtime.InteropServices;
using CAMAPI.ResultStatus;
using CAMIPC.Application;
using CAMIPC.ExecuteContext;
using CAMIPC.Helper;
using SprutTechnology.NativeLibUtils;

namespace ApplicationEmptyNet;

public static class CamHelper
{
    private static IIpcHelper? _helper;
    private static ICamIpcApplication? _application;
    private static IntPtr _helperObjectPtr;
    private static IntPtr _helperDllPtr;
```

```
    private delegate IntPtr CreateHelperDelegate();

    /// <summary>
    /// Connect to CAM application
    /// </summary>
    private static IIpcHelper GetHelper()
    {
        if (_helper != null)
            return _helper;

        const string camFolder = @"C:\Program Files\Sprut Technology\SprutCAM
NB 18\Bin64";
        var helperPath = Path.Combine(camFolder, "CAMIPC.Helper.Cam.dll");
        if (!File.Exists(helperPath))
            throw new Exception($"{helperPath} not found");

        _helper = NativeLibLoader.CreateComObject<IIpcHelper, CreateHelperDelegate>(
                      helperPath, "CreateHelper", out _helperObjectPtr,
out _helperDllPtr)
                  ?? throw new Exception("Can't create helper");

        return _helper;
    }

    /// <summary>
    /// Get instance of running CAM application to interact with it
    /// </summary>
    public static ICamIpcApplication GetApplication()
    {
        if (_application != null)
            return _application;

        var executeContext = new TExecuteContext();
        var instances = GetHelper().GetRunningCamAppList(ref executeContext);
        if (executeContext.ResultStatus.Code == TResultStatusCode.rsError)
            throw new Exception(executeContext.ResultStatus.Description);
        if (instances.Count == 0)
            throw new Exception("SprutCAM running instance not found");

        _application = instances.Get(0, executeContext);
        if (executeContext.ResultStatus.Code == TResultStatusCode.rsError)
            throw new Exception(executeContext.ResultStatus.Description);

        return _application;
    }
```

```
    /// <summary>
    /// Destructor. Release COM objects
    /// </summary>
    public static void Clean()
    {
        if (_helper != null)
        {
            if (_application != null)
                Marshal.ReleaseComObject(_application);
            NativeLibLoader.FreeDll(_helper, _helperObjectPtr, _helperDllPtr);
        }
    }
}
```

Изменение singleton

Интерфейс IIpcHelper предоставляет методы для следующих действий:

- Подключиться к первому найденному экземпляру, как в примере выше;
- Подключиться к CAM-системе по PID процесса;
- Запустить новый экземпляр CAM-системы и подключиться к нему.

Подключение к CAM-системе из приложения Delphi

Подключение к CAM-системе из приложения C++

Использование ComWrapper

Ядро CAM-системы написано на языке с ручным освобождением памяти. В C# и других языках с Garbage Collector память освобождается автоматически.

Поэтому при взаимодействии .NET-расширений с ядром CAM-системы требуется механизм явного освобождения COM-объектов. Класс `ComWrapper` объединяет логику управления памятью этих объектов.

Использование `ComWrapper` решает две задачи:

1. Доступ к COM-объекту из нескольких потоков;
2. Корректное освобождение памяти после использования, чтобы ядро CAM-системы могло продолжить работу.

Примеры

Получение другого COM-объекта:

```
using var ncMakerCom = activeProjectCom.InvokeAndWrap(project => project.NCMaker);
```

Вызов метода COM-объекта:

```
applicationCom.Invoke(application =>
{
    application.MainWorkMode = TMainWorkMode.mwmModel;
});
```

Передача ownership:

```
var curveCom2 = curveCom1.TransferOwnership();
```

Требование к .NET-расширениям

Использование `ComWrapper` обязательно для расширений на .NET.