Справочник по настройке CAM-системы

Эта папка содержит документацию для разработчиков и интеграторов по **настройке CAM-системы через её декларативный XML** — добавлению собственных операций, станков и значений по умолчанию — вместе со смежными темами. Она предназначена для **OEM, интеграторов и партнёров**, а также для **продвинутых пользователей**, которые настраивают систему.

Всё здесь построено на одном фундаменте — фреймворке **XMLProperties**, который декларативно описывает настраиваемые объекты системы. **Начните с фреймворка** (там же изложены общие правила, на которые опираются все разделы), затем переходите к разделам по операциям и станкам.

Разделы документации

- **Фреймворк XMLProperties** — **начните отсюда:** сам декларативный фреймворк свойств (мысленная модель, синтаксис дескрипторов, язык выражений, использование свойств из кода, пользовательские значения по умолчанию) **и общие правила**, на которые опираются все разделы — почему нельзя редактировать `Supplement`, где лежат поставляемые файлы и какие соглашения используются в примерах.
- **XML-дескрипторы операций** — описание и регистрация технологических операций, а также полное дерево наследования операций.
- **Дескрипторы станков** — описание станка: структура файлов, кинематическая схема, параметры *Machine Setup*, продвинутые темы и 3D-модели.

В папке `Supplement/docs` может находиться и другая, не связанная с этим документация; всё, что касается фреймворка XMLProperties и его дескрипторов, расположено в перечисленных выше разделах.

Фреймворк XMLProperties

Фреймворк **XMLProperties** — это способ, которым CAM-система **декларативно** описывает свои настраиваемые объекты (операции, станки, инструменты, …). Вместо жёсткого программирования параметров фрезерной операции или кинематической схемы станка система загружает дерево **дескрипторов свойств** из XML. Каждый дескриптор несёт имя, тип значения, значение по умолчанию, метаданные отображения и необязательные правила (видимость, доступность, вычисляемые значения). Работающее приложение превращает это дерево в интерфейс инспектора, модель данных за каждым проектом и значения, доступные через CAM API. Поскольку модель декларативна, вы можете добавить параметр, перегруппировать инспектор, изменить значение по умолчанию или описать совершенно новый станок — всё это редактированием XML, без перекомпиляции приложения.

Этот раздел документирует сам фреймворк — часть, общую для *любого* вида дескрипторов. Разделы по операциям и станкам опираются на него: Операции, Станки.

Читайте сверху вниз — каждая страница опирается на предыдущую:

- **Обзор фреймворка** — мысленная модель: типы и экземпляры, пространства имён, как файлы загружаются и объединяются и куда помещать ваши файлы.
- **Синтаксис дескрипторов** — основной справочник: `SCType`, `SCProperty`, система типов, наследование и переопределение, полный каталог атрибутов.
- **Язык выражений** — вычисляемые значения для `DefaultValue`, `Visible`, `Enabled` и подобных: операторы, функции, ссылки на свойства и аксессоры.
- **Использование XML-свойств из кода** — единый шаблон доступа, общий для CAM API, нативного Delphi/C++ и постпроцессоров (.NET и `.sppx`).
- **Пользовательские значения по умолчанию** — переопределение поставляемых *значений по умолчанию* своими, хранящимися в файлах `.usrdef` и применяемыми без редактирования `Supplement`.

(Целевая аудитория указана в корне документации.)

Общие правила

Они действуют во *всех* разделах — включая операции и станки — поэтому прочитайте их один раз здесь.

> **Не редактируйте папку `Supplement`.** Файлы в `Supplement` — это дистрибутивные файлы системы, и они заменяются при обновлении, поэтому любые ваши изменения в них теряются. Ваш пользовательский контент размещается в отдельных, выделенных папках, которые система загружает автоматически. Файлы `Supplement` при этом остаются вашим лучшим *справочником*

> — изучайте их, чтобы понять шаблоны, — но свои файлы создавайте в другом месте. См. Куда помещать ваши файлы.

Где лежат поставляемые файлы

Поставляемые файлы дескрипторов расположены в **корне установки программы**, например:

```
C:\Program Files\<Vendor>\<Product>\Supplement\
```

Это системные файлы только для чтения — используйте их как справочник, но не изменяйте. Загрузка начинается с небольшого набора корневых файлов в `Supplement`, каждый из которых подключает остальные через `<SCInclude>` — включая необязательные папки-расширения с масками, откуда подхватываются ваши собственные файлы:

Корневой файл	Назначение
SCConfig.xml	Главная точка входа — общие типы, операции, схема проекта, пользовательские умолчания
Machines/MachinesConfig.xml	Дескрипторы станков
STApplicationConfig.xml	Настройки уровня приложения
Postprocessor/PostprocessorConfig.xml	Дескрипторы постпроцессоров

Соглашения в примерах

Короткие *синтетические* фрагменты (например, типы с именами `TFoo`, `Bar`) иллюстрируют чистый синтаксис. *Реальные* фрагменты, процитированные из поставляемых файлов `Supplement`, показывают, как правила применяются на практике; они ссылаются на исходный файл, чтобы вы могли увидеть полный контекст. Пути к файлам внутри каждого раздела указаны относительно папки `Supplement`.

Что лежит на этом фреймворке

Операции и станки (описанные в своих разделах) — это основное, что вы настраиваете с помощью этого фреймворка. Другие семейства дескрипторов — инструменты, конфигурация постпроцессора и конфигурация приложения — построены на **том же** фреймворке, но представляют собой внутренние данные сериализации, поддерживаемые разработчиками системы и не предназначенные для настройки; поэтому подробно они не документируются.

Обзор фреймворка

Эта глава даёт мысленную модель, лежащую в основе фреймворка XMLProperties. Как только вы поймёте четыре ключевые идеи ниже, остальной синтаксис — это детали.

1 Для чего нужен фреймворк

Каждый настраиваемый объект CAM-системы — технологическая операция, станок, инструмент, проект — описывается деревом **свойств**. У свойства есть имя, значение, тип, значение по умолчанию и метаданные, которые сообщают интерфейсу, как его отобразить, а движку — как себя вести.

Эти деревья свойств не пишутся в коде. Они объявляются в XML-файлах дескрипторов и загружаются при старте. Это значит, что вы можете:

- добавлять и удалять параметры операции или станка,
- менять значения по умолчанию (в том числе отдельные для метрической и дюймовой систем),
- перестраивать группировку и отображение параметров в инспекторе,
- делать параметр видимым, скрытым, доступным или недоступным в зависимости от других параметров,
- определять совершенно новую операцию или станок,

— и всё это редактированием XML.

2 Типы и экземпляры

Это важнейшее различие во всём фреймворке.

- **Дескриптор (тип)** — это *шаблон*. Он говорит: «блок `RoughPasses` содержит логическое свойство `Enabled` со значением по умолчанию `true` и `Step`, равный `1 мм`». Дескрипторы объявляются через `<SCType>` (а на верхнем уровне — `<SCProperty>`). Они образуют цепочку наследования: тип может базироваться на другом типе и расширять или переопределять его.

- **Экземпляр (указатель свойства)** — это *конкретное значение*, привязанное к конкретному объекту: например, `RoughPasses.Step` у одной конкретной операции торцевания в вашем проекте. Экземпляры создаются из дескрипторов при создании объекта и хранят реальные, редактируемые данные.

Вы создаёте **дескрипторы**. Приложение создаёт из них **экземпляры**. Когда вы читаете или записываете значения через CAM API (см. Использование XML-свойств из кода), вы работаете с экземплярами.

Важное следствие: экземпляр свойства, значение которого всё ещё равно значению по умолчанию из его дескриптора, *не* хранится дословно — оно вычисляется из дескриптора по запросу. Поэтому

изменение значения по умолчанию в XML влияет на все существующие экземпляры, которые не были изменены явно.

3 Форма дерева дескрипторов

Дескрипторы вкладываются друг в друга. Дескриптор `ComplexType` содержит дочерние дескрипторы; те, в свою очередь, тоже могут быть составными; массивы содержат единственного потомка, который служит шаблоном элемента; перечисления содержат свои варианты как потомков. Поэтому дескриптор одной операции — это глубокое дерево, листья которого — простые значения (числа, строки, логические значения, выбранные варианты перечисления).

```
<SCType ID="TFinishPasses" type="ComplexType">
    <SCType ID="Enabled" type="Boolean" DefaultValue="true"/>
    <SCType ID="Step"    type="Double"  DefaultValue="1" DimensionKind="Linear"/>
    <SCType ID="Count"   type="Integer" DefaultValue="1"/>
</SCType>
```

Дальнейшая навигация по дереву (в выражениях и в API) использует точечные пути вида `FinishPasses.Step`.

4 Пространства имён

Дескрипторы сгруппированы в **пространства имён**. Пространство имён — это самодостаточный словарь имён типов: тип операции `TSTFaceMillingOp` живёт в пространстве `Operations`, тип станка — в пространстве `Machines`, и так далее. Два пространства имён могут содержать типы с одинаковыми именами, не конфликтуя.

Пространства имён объявляются через `<SCNameSpace ID="...">`. Один и тот же физический файл типов часто включается более чем в одно пространство (например, общий `CommonTypes.xml` включается почти везде), так что общие строительные блоки доступны там, где они нужны.

Основные пространства имён, которые вам встретятся:

Пространство имён	**Содержимое**
`Operations`	Типы технологических операций
`Machines`	Типы станков и их кинематические строительные блоки
`Project`	Схема проекта / установа
`Application`	Настройки уровня приложения

Пространство имён	Содержимое
`Postprocessor`	Дескрипторы постпроцессоров
`OperationRegistrator`	Небольшие «регистрационные» записи, объявляющие, какие типы являются настоящими операциями (см. Дескрипторы операций)

5 Как файлы загружаются и объединяются

Загрузка начинается с корневого файла (например, `SCConfig.xml`) и следует директивам `<SCInclude>`, собирая всю модель. Важны три правила:

1. **Порядок имеет значение.** Файлы обрабатываются сверху вниз. Тип должен быть объявлен до того, как он используется в качестве базового, а более поздние объявления могут переопределять более ранние. Именно поэтому корневые файлы подключают абстрактные/базовые типы раньше конкретных, которые от них наследуются.

2. **Включения могут быть необязательными и с масками.** `Optional="True"` означает «загрузить, если файл существует, иначе молча пропустить»; маски вроде `*_ExtOp.xml` позволяют добавлять файлы-расширения в папку, не редактируя корневой конфиг. Несколько таких необязательных включений с масками указывают на папки **вне** `Supplement` — именно так ваш пользовательский контент загружается, не затрагивая системные файлы. См. §7.

3. **Пути используют файловые переменные.** Вместо абсолютных путей включения (и ссылки на файлы, например иконки) используют подстановки `$(VARIABLE)`, которые во время выполнения превращаются в реальные папки. Список см. в §7.

```
<!-- из SCConfig.xml -->
<SCInclude>$(SUPPLEMENT_FOLDER)\CommonTypes.xml</SCInclude>
<SCInclude>$(SUPPLEMENT_FOLDER)\Operations.xml</SCInclude>
<!-- необязательное, с маской, загружается только при наличии -->
<SCInclude Optional="True">$(LOCAL_OPERATIONS_FOLDER)\*.usrdef</SCInclude>
```

> **Не у каждого пространства имён есть корневой файл на диске.** Сборка пространства имён — это просто «открыть пространство, загрузить в него последовательность файлов, закрыть». Эта последовательность обычно объявляется в XML (`<SCNameSpace>` + `<SCInclude>`), но приложение может также собрать пространство имён **в памяти при старте**, загрузив те же файлы строительных блоков по порядку. Пространство `Project` собирается именно так — единого корневого конфига проекта нет; система загружает `CommonTypes.xml`, `ProjectSchema/CommonProjectSchema.xml`, `ProjectSchema/ProjectSchema.xml` и ещё несколько в

> новое пространство `Project`. Результат идентичен декларативной форме — поэтому, если вы не можете найти корневой файл для пространства имён, вот почему.

6 Чем управляет фреймворк

Одно дерево дескрипторов питает сразу несколько подсистем. Понимание этого объясняет, почему дескриптор несёт столько разных атрибутов:

- **Модель данных** — фактические значения, хранящиеся в проекте.
- **Интерфейс инспектора** — подписи, группировка, порядок, иконки, видимость/доступность строк и специальные редакторы. Большинство «оформительских» атрибутов (`Caption`, `Visible`, `Compact`, `Category`, `ImageFile`, …) существуют именно для этого.
- **Вычислительный движок** — решатели и конвейер построения траектории/УП читают значения свойств по имени.
- **CAM API** — внешний код читает и записывает те же свойства.

Следующая глава — справочник по синтаксису, на котором всё это построено.

7 Куда помещать ваши файлы

Папка `Supplement` — это **системный контент только для чтения**: она перезаписывается при каждом обновлении, поэтому любые ваши изменения в ней теряются. Вместо этого корневые конфиги подключают ряд *необязательных* расположений с масками **вне** `Supplement`, зарезервированных под ваш контент. Положите свои файлы туда — и они загрузятся автоматически вместе с системными дескрипторами, в правильном порядке.

Расположения адресуются через файловые переменные, чтобы корректно разрешаться для каждой установки и каждого пользователя:

Что вы хотите добавить…	**Как**	**Кем загружается**
Новый / расширенный **тип операции** (автоматически по имени)	положите файл `*_ExtOp.xml` в `$(COMMON_CONTAINERS_FOLDER)`	`Operations.xml`
Новый **тип операции** (регистрация через мастер)	зарегистрируйте XML-файл операции в *Operations Manager* (см. ниже) → он добавляется в `$(OPERATIONS_FOLDER)\UserOperationsList.xml`	`Operations.xml`

Что вы хотите добавить…	Как	Кем загружается
Переопределения пользовательских значений по умолчанию	поместите файлы `*.usrdef` в `$(LOCAL_OPERATIONS_FOLDER)`	`SCConfig.xml`
Пользовательский станок	создайте его инструментами построения станков; хранится в `$(SCHEMAS_FOLDER)` / `$(CUSTOM_SCHEMAS_FOLDER)`	подсистема построения станков

Рекомендации:

- **Операции — два способа.** Есть два независимых механизма добавления пользовательской операции, и оба загружаются *после* системных операций, поэтому любой из них может объявлять новые типы операций или переопределять существующие:
  - **По соглашению об именах.** Положите файл `*_ExtOp.xml` в `$(COMMON_CONTAINERS_FOLDER)`. Любой файл, подходящий под маску, подхватывается автоматически — больше ничего настраивать не нужно.
  - **Через мастер Operations Manager** (меню **Utilities** → **Operations Manager**). Вы указываете мастеру XML-файл вашей операции (хранится где угодно); он регистрирует этот файл, добавляя запись в `$(OPERATIONS_FOLDER)\UserOperationsList.xml`. Этот список *ведётся мастером* — вручную его не редактируют.

  В обоих случаях сам файл операции — это обычный файл дескриптора (`<SCCollection>` в пространстве имён `Operations`); различается только способ его регистрации.
- **Станки.** Пользовательские станки обычно создаются интерактивно приложением **MachineMaker**, которое записывает их кинематические схемы в папки схем станков, а не в `Supplement`. Рукописный XML — только для особых случаев; синтаксис дескриптора станка, на котором строятся эти схемы, описан в разделе Дескрипторы станков.
- **Читайте `Supplement`, но не редактируйте его.** Относитесь к поставляемым файлам исключительно как к справочнику по шаблонам и базовым типам, на которых вы строите своё.

Механика одинакова, где бы ни лежал файл: тот же корень `<SCCollection>`, те же пространства имён, тот же синтаксис. Различается только *папка*.

Далее: **Синтаксис дескрипторов**

Синтаксис дескрипторов

Это основная справочная глава. Она охватывает XML-элементы, систему типов, работу наследования и переопределения, а также каталог атрибутов, которые можно задать на дескрипторе.

1 Структура документа

Каждый файл дескриптора — это XML-документ с корневым элементом `<SCCollection>`. Внутри него размещаются пространства имён, включения, объявления типов и объявления свойств.

```
<?xml version="1.0" encoding="UTF-8" ?>
<SCCollection>
    <SCNameSpace ID="Operations">
        <SCInclude>$(SUPPLEMENT_FOLDER)\CommonTypes.xml</SCInclude>
        <SCType ID="TFoo" type="ComplexType"> ... </SCType>
    </SCNameSpace>
</SCCollection>
```

Структурные элементы:

Элемент	Назначение
`SCCollection`	Корень документа. Присутствует всегда.
`SCNameSpace`	Открывает пространство имён (`ID="Operations"`, `"Machines"`, …). Всё внутри объявляется в этом пространстве. Может быть вложенным.
`SCInclude`	Подключает другой файл в этом месте. Поддерживает `Optional="True"` и маски.
`SCType`	Объявляет **тип** (переиспользуемый дескриптор) — или, будучи вложенным в другой тип, объявляет его **член**. Главный рабочий элемент.
`SCProperty`	Объявляет **именованное глобальное свойство** — одиночное значение/объект на уровне пространства имён (см. §6).

Файлы вне какого-либо `<SCNameSpace>` вносятся в то пространство имён, в которое их включают — поэтому у `CommonTypes.xml` нет собственного пространства имён: он включается во многие.

2 `SCType` — рабочая лошадка

`SCType` выполняет двойную роль в зависимости от того, где он появляется.

На верхнем уровне (или прямо внутри пространства имён) он *объявляет тип* — именованный переиспользуемый шаблон:

```
<SCType ID="T2DPoint" Caption="Point" type="ComplexType">
    <SCType ID="X" Caption="X" type="Double" DefaultValue="0" DimensionKind="Linear"/>
    <SCType ID="Y" Caption="Y" type="Double" DefaultValue="0" DimensionKind="Linear"/>
</SCType>
```

`ID` — имя типа; `type` указывает его **базу** (здесь встроенный `ComplexType`). Вложенные элементы `<SCType>` объявляют *члены* типа.

Вложенный в другой тип `<SCType>` *объявляет новый член* охватывающего типа. `ID` члена — это его имя внутри родителя, а `type` — тип значения члена, который может быть встроенным типом или любым ранее объявленным типом:

```
<SCType ID="TSTFaceMillingOp" type="TSTMillOp">
    <!-- член с именем MillingType, ранее объявленного типа TMillMode -->
    <SCType ID="MillingType" type="TMillMode" Caption="Milling type"/>
</SCType>
```

Итого: **`ID` + `type` = «создать нечто с именем *ID*, имеющее тип *type*»**, независимо от того, является ли это «нечто» типом верхнего уровня или членом составного типа.

3 Система типов

3.1 Встроенные (примитивные) типы

Атрибут `type` принимает следующие встроенные имена (регистр не важен — `Double` и `double` это одно и то же):

Встроенный `type`	**Хранит**	**Примечания**
`Boolean`	`True` / `False`	
`Integer`	целое число	
`Double`	вещественное число	Используйте `DimensionKind` для работы с единицами
`String`	текст	

Встроенный `type`	**Хранит**	**Примечания**
`Enumerated`	один вариант из фиксированного списка	Варианты — дочерние `SCType` с `type="none"`
`ComplexType`	запись из именованных членов	Потомки — это члены
`Array`	список однотипных элементов	Единственный потомок задаёт шаблон элемента; см. §5
`none` (или `None`)	ничего	Используется для вариантов перечислений, разделителей и чисто интерфейсных заглушек

Всё, что *не* входит в эти имена, трактуется как **ссылка на ранее объявленный тип** — именно так работают композиция и наследование.

3.2 Перечисления

Перечисление перечисляет свои варианты как дочерние `SCType` с `type="none"`. `DefaultValue` перечисления — это `ID` варианта по умолчанию.

```
<SCType ID="TXYZCoordinate" Caption="XYZ Coordinate" type="Enumerated"
DefaultValue="CoordX">
    <SCType ID="CoordAuto" Caption="Auto" type="none"/>
    <SCType ID="CoordX"    Caption="X"    type="none"/>
    <SCType ID="CoordY"    Caption="Y"    type="none"/>
    <SCType ID="CoordZ"    Caption="Z"    type="none"/>
</SCType>
```

(реальный пример: `CommonTypes.xml`)

Хранимое значение свойства-перечисления — это **`ID` варианта** (например, `CoordX`), а не его подпись. Каждый вариант может нести собственные метаданные (`Caption`, `ImageFile`, `Priority`, правило `Visible`, …).

3.3 Составные типы

`ComplexType` — это запись: её дочерние `SCType` являются её полями. Составные типы могут вкладываться произвольно и переиспользоваться как тип других членов.

```
<SCType ID="TColor" Caption="Color" type="ComplexType">
    <SCType ID="R" type="Double" DefaultValue="0.5"/>
    <SCType ID="G" type="Double" DefaultValue="0.5"/>
    <SCType ID="B" type="Double" DefaultValue="0.5"/>
</SCType>
```

В инспекторе `TColor` отображается как образец цвета с диалогом выбора, а не как три числовые строки — см. §3.4.

3.4 Типы с особым поведением

Несколько часто используемых типов объявлены в `CommonTypes.xml` как тонкие **псевдонимы** базового типа — вы ссылаетесь на них по имени, как на любой объявленный тип, — но каждый добавляет к базе дополнительное поведение при сериализации или в инспекторе. Объявляйте член одним из них, когда нужно такое поведение:

Тип	База	Что добавляет
`TranslatableString`	`String`	Его значение попадает в подсистему **локализации** и может быть переведено. По умолчанию локализуются только `Caption` узлов; используйте этот тип, когда нужно, чтобы переводилось и *значение* строки (обычно значение по умолчанию).
`CDATAString`	`String`	Сериализуется внутри секции `CDATA`, поэтому значение может содержать символы, иначе запрещённые в XML, — что позволяет сохранить исходное форматирование, например переносы строк и табуляции.
`FileName`	`String`	Хранит **путь к файлу**. При сохранении/загрузке путь автоматически преобразуется между абсолютной и относительной формой и с использованием файловых переменных, например `$(SUPPLEMENT_FOLDER)`. В инспекторе предлагает стандартный **диалог открытия файла**; атрибут `Filter` (вида `Caption (*.ext)|mask1;mask2`) ограничивает этот диалог выбранными типами файлов.
`DynamicArray`	`Array`	Ведёт себя как `Array` (§5), но в инспекторе показывает **количество элементов** в собственном поле значения строки и позволяет пользователю менять его, добавляя или удаляя дочерние элементы на лету.

Тип	База	Что добавляет
`TColor`	`ComplexType`	Цветовое значение (`R`/`G`/`B`, §3.3), отображаемое в инспекторе как **образец цвета** с диалогом выбора цвета.

(реальный пример: `CommonTypes.xml`. Фильтр диалога для `FileName` — поле постпроцессора станка — в `Machines/AbstractMachine.xml`: `Filter="Postprocessors (*.sppx, *.dll)|*.sppx;*.dll"`.)

4 Наследование и переопределение

4.1 Наследование типа

Чтобы основать новый тип на существующем, укажите существующий тип в `type`. Новый тип начинается со **всех** членов базы и затем добавляет члены, которые вы объявляете:

```
<SCType ID="T3DPoint" Caption="Point" type="T2DPoint">
    <!-- наследует X и Y из T2DPoint, добавляет Z -->
    <SCType ID="Z" type="Double" DefaultValue="0" DimensionKind="Linear"/>
</SCType>
```

(реальный пример: `CommonTypes.xml`)

Цепочки наследования могут быть длинными. Конкретная операция, например, наследуется от семейства абстрактных операций (`TSTFaceMillingOp` → `TSTMillOp` → … → `TOperationDescriptor`).

4.2 Переопределение унаследованного члена

Внутри тела типа можно написать две разные вещи, и различие между ними принципиально:

- `<SCType ID="Name" type="...">` — **объявляет *новый* член** с именем `Name`.
- `<Name ... />` — **переопределяет *унаследованный* член** с именем `Name`. Тег *и есть* имя члена; вы **не** повторяете `type`, а лишь заново указываете атрибуты, которые хотите изменить.

```
<SCType ID="ZCleanup" type="TZCleanup" Caption="Cleanup height">
    <!-- переопределяем значения по умолчанию унаследованных членов: -->
    <Enabled DefaultValue="True"/>
    <Height  DefaultValue="2"/>
</SCType>
```

(реальный пример: `Operations/MillOperations/FaceMillingOp.xml`)

Переопределения могут затрагивать вложенные члены и даже варианты перечислений. Шаблон `<EnumItem Visible="True"/>` снова включает вариант, скрытый базовым типом:

```
<CirclesDivision DefaultValue="Halves">
    <AbsQuadrants Visible="True"/>   <!-- показать вариант, скрытый базой -->
    <AbsHalves    Visible="True"/>
</CirclesDivision>
```

(реальный пример: Machines/AbstractMachine.xml)

4.3 Значения по умолчанию: метрика и дюймы

Числовые свойства могут объявлять два значения по умолчанию: `DefaultValue` для метрической системы и `InchDefaultValue` для дюймовой. Система выбирает нужное в соответствии с активной системой единиц — это *независимые значения*, а не автоматический пересчёт единиц, поэтому задавайте каждое разумным круглым числом.

```
<SCType ID="Outer" type="Double" DefaultValue="0.02" InchDefaultValue="0.001"
        DimensionKind="Linear"/>
```

(реальный пример: Operations/AbstractOP.xml)

5 Массивы

`Array` (или предопределённый `DynamicArray`) хранит переменное число элементов одного типа. Шаблон элемента объявляется как единственный потомок массива:

```
<SCType ID="TStringList" Caption="List of strings" type="Array">
    <SCType ID="L" Caption="Line" type="String" DefaultValue=""/>
</SCType>
```

Для массивов, элементам которых нужна устойчивая идентичность (чтобы приложение могло сопоставлять элементы между правками), задайте в `CollectionKeyField` имя члена, хранящего ключ:

```
<SCType ID="List" type="Array" CollectionKeyField="Name">
    <SCType ID="Script" type="TScript"/>
</SCType>
```

(реальный пример: Operations/AbstractOP.xml)

Внутри элемента массива текущий индекс доступен выражениям через `Attribute(ItemIndex)`, что удобно для автонумеруемых подписей (см. язык выражений):

```
<SCType ID="J23ValuePair" Caption="Point [Attribute(ItemIndex)]" type="TJ23ValuePair"/>
```

6 SCProperty — глобальные именованные свойства

Если SCType объявляет *шаблоны*, то SCProperty объявляет **именованное глобальное свойство**: одиночное адресуемое значение или объект на уровне пространства имён. Они работают как общие константы/объекты, на которые выражения в других местах могут ссылаться по имени.

```
<SCProperty ID="MetricMeasurementsSystem" type="TMeasurementsSystem">
    <LinearUnits DefaultValue="Millimeters"/>
    <CuttingSpeedUnits DefaultValue="MetersPerMinute"/>
</SCProperty>

<!-- вычисляемая глобальная строка, берущая значение из другого глобального свойства -->
<SCProperty ID="LinearUnits$" type="String"
            DefaultValue="
[CurrentMeasurementsSystem.LinearUnits.EnumValue.Attribute(Caption)]"/>
```

(реальный пример: CommonTypes.xml)

> **Соглашение об именовании с $.** Завершающий $ в ID свойства (например, LinearUnits$) — это *соглашение*, помечающее производное строковое свойство для отображения. $ — просто часть имени; для парсера он не имеет особого значения.

7 Файловые переменные

Ссылки на файлы — как цели <SCInclude>, так и атрибуты вроде ImageFile, Icon, SPPFile — используют подстановки $(VARIABLE) вместо абсолютных путей. Они разрешаются во время выполнения относительно установки и профиля пользователя.

Переменная	Во что разрешается (обычно)
$(SUPPLEMENT_FOLDER)	<корень установки>\Supplement
$(MACHINES_FOLDER)	пользовательская …\Machines
$(SCHEMAS_FOLDER)	$(MACHINES_FOLDER)\Schemas
$(OPERATIONS_FOLDER)	пользовательская …\Operations
$(LOCAL_OPERATIONS_FOLDER)	локальная папка операций пользователя

Переменная	Во что разрешается (обычно)
`$(COMMON_CONTAINERS_FOLDER)`	общая папка контейнеров/расширений
`$(PROGRAM_PERSONAL)`	пользовательские данные программы

Всегда используйте эти переменные вместо жёстко прописанных путей, чтобы ваши файлы продолжали работать независимо от того, куда установлен продукт и какой профиль пользователя активен.

8 Каталог атрибутов

Атрибуты, заданные на `SCType` / `SCProperty`, делятся на две группы. Небольшой набор **основных атрибутов** понимает сам загрузчик дескрипторов; всё остальное — **потребительские атрибуты**, которые переносятся вместе с дескриптором и интерпретируются интерфейсом, движком или конкретным презентером. С точки зрения автора и те, и другие — «просто атрибуты»; таблица ниже поясняет назначение каждого.

Логические атрибуты принимают буквальный текст `True` / `False` (регистр не важен).

8.1 Идентичность и тип

Атрибут	Значение	Смысл
`ID`	имя	Имя члена/типа. Обязательно в объявлениях.
`type` / `Type`	имя типа	Тип значения — встроенный или ранее объявленный тип.
`Caption`	строка / выражение	Человекочитаемая подпись, показываемая в интерфейсе. Может вычисляться.
`Version`	целое	Версия схемы типа; увеличивайте её при изменении структуры, чтобы старые сохранённые данные мигрировали.
`Obsolete`	лог.	Помечает член как устаревший; сохраняется для совместимости, обычно скрыт.

8.2 Значения и единицы

Атрибут	Значение	Смысл
`DefaultValue`	литерал / выражение	Значение по умолчанию (метрика). Для перечислений — `ID` варианта по умолчанию.
`InchDefaultValue`	литерал / выражение	Значение по умолчанию при активной дюймовой системе единиц.
`DimensionKind`	см. ниже	Физическая размерность числового значения, чтобы интерфейс показывал единицы и корректно пересчитывал.
`UnitsChar`	строка / выражение	Явная подпись единицы для отображения (например, `ml/s` или `[TimeUnits_Sec$]`).

Значения `DimensionKind`: `None`, `Linear`, `InverseLinear`, `Angular`, `Feedrate`, `CuttingSpeed`, `Revolution`. Используйте `None` для безразмерных чисел и компонентов направления; `Linear` для расстояний; `Angular` для углов.

8.3 Состояние и поведение

Атрибут	Значение	Смысл
`Enabled`	лог. / выражение	Активно/редактируемо ли свойство. `False` делает его серым (недоступным).
`Visible`	лог. / выражение	Показывается ли строка. Управляйте этим из других свойств для динамических форм.
`ReadOnly`	лог.	Значение показывается, но не редактируется.
`EnabledValue`	имя члена	Для логического члена-«переключателя»: указывает соседний член, который он включает при значении true (используется с `Compact`).
`IsStructural`	лог.	Изменение этого значения меняет структуру объекта (вызывает перестроение, а не простое обновление значения).
`IsDynamic`	лог.	Подпись/иконка члена вычисляется для каждого экземпляра (например, элементы массива, которые сами себя именуют).

8.4 Отображение и группировка

Атрибут	Значение	Смысл
`Category`	id категории	Назначает свойство в категорию/вкладку инспектора (например, `TPropertiesCategoryList.Feeds`).
`Parent`	имя члена	Переподчиняет эту строку другому члену в дереве инспектора, независимо от порядка объявления.
`Priority` / `Order`	целое	Порядок сортировки строки внутри её группы/категории в инспекторе (бóльший `Priority` всплывает выше). Для размещения в меню *новой операции* используйте `MultiGroup` + `OrderInGroup` (см. Дескрипторы операций).
`Expanded`	лог.	Развёрнута ли составная строка изначально в дереве.
`Compact`	лог.	Сворачивает составной тип в одну строку инспектора (см. §9).
`Transparent`	лог.	Сам контейнер не показывается; видны только его потомки (чисто группирующая обёртка).
`Text`	строка / выражение	Сводный текст, показываемый для свёрнутой составной строки.
`ImageFile`	путь	Иконка строки (используйте файловую переменную `$(…)`).

8.5 Редакторы / презентеры

Эти атрибуты подключают к свойству специализированный редактор или отрисовщик. Значение — зарегистрированное имя презентера, предоставляемого приложением; вы ссылаетесь на него, но не определяете его.

Атрибут	Смысл
`EnumPresenter`	Пользовательский выпадающий список/селектор для строки (выбор инструмента, выбор магазина, ...).
`BtnPresenter`	Отображает строку как кнопку действия (например, «Нажмите, чтобы вычислить»).
`VisPresenter` / `CaptionPresenter`	Пользовательский отрисовщик значения/подписи.

Атрибут	Смысл
`IsRadioEdit` / `RadioEditValue`	Объединяет перечисление с соседними членами-значениями в одну строку (см. §9).
`Filter` / `CustomDialogID`	Для свойств с именем файла: фильтр файлового диалога / пользовательский диалог.

8.6 Массивы и интеграция

Атрибут	Смысл
`CollectionKeyField`	Член, однозначно идентифицирующий элемент массива.
`CollectionValueField`	Член, хранящий полезное значение элемента.
`Optional`	На `<SCInclude>`: загружать только если файл существует.
`ClassName` / `ClassGUID` / `InterfaceName` / `InterfaceGUID`	Привязывают свойство к COM-объекту/интерфейсу, который приложение для него создаёт. Продвинутое; редко нужно в рукописных дескрипторах.

> **Неизвестные атрибуты сохраняются, а не отвергаются.** Если вы зададите атрибут, который загрузчик не распознаёт, он сохраняется на дескрипторе и доступен потребителям и API. Это сделано намеренно — именно так презентеры получают свои параметры (например, `AxisIdx`, `HolderType`). Это также означает, что опечатка в имени *известного* атрибута проходит молча, поэтому перепроверяйте написание и регистр.

9 Строки Compact и radio-edit

В реальных дескрипторах постоянно встречаются два приёма отображения; их стоит понять, потому что они меняют то, как несколько членов отображаются в одной строке инспектора.

Compact

`Compact="True"` на составном типе сворачивает его в **одну** строку инспектора: первый видимый потомок поднимается рядом с подписью родителя. Классическая форма — флаг `Enabled` плюс значение, которое он включает, связанные через `EnabledValue`:

```
<SCType ID="SmoothCorners" type="ComplexType" Compact="True">
    <SCType ID="Enabled" type="Boolean" DefaultValue="True" EnabledValue="Radius"/>
    <SCType ID="Radius"  type="TPercentageValue" Visible="False">
```

```
        <PercentValue DefaultValue="25"/>
    </SCType>
</SCType>
```

(реальный пример: Operations/MillOperations/FaceMillingOp.xml)

Пользователь видит одну строку «Smooth corners» с флажком и значением.

Radio edit

`IsRadioEdit="True"` на перечислении объединяет его с соседними членами-значениями в одну строку, где выбор перечисления определяет, какой сосед активен. Каждый вариант перечисления указывает на своего соседа через `RadioEditValue`:

```
<SCType ID="ReferenceType" type="Enumerated" IsRadioEdit="True">
    <SCType ID="Angle"  Caption="(degrees)"     type="none" RadioEditValue="Angle"/>
    <SCType ID="Linear" Caption="[LinearUnits$]" type="none" RadioEditValue="Linear"/>
</SCType>
<SCType ID="Angle"  type="Double" DefaultValue="5"  Visible="False"
DimensionKind="Angular"/>
<SCType ID="Linear" type="Double" DefaultValue="50" Visible="False" DimensionKind="Linear"/>
```

(реальный пример: Machines/AbstractMachine.xml)

Далее: **Язык выражений**

Язык выражений

Многие значения атрибутов — не константы, а **выражения**, вычисляемые относительно других свойств. Именно это делает дескрипторы динамическими: значение по умолчанию, зависящее от другого поля; строка, появляющаяся только при определённых условиях; подпись, отражающая текущую систему единиц.

1 Где используются выражения

Любой из этих атрибутов может содержать выражение вместо литерала:

- `DefaultValue`, `InchDefaultValue` — вычисляемые значения по умолчанию.
- `Visible`, `Enabled`, `ReadOnly` — динамическое состояние (выражение должно давать логическое значение).
- `Caption`, `Text`, `UnitsChar` — вычисляемые строки для отображения.
- ещё несколько, используемых отдельными подсистемами (например, флаги состояния ошибки).

Значение атрибута считается выражением, если содержит ссылку на свойство в скобках `[...]` либо распознаваемую функцию/оператор. Простое значение вроде `DefaultValue="1"` или `DefaultValue="Millimeters"` берётся буквально.

2 Ссылки на свойства: `[...]`

Ссылка в квадратных скобках разрешается в **значение** другого свойства.

```
Visible="[Enabled]"
DefaultValue="[ApproachFeed.ValuePerMinut]"
```

Разрешение имён

Голое имя вроде `[Enabled]` ищется в следующем порядке:

1. **соседи** текущего свойства,
2. цепочка **родителей**,
3. **глобальные** свойства / объекты контекста в области видимости.

Поэтому `[Enabled]` рядом со свойством обычно означает «соседний `Enabled`».

Точечные пути

Точки позволяют заходить в составные типы и перемещаться по дереву:

```
DefaultValue="[ApproachFeed.LatheFeedRelativeMeasurement]"
Visible="[TechOperation.IsPrevOpInOtherChannel]"
```

Путь также может адресовать **вариант перечисления** по имени — `[Group.LinearAxis]` это вариант `LinearAxis` перечисления `Group`. Сравнение свойства с одним из его вариантов — идиоматический способ проверить перечисление:

```
Visible="[SimulationType] = [SimulationType.Painting]"
DefaultValue="([Group] != [Group.LinearAxis]) AND (([Max]-[Min]) GT 10000)"
```

Корни контекста

Помимо ближайших свойств, в качестве корней пути доступны некоторые **объекты контекста**, дающие выражениям доступ к более широкому окружению. Те, что встречаются чаще всего:

Корень	Даёт доступ к
`TechOperation`	Операция, которой принадлежит свойство (флаги, соседи, информация о магазине, ...).
`CAMApplication`	Состояние уровня приложения (например, `[CAMApplication.PLMManager.ConnectionCount]`).
`CurrentMeasurementsSystem`	Активная система единиц и её подписи.

3 Аксессоры внутри `[...]`

Внутри ссылки можно вызывать аксессоры на разрешённом свойстве:

Аксессор	Возвращает
`.Attribute(Name)`	Значение атрибута дескриптора `Name` (например, `Caption`, `ItemIndex` или любой пользовательский атрибут).
`.EnumValue`	Текущий выбранный вариант перечисления (можно дополнить `.Attribute(Caption)`, чтобы получить его подпись).
`.Value`	Значение свойства явно (обычно подразумевается).
`.Count`	Количество элементов массива.

```
<!-- подпись текущего выбранного варианта LinearUnits -->
DefaultValue="[CurrentMeasurementsSystem.LinearUnits.EnumValue.Attribute(Caption)]"

<!-- текущий индекс этого элемента массива, для самонумерующейся подписи -->
Caption="Point [Attribute(ItemIndex)]"
```

Внутри шаблона элемента массива `$ParentNode` ссылается на охватывающий элемент коллекции, например `Caption="Rule [$ParentNode.Attribute(ItemIndex)]"`.

4 Операторы

Группа	Операторы
Арифметические	`+ - * / ^` (степень) `DIV` (целочисленное деление) `MOD`
Сравнения	`= != <> < > <= >=` и словесные формы `LT GT LE GE`
Логические	`AND OR NOT`

Словесные формы существуют потому, что значения XML-атрибутов заключены в кавычки, а символьные `</>` неудобно читать рядом с разметкой — `GT`/`LT`/`GE`/`LE` обычно и используются в реальных файлах. Скобки группируют подвыражения как обычно.

```
Visible="[Enabled] and ([StepCount] GT 1)"
DefaultValue="[UpperWorkLevel] + 5"
SupportShortestPathRotation DefaultValue="([Group] != [Group.LinearAxis]) AND (([Max]-[Min]) GT 10000)"
```

5 Функции

Условные — `CASE`, `IF`, `SWITCH`

Эти три покрывают почти всю реальную логику.

`IF(условие, значениеЕслиИстина, значениеЕслиЛожь)` — простой тернарный оператор.

```
DefaultValue="IF([SimulateToolChange], 'Auto', 'DoNotUse')"
```

`CASE(...)` — самый частый условный оператор в дескрипторах. В форме с двумя значениями ведёт себя как `IF`: `CASE(условие, значениеЕслиИстина, значениеЕслиЛожь)`. Он также допускает цепочку: ряд пар `условие, результат`, вычисляемых по порядку, с необязательным финальным значением по умолчанию.

```
<!-- форма if/else -->
DefaultValue="CASE([IsSpatial], 'SnapToWorkpieceCS', 'SnapToToolCS')"

<!-- цепочка: побеждает первое подходящее условие, последнее «голое» значение — по умолчанию
-->
DefaultValue="CASE(
    [HolderType]=[HolderType.LeftLatheSpindle],  '1',
    [HolderType]=[HolderType.RightLatheSpindle], '2',
    -1)"
```

(реальный пример: Machines/MachineTypes.xml)

`SWITCH(значение, вариант1, результат1, вариант2, результат2, …, по_умолчанию)` — выбирает по **равенству** с `значением`, как классический оператор switch. Удобно, когда одно свойство отображается во множество исходов:

```
RotationsSequence DefaultValue="
    SWITCH([ControlData.ToolFrameOutput.Format],
        [ControlData.ToolFrameOutput.Format.EulerXYZ], 'XYZ',
        [ControlData.ToolFrameOutput.Format.EulerXZY], 'XZY',
        'XYZ')"          <!-- финальное значение — по умолчанию -->
```

(реальный пример: Machines/AbstractMachine.xml)

Числовые функции

Функция	Смысл
`MIN(a, b, …)` / `MAX(a, b, …)`	Минимум / максимум
`ABS(x)`	Абсолютное значение
`ROUND(x [, digits])` / `SMARTROUND(x)`	Округление
`SQRT`, `EXP`, `LN`, `SGN`/`SIGN`	Базовая математика
`SIN`, `COS`, `TAN`, `ASIN`, `ACOS`, `ARCTAN`, …	Тригонометрия

Строковые функции

Функция	Смысл
`STR(x)`	Число → строка

Функция	Смысл
`CHR(code)`	Символ по коду
`CONCATENATE(sep, s1, s2, …)`	Объединить строки через разделитель
`ISVALUEINSET(v, a, b, …)`	Истина, если `v` равно одному из перечисленных значений

6 Строковые литералы

Строковые литералы внутри выражений записываются в одинарных кавычках: `'XYZ'`, `'Auto'`.

```
DefaultValue="IF([SimulateToolChange], 'Auto', 'DoNotUse')"
```

7 Разобранные примеры

Значение, по умолчанию равное другому свойству — держать две подачи синхронными, пока их не изменили:

```
<ValuePerMinut DefaultValue="[ApproachFeed.ValuePerMinut]"
               InchDefaultValue="[ApproachFeed.ValuePerMinut]"/>
```

Строка, появляющаяся только когда установлен флаг и количество имеет смысл:

```
<SCType ID="StepValue" type="Double" DefaultValue="1" DimensionKind="Linear"
        Visible="[Enabled] and ([StepCount] GT 1)"/>
```

Значение перечисления по умолчанию, выбранное из состояния другого перечисления:

```
<SCType ID="DefaultState" type="Enumerated"
        DefaultValue="if([RotaryTrans.TCPMAvailable] AND (NOT [RotaryTrans.TCPMDefault]),
'Auto', 'Off')">
    <SCType ID="Off"  type="none"/>
    <SCType ID="Auto" type="none"/>
</SCType>
```

(реальный пример: Machines/AbstractMachine.xml)

> **Советы.** Держите выражения свободными от побочных эффектов — они вычисляются при каждом обновлении интерфейса. Проверяйте сравнения перечислений через форму `[Enum.Option]`, а не строковые литералы, чтобы переименованный вариант падал явно, а не

> молча не совпадал. Когда выражение `Visible`/`Enabled` ссылается на свойство в другой ветви дерева, используйте полный точечный путь от известного корня.

Далее: **Дескрипторы операций**

Использование XML-свойств из кода

Свойства, которые вы объявляете в XML, — это те же свойства, что вы читаете и записываете из кода. Они доступны из нескольких окружений — **CAM API** (.NET, C++, …), **нативных пакетов Delphi / C++**, **.NET-постпроцессоров** и **постпроцессоров** `.sppx` — и, что важно, **шаблон доступа во всех них одинаков**. Эта глава один раз описывает общий шаблон, а затем показывает небольшие особенности каждого окружения.

1 Одна модель, один шаблон доступа, много окружений

Вспомните различие типы и экземпляры. Из любого окружения вы работаете с **экземплярами**: живым деревом свойств конкретного объекта, где каждый узел — это **указатель свойства**. Через указатель вы читаете или записываете его значение, заходите в потомков, ищете и изучаете его **дескриптор** для метаданных типа.

Указатель всегда предоставляет **одни и те же именованные аксессоры**, в каком бы окружении вы ни были:

Аксессор	Что делает
`Str[name]` / `Int[name]` / `Flt[name]` / `Bol[name]`	читают **и записывают** значение потомка как строку / целое / double / логическое
`Ptr[name]`	получить указатель на потомка (чтобы зайти в `ComplexType`)
`Arr[name]`	получить дочерний массив
`ValueAsString` / `ValueAsInteger` / `ValueAsDouble` / `ValueAsBoolean`	*собственное* значение указателя
`FindChild` / `FindProperty`	найти потомка по имени

Имена могут быть составными, и регистр не важен. Вместо пошагового спуска через `Ptr` передавайте **точечный путь** прямо в аксессор, а к **элементам массива по ключу или индексу** обращайтесь через `(…)`:

```
Flt["RoughPasses.StepValue"]          # вложенное значение, через точку
Int["controldata.usearc"]             # регистронезависимо
Flt["Axes(AxisXPos).Value"]           # элемент массива по ключу, затем потомок
Str["Project.Operations(0).Name"]     # элемент массива по индексу
```

Одни и те же имена аксессоров, тот же точечный/регистронезависимый поиск и тот же синтаксис массивов `(key)` присутствуют в **каждом** окружении; различается только языковая обёртка:

Окружение	Тип указателя	COM-обёртка?
CAM API (.NET / C++ / ...)	`IST_XMLPropPointer`	да
Нативные пакеты Delphi / C++	`IST_XMLPropPointer`	нет
.NET-постпроцессоры	`INamedProperty`	нет
Постпроцессоры `.sppx` (Pascal-подобный язык)	встроенный	нет

Поэтому **§2–9 ниже — это вариант .NET CAM API**: встречающиеся там `ComWrapper`, `Invoke` / `InvokeAndWrap` и `using` — это **обёртка .NET-COM**, *а не* часть модели свойств. Нативное окружение и постпроцессоры (§10–12) используют те же имена аксессоров без этой церемонии.

Как узнать полное имя свойства

Чтобы пользоваться аксессорами, нужно **полное точечное имя** свойства. Быстрее всего получить его из работающей программы:

1. Откройте **инспектор свойств** операции и найдите свойство.
2. У **правого края** его строки нажмите кнопку **⋯ (три точки)**, чтобы открыть меню строки, и выберите **«Скопировать имя свойства в буфер обмена»**.

В том же меню есть **«Настроить...»**, открывающее окно, где показано не только **вычисленное значение** свойства, но и его **исходное выражение** — удобно, когда значение параметризовано от других свойств (см. язык выражений). *(Это окно есть только в недавних версиях.)*

> **Старые версии — обходной путь.** Если окна *«Настроить...»* нет, воспользуйтесь контекстным меню операции → **«Сохранить как пользовательскую операцию»**. В открывшемся окне найдите свойство в инспекторе, щёлкните по нему правой кнопкой и используйте там **«Скопировать имя свойства в буфер обмена»**. Затем **уберите ведущий `RefOperations(0).`** из скопированного имени — он появляется лишь потому, что это окно показывает *копию* свойств внутри отдельной сущности *UserOperation*, а не живую операцию. Останется имя, которое вы передаёте аксессорам.

2 Как получить дерево свойств объекта

Объекты предметной области предоставляют своё дерево свойств через член `XMLProp`. Для технологической операции:

```
using var xmlPropsCom = operationCom.InvokeAndWrap(operation => operation.XMLProp);
```

> `XMLProp` **— это единое дерево свойств объекта.** Для свойств, хранящихся **только** в `XMLProp` (современный случай), оно авторитетно — для чтения/записи не нужен отдельный шаг фиксации. Но некоторые **операции** до сих пор дублируют отдельные значения в собственных полях объекта (унаследованный код); там `XMLProp` и объект могут разойтись и должны синхронизироваться через `SaveToXMLProp` / `LoadFromXMLProp` — см. §7.

Тот же шаблон с `XMLProp` применим к станкам, инструментам и другим объектам.

3 Чтение и запись значений по имени

Каждый узел предоставляет типизированные индексаторы по имени потомка: `Str`, `Flt`, `Int`, `Bol`. Это идиома, используемая во всех примерах — значение задаётся присваиванием индексатору:

```
// из FullWorkflow3DProject/TechnologyHelper.cs
using var xmlPropsCom = operationCom.InvokeAndWrap(operation => operation.XMLProp);
xmlPropsCom.Invoke(xmlProps =>
{
    xmlProps.Str["DrillingType"] = "HolePocketing";   // записать строку / id
варианта перечисления
});
```

Чтение симметрично:

```
xmlPropsCom.Invoke(xmlProps =>
{
    double step  = xmlProps.Flt["StepValue"];
    int    count = xmlProps.Int["StepCount"];
    bool   on    = xmlProps.Bol["Enabled"];
    string mode  = xmlProps.Str["DrillingType"];
});
```

Как и в любом окружении (§1), имя может быть **точечным путём** и **регистронезависимо**, так что можно добраться до вложенного значения за один вызов, а не спускаться через `Ptr` (§4):

```
double step = xmlProps.Flt["RoughPasses.StepValue"];
bool   on   = xmlProps.Bol["roughpasses.enabled"];   // регистр игнорируется
```

Сначала проверьте существование

Свойство может отсутствовать на конкретном объекте (например, необязательный или пользовательский параметр). Защищайте чтение через `PropExists`:

```
var value = xmlProps.PropExists["DrillingType"]
    ? xmlProps.Str["DrillingType"]
    : "";
```

> **Перечисления — по ID или по индексу.** Значение перечисления — это *`ID` варианта*, поэтому `Str["DrillingType"] = "ChipRemoving"` задаёт его по ID (`ID` варианта, а не его подпись — см. §3.2). Можно **также** читать или записывать его **по нулевому индексу варианта** через `Int`: `Int["DrillingType"] = 2` выбирает *третий* вариант, а чтение `Int["DrillingType"]` возвращает индекс текущего варианта. (Это работает в каждом окружении.)

Удобный слой-обёртка (.NET)

Пакет-хелпер для .NET оборачивает эти индексаторы в методы-расширения над `ComWrapper<IST_XMLPropPointer>`, что позволяет обойтись без явного `Invoke`:

```
double step = xmlPropsCom.Flt("StepValue");      // геттер
xmlPropsCom.SetFlt("StepValue", 2.5);            // сеттер
xmlPropsCom.SetStr("DrillingType", "ChipRemoving");
string calc = xmlPropsCom.CStr("SomeComputed");  // вычисленное значение
вычисляемого свойства
```

Хелперы есть для каждого типа (`Str`/`Flt`/`Int`/`Bol` и их формы `Set…`), плюс `Ptr`, `Arr`, `FindProperty`, `FindPropertyInWholeScope` и другие. Используйте любой стиль, подходящий вашему коду; все они вызывают один и тот же низлежащий интерфейс.

4 Навигация по дереву

Заход в составные типы

`Ptr[name]` (или хелпер `Ptr(name)`) спускается в потомка-`ComplexType`; дальше продолжайте типизированными индексаторами. Это отражает точечные пути из вашего XML (`RoughPasses.StepValue`):

```
xmlPropsCom.Invoke(xmlProps =>
{
    var rough = xmlProps.Ptr["RoughPasses"];
    rough.Bol["Enabled"]   = true;
```

```
    rough.Flt["StepValue"] = 1.0;
});
```

Поиск по имени

Когда не хочется выписывать полный путь, ищите по поддереву:

```
using var dt = xmlPropsCom.FindPropertyInWholeScope("DrillingType");
string current = dt.ValueAsString();
dt.SetNodeValue("ChipRemoving");
```

`FindPropertyInWholeScope` возвращает первое совпадение на любой глубине — предпочитайте цепочки `Ptr` (или ограниченный областью `FindProperty`), когда имя может быть неуникальным.

5 Массивы

Дочерний массив — это `IST_XMLPropArray`, получаемый через `Arr[name]`. Он поддерживает создание и добавление элементов поверх обычных операций с указателем. Вот реальный шаблон добавления пользовательского элемента массива:

```
// из ExtensionOperationParamsNet/OperationCustomPropsHelper.cs
const string arrayId = "CustomOperationPropertiesArray";
using var paramArrayCom = ComWrapper.Create(operation.XMLProp.Arr[arrayId]);

if (!operation.XMLProp.PropExists[fullId])
{
    var newItem = paramArrayCom.Invoke(a => a.CreateNewItem("CustomOperationProperty"));
    newItem.Str["Name"] = paramId;
    paramArrayCom.Invoke(a => a.AddItem(newItem));
}
```

Адресация элемента массива по ключу

Когда у элемента массива есть член `Name`/ключ, можно обратиться к нему напрямую синтаксисом пути `ArrayName(key)` в индексаторе — без ручного перебора:

```
var fullId = $"{arrayId}({paramId})";          // например,
CustomOperationPropertiesArray(MyParam)
operation.XMLProp.Str[fullId] = "true";        // чтение/запись значения элемента
bool exists = operation.XMLProp.PropExists[fullId];
```

6 Собственное значение указателя

Аксессоры из §3 читают **потомка** по имени (`Str["StepValue"]`). Когда у вас уже есть указатель на само свойство (а не на его родителя), читайте или записывайте **его собственное значение**. Это тоже **типизировано** — `ValueAsString`, `ValueAsDouble`, `ValueAsInteger`, `ValueAsBoolean` (см. список аксессоров в §1). `ValueAsString` применим к **любому** типу свойства и возвращает его строковое представление:

```
double v   = somePropCom.ValueAsDouble();   // типизированное собственное значение
string raw = somePropCom.ValueAsString();   // любой тип → его текстовая форма
somePropCom.SetNodeValue("75");             // задать собственное значение указателя
```

7 Операции: синхронизация полей объекта с `XMLProp`

(Только для операций.) Для операции `XMLProp` — это её **единое дерево свойств** с двумя задачами: единое дерево времени выполнения (чтобы общий код мог показать любую операцию в инспекторе и значения можно было копировать между операциями) и форма, в которой свойства операции сериализуются в/из XML при сохранении или загрузке проекта.

Исторически операции хранили свои значения в обычных **полях объекта**, а `XMLProp` был добавлен позже сверху. Поэтому **унаследованные** свойства живут в *обоих* местах — в полях объекта **и** в `XMLProp` — и обе копии нужно держать согласованными:

- **`SaveToXMLProp`** — заталкивает текущие **значения полей операции в `XMLProp`**. Вызывайте перед чтением `XMLProp` (например, перед заполнением инспектора из дерева), чтобы дерево отражало актуальное состояние объекта.
- **`LoadFromXMLProp`** — читает значения **из `XMLProp` обратно в поля объекта**. Вызывайте после правки `XMLProp` (например, после того как пользователь изменил значение в инспекторе), чтобы модель объекта подхватила изменение.

Современные свойства живут только в `XMLProp` — дублирующего поля нет, поэтому синхронизация не нужна; достаточно читать/писать `XMLProp`. Вызовы синхронизации важны в основном для старых классов, которых всё ещё много.

`LoadFromXMLProp` также принимает `XMLProp` *другой* операции — именно так настройки одной операции копируются из другой:

```
operationCom.LoadFromXmlProp(otherXmlPropsCom);   // применить чужое дерево свойств к
этой операции
```

`SaveToXmlProp()` возвращает свойства операции в виде дерева (с уже сброшенными в него полями объекта) — удобно для снимка, сравнения или переноса.

8 Изучение дескрипторов и значений по умолчанию

Каждый указатель свойства связан со своим **дескриптором** (`IST_XMLPropDescriptor`), который несёт объявленные вами в XML метаданные — имя, подпись, простой тип, значение по умолчанию, пользовательские атрибуты, наследование. Используйте его, когда нужно рассуждать о *схеме*, а не о значении (например, обобщённо перечислить параметры объекта или прочитать пользовательский атрибут, заданный на дескрипторе).

Значения по умолчанию доступны через `IST_PropDefaultsEditor` (запрашивается у дескриптора): он различает **системное значение по умолчанию** (объявленное в XML дескриптора) и любое **пользовательское** переопределение, для метрической и дюймовой систем единиц, и позволяет читать, задавать или сбрасывать пользовательское значение по умолчанию.

9 Практические заметки

- **Освобождайте COM-обёртки.** Указатели свойств и массивы — это COM-объекты; оборачивайте их в `using`, чтобы они освобождались своевременно. В примерах работа выполняется внутри лямбд `Invoke` / `InvokeAndWrap`, которые управляют временем жизни за вас.
- **Используйте точный `ID`** из дескриптора — имена должны совпадать с объявленными в XML.
- **Задавайте перечисления и логические значения их каноническим текстом** — `ID` варианта для перечислений; для логических значений в примерах используются строки `"true"`/`"false"` через `Str[...]` либо типизированный индексатор `Bol[...]`.
- **Не боритесь с вычисляемыми значениями.** Свойство, значение которого определяется выражением `DefaultValue`/`Visible`, будет пересчитываться; читайте его вычисленный результат через `CStr` (или `ValueAsCalculatedString`), а если нужно хранить фиксированное значение, сделайте его реальным редактируемым полем.
- **Куда смотреть дальше.** Репозиторий `cam-api-examples` (`Operation/ExtensionOperationParamsNet`, `FullWorkflow`, `ProjectMachine`) содержит полные компилируемые примеры использования всего вышеописанного.

10 Нативное использование (Delphi / C++)

Нативные пакеты предоставляют тот же указатель свойства через публичный `IST_XMLPropPointer`. **Нет COM-обёртки, нет `Invoke`, нет `using`**: вы держите обычную ссылку на интерфейс и вызываете те же аксессоры напрямую. Методы объекта предметной области `LoadFromXMLProp` / `SaveToXMLProp` (и `LoadConditionsFromXMLProp`) получают такой указатель:

```
procedure LoadFromXMLProp(XMLProp: IST_XMLPropPointer);
var p: IST_XMLPropPointer;
begin
  if XMLProp <> nil then with XMLProp do begin
    p := FindProp('SafeLevel');
    if p <> nil then with p do begin
```

```
      fZRetract.IsAbs  := Int['ReferenceType'] = 0;   // те же аксессоры Int[…]/Flt[…]
      fZRetract.IncVal := Flt['RelValue'];
      fZRetract.AbsVal := Flt['AbsValue'];
    end;
  end;
end;
```

Это те же `Int[…]` / `Flt[…]` / `Str[…]` / `Bol[…]` / `FindProp`, что и из .NET — просто без COM-церемонии.

11 В .NET-постпроцессорах (`INamedProperty`)

Постпроцессор на .NET читает именованные свойства команды/проекта, которые ему передали, и выдаёт УП. Он использует **чисто-.NET-слой без COM — `INamedProperty`**, но доступ тот же: индексаторы `Str` / `Int` / `Flt` / `Bol` / `Ptr` / `Arr` плюс `ValueAsString` / `ValueAsInteger` / `ValueAsDouble` / `ValueAsBoolean`, с привычными **точечными, регистронезависимыми** именами и адресацией массивов `(key)` / `(index)`:

```
double x      = cmd.Flt["Axes(AxisXPos).Value"];
int    origin = cmd.Int["OriginType"];
double ox     = cmd.Flt["WCS.OriginPoint.X"];
```

Разобранный постпроцессор есть в репозитории `postprocessors-examples` в `Features/NamedParameters/DemoOfWritePPFun` (см. `WritePPFun.cs`). Тот пример — слегка необычный контекст: он *записывает* параметры в файл — но доступ к свойствам ровно такой, как выше.

12 В постпроцессорах `.sppx` (Pascal-подобный язык)

Постпроцессоры `.sppx` пишутся на специальном Pascal-подобном языке. Набор доступных типов отличается, но **имена методов и способ доступа к свойствам идентичны**. В фрагментах ниже `Cmd` — один из таких объектов со свойствами — текущая **команда CLData** — и **те же аксессоры применимы к любому объекту свойств**: `.Int["…"]` / `.Flt["…"]` / `.Str["…"]` / `.Bol["…"]` для значений и `.Ptr["…"]` для указателя на дочернее свойство, все с привычными точечными / `(key)` именами:

```
if Cmd.Ptr["Axes(AxisXPos)"] <> 0 then          // указатель на дочернее свойство (здесь:
существует ли ось?)
  X = Cmd.Flt["Axes(AxisXPos).Value"]
if Cmd.Int["Axes(AxisAPos).BrakeState"] > 0 then ...
CurG68_2_X = Round(Cmd.Flt["WCS.OriginPoint.X"], 4)
```

(реальный пример: `MyDocuments\Postprocessors\Mill\Fanuc (30i)_Mill.sppx`)

Далее: **Пользовательские значения по умолчанию** · Назад к **оглавлению**.

Пользовательские значения по умолчанию — переопределение поставляемых

Каждый дескриптор свойства поставляется со **значением по умолчанию** — значением, с которого начинает новый объект; оно объявлено как `DefaultValue` (и `InchDefaultValue`) в XML (§8). Эти поставляемые значения лежат в `Supplement`, который вы **не** редактируете. Вместо этого фреймворк позволяет задать **собственное** значение по умолчанию для любого свойства — *пользовательское значение по умолчанию* — которое имеет приоритет над поставляемым. Пользовательские значения хранятся в небольших файлах `.usrdef`, загружаются при старте и применяются без перекомпиляции и без редактирования `Supplement`.

Это штатный способ ответить на запрос «я всегда хочу, чтобы этот параметр начинал с *моего* значения, а не заводского» — для одного рабочего места или с развёртыванием на многие.

Системное значение vs. пользовательское

За значением по умолчанию каждого свойства стоят два слоя:

- **системное значение по умолчанию** — значение, поставляемое в `DefaultValue` / `InchDefaultValue` дескриптора. Оно только для чтения и этим механизмом никогда не меняется; к нему всегда можно вернуться.
- **пользовательское значение по умолчанию** — ваше переопределение. Будучи заданным, оно становится *действующим* значением по умолчанию свойства: новые объекты начинают с него, и «сброс к значению по умолчанию» возвращает к нему.

Поскольку они хранятся отдельно, вы всегда можете **сбросить к системному значению**, чтобы вернуть заводское, или вовсе **удалить** своё пользовательское значение.

Задание пользовательского значения из инспектора

Пользовательскими значениями вы управляете прямо из инспектора параметров. Откройте контекстное меню строки свойства (то же меню строки, через которое копируется полное имя свойства — см. Использование XML-свойств из кода); оно предлагает:

Команда	Действие
Сохранить как пользовательское значение по умолчанию	Сделать текущее значение свойства вашим пользовательским значением по умолчанию.
Сбросить к пользовательскому значению	Вернуть свойству ваше пользовательское значение по умолчанию.

Команда	Действие
Сбросить к системному значению	Вернуть свойству поставляемое (заводское) значение.
Удалить пользовательское значение	Забыть ваше переопределение; снова действует системное значение.
Настроить…	Открыть диалог настройки для более широкого редактирования.

Изменения вступают в силу **сразу** — перезапускать приложение не нужно.

> **Метрика и дюймы.** Как дескрипторы несут и `DefaultValue`, и `InchDefaultValue`, так и пользовательское значение хранится для каждой системы единиц: задавая его в дюймах, вы записываете дюймовое значение, независимое от метрического. Каждое используется, когда проект работает в соответствующей системе единиц.

> **Сгруппированные строки инспектора.** Когда несколько свойств делят одну строку инспектора — составной тип `Compact` или строка radio-edit с переключением единиц (§9) — сохранение или сброс строки действует на **все** видимые в ней свойства вместе, а не только на то, по которому вы щёлкнули.

Где хранятся пользовательские значения

Пользовательские значения хранятся в файлах `.usrdef`. Они подключаются при старте главной конфигурацией через необязательные включения с масками — например, для семейства `Operations`:

```
<SCInclude Optional="True">$(SUPPLEMENT_FOLDER)\Defaults\*.usrdef</SCInclude>
<SCInclude Optional="True">$(LOCAL_OPERATIONS_FOLDER)\*.usrdef</SCInclude>
```

Они включаются **после** системных дескрипторов, поэтому их значения побеждают поставляемые. Команда *«Сохранить как пользовательское значение по умолчанию»* записывает эти файлы в вашу пользовательскую область автоматически (расположение `$(LOCAL_OPERATIONS_FOLDER)`, рядом с вашими пользовательскими операциями — см. §7), так что при обычном использовании вы никогда не правите их вручную. Другие семейства дескрипторов загружаются так же своей конфигурацией.

Формат файла `.usrdef`

Для просмотра, контроля версий или скриптового развёртывания файл `.usrdef` — это обычный XMLProperties. Он повторяет путь от корневого типа до переопределяемого свойства, и листовой элемент несёт переопределение в атрибуте `UserDefaultValue` (или `UserInchDefaultValue`):

```
<?xml version="1.0" encoding="UTF-8" ?>
<SCCollection>
    <SCNameSpace ID="Operations">
        <TST2DContouringOp>
            <HelicalMachining>
                <Enabled UserDefaultValue="True"/>
            </HelicalMachining>
        </TST2DContouringOp>
    </SCNameSpace>
</SCCollection>
```

- По одному **`<SCNameSpace ID="…">`** на пространство имён (например, `Operations`, `Tools`, …); см. пространства имён.
- Вложенность элементов повторяет **путь свойства** от корневого типа (`TST2DContouringOp`) через любые подобъекты (`HelicalMachining`) до свойства (`Enabled`).
- Листовой элемент несёт **`UserDefaultValue`** для метрического значения и/или **`UserInchDefaultValue`** для дюймового — но никогда системный `DefaultValue`, который остаётся нетронутым.
- По соглашению — **один файл на корневой тип**, с именем `<RootType>.usrdef` (например, `TST2DContouringOp.usrdef`).
- Если `.usrdef` ссылается на тип или свойство, которого нет в работающей системе, эта запись просто **игнорируется** — устаревшее переопределение не вредит.

Развёртывание значений по умолчанию между рабочими местами

Поскольку пользовательские значения — это просто файлы, стандартную конфигурацию можно переносить между рабочими местами. Ваши пользовательские операции вместе с их пользовательскими значениями можно экспортировать в единый архив и импортировать из него — это позволяет интегратору подготовить корпоративный стандарт на одной машине и развернуть его на другие. Архив содержит конфигурацию операций, файлы пользовательских операций, опциональные иконки и файлы `Defaults\*.usrdef`.

Полное отключение пользовательских значений

Если нужно разом вернуть заводское поведение, есть «аварийный выход», отключающий **все** пользовательские значения сразу: он перемещает каждый `*.usrdef` в резервную подпапку, чтобы

ни один не загружался. Поскольку дескрипторы читаются при старте, для полного вступления изменения в силу **перезапустите приложение**.

Назад к оглавлению **Фреймворк XMLProperties**.

XML-дескрипторы операций

Этот раздел посвящён **XML-дескрипторам технологических операций** — как описать и зарегистрировать операцию и как устроены поставляемые операции. Это не полное руководство по тому, что каждая операция *делает*; речь о создании и понимании их XML-дескрипторов. Раздел опирается на фреймворк XMLProperties (сначала прочитайте синтаксис дескрипторов и язык выражений).

- **XML-дескрипторы операций** — анатомия дескриптора операции: регистрационная запись, выбор базового типа, заголовок, параметры, `ContainerID` / `SolverID`, `MultiGroup` и куда поместить ваш файл.
- **Справочник иерархии операций** — полное дерево наследования всех поставляемых операций (ID, caption, родитель, контейнер/решатель, исходный файл), чтобы помочь выбрать базовый тип. *Генерируется — не редактировать вручную.*

Общие правила (не редактировать `Supplement`, где лежат поставляемые файлы, соглашения в примерах) см. в разделе Фреймворк XMLProperties, а целевую аудиторию — в корне документации.

Дескрипторы операций

Технологическая операция описывается одним большим `ComplexType`, унаследованным от существующей операции, плюс небольшой **регистрационной записью**, объявляющей её системе. Эта глава показывает анатомию операции на двух реальных файлах: `Operations/MillOperations/FaceMillingOp.xml` (поставляемая операция) и парном примере `OperationSimpleNet.xml` из репозитория `cam-api-examples` (операция, траектория которой вычисляется расширением на C#).

Поставляемые файлы операций лежат в `Operations\` и `Operations\MillOperations\` и подключаются через `Operations.xml` — используйте их как справочник. Ваша собственная операция туда **не** помещается. Регистрируйте её одним из двух механизмов из §7: положите файл `*_ExtOp.xml` в общую папку контейнеров (подхватывается автоматически) либо укажите мастеру **Operations Manager** (Utilities → Operations Manager) ваш XML-файл операции. В любом случае файл — это `<SCCollection>` в пространстве имён **`Operations`** и использует ровно тот синтаксис, что показан ниже.

1 Две части

Чтобы добавить операцию, вы предоставляете:

1. **Регистрационную запись** в пространстве имён `OperationRegistrator`, чей `TypeName` равен `ID` вашего типа операции.
2. **Сам тип операции**, унаследованный от существующей базовой операции.

```
<SCCollection>
    <!-- 1. Регистрируем имя типа -->
    <SCNameSpace ID="OperationRegistrator">
        <SCType ID="RegTSTFaceMillingOp" type="TRegisterOperationRecord" Enabled="True">
            <TypeName DefaultValue="TSTFaceMillingOp"/>
        </SCType>
    </SCNameSpace>

    <!-- 2. Тип операции (его ID должен совпадать с TypeName выше) -->
    <SCType ID="TSTFaceMillingOp" Caption="FaceMilling" type="TSTMillOp" Enabled="True">
        ...
    </SCType>
</SCCollection>
```

(реальный пример: `Operations/MillOperations/FaceMillingOp.xml`)

Зачем нужна регистрационная запись

В системе может быть очень много дескрипторов. Пространство имён `OperationRegistrator` — это небольшой быстрый **индекс поиска**: он перечисляет, какие XML-типы действительно являются технологическими операциями, чтобы система находила их напрямую, а не сканировала весь массив дескрипторов. Запись — это просто `TRegisterOperationRecord`, чей `TypeName` указывает на `ID` вашего типа операции. Если забыть запись, ваш тип существует, но не распознаётся как операция.

2 Выбор базового типа

Операции наследуют массу структуры (секция инструмента, подачи, врезания, симуляция, состояние станка, …) от цепочки базовых операций. Выберите базу, поведение которой ближе всего к вашему, и переопределяйте/расширяйте от неё:

Базовый тип	**Для чего**
`TSTMillOp` (семейство фрезерования)	Фрезерные операции — например, от него наследуется `TSTFaceMillingOp`
База токарных операций	Токарные операции
База осевых / сверлильных	Сверление и осевые циклы
`TSTMillExtensionOp`	Операции, траектория которых вычисляется **расширением CAM API** (см. §3)

Проще всего найти нужную базу, открыв поставляемую операцию, наиболее похожую на желаемую, и переиспользовав её базовый тип. Полное дерево наследования всех поставляемых операций — ID, подписи, родители, реализующие классы и исходные файлы — сведено в **справочнике иерархии операций**.

3 Заголовок операции

Ближе к началу типа операции вы переопределяете набор унаследованных «заголовочных» членов, которые идентифицируют и представляют операцию:

```
<SCType ID="TSTFaceMillingOp" Caption="FaceMilling" type="TSTMillOp" Enabled="True">
    <GUID        DefaultValue="{9F01E1A0-6F3E-4C7A-9C52-9A024CED54FF}"/>
    <ContainerID DefaultValue="{F4DE00D1-7AE3-468E-BA5C-ED3C8275CBF8}"/>

    <Name    DefaultValue="FaceMilling"/>
    <Comment DefaultValue="Face Milling"/>
    <OperationGroup DefaultValue="Mill"/>

    <Image DefaultValue="Images\FaceMilling.png"/>
    <Icon  DefaultValue="Images\FaceMilling_Ico.bmp"/>
```

```
    <Video DefaultValue="Video\FaceMilling.wmv"/>
    ...
</SCType>
```

(реальный пример: Operations/MillOperations/FaceMillingOp.xml)

Заголовочный член	**Смысл**
`GUID`	Уникальная постоянная идентичность типа операции. Генерируйте новый; никогда не переиспользуйте.
`ContainerID`	GUID класса Delphi-**контейнера**, который представляет операцию в дереве операций (см. ниже). Обычно наследуется; переопределяйте только когда вашей операции нужен конкретный класс контейнера.
`SolverID`	Идентифицирует **решатель**, вычисляющий траекторию (см. ниже). Обычно наследуется.
`Name` / `Comment`	Имя для отображения и описание по умолчанию.
`OperationGroup`	Широкая технологическая группа, к которой относится операция (`Mill`, `Lathe`, ...).
`Image` / `Icon` / `Video`	Иллюстрация, иконка в списке и обучающий ролик.

Полезные атрибуты на самом элементе типа:

- `Enabled="True"` — операция доступна. Установите `False`, чтобы поставлять её отключённой.
- `DefaultVisibility="False"` — специализированная операция остаётся загружаемой, но скрыта из меню по умолчанию.
- `Version` — версия схемы; увеличивайте её при изменении структуры операции, чтобы существующие сохранённые данные корректно мигрировали.

ContainerID и SolverID — контейнер, решатель и траектория

Эти два идентификатора описывают, как реализована операция. Понимание этого разделения помогает решить, что наследовать, а что переопределять.

- **`ContainerID` — класс контейнера.** Это GUID класса Delphi, который *представляет операцию* в дереве операций CAM-системы. Система создаёт экземпляр этого класса контейнера; затем контейнер строит экземпляр операции из XML-дескриптора, **владеет им и всеми его параметрами** и является объектом, с которым общается остальная система.

- `SolverID` **— (необязательный) решатель.** Когда задан, контейнер создаёт отдельный подкласс-**решатель**, идентифицируемый `SolverID`, и этот решатель вычисляет траекторию. Когда `SolverID` *не* задан, отдельного решателя нет и **класс контейнера вычисляет траекторию сам**.

> **Почему существует это разделение (история).** Изначально не было ни XML-дескрипторов, ни отдельных решателей — операция была просто классом-контейнером, определявшим всё своё поведение: свои параметры и вычисление траектории. XML-дескрипторы (чтобы объявлять параметры декларативно) и решатели (чтобы вынести вычисление траектории, в том числе в расширения CAM API) были добавлены позже. Поэтому операция и сейчас может работать с одним лишь контейнером без решателя.

На практике вы наследуете и `ContainerID`, и `SolverID` от своего базового типа. Переопределяйте `ContainerID` только когда вашей операции нужен конкретный класс контейнера; задавайте `SolverID`, чтобы выбрать движок, вычисляющий траекторию.

`SolverID` принимает **две формы**:

1. **GUID существующего встроенного класса-решателя** — переиспользовать движок, который система уже предоставляет:

   ```
   <SolverID DefaultValue="{301F6C21-2499-4514-83D1-72A4931529BE}"/>
   ```

2. **Идентификатор написанного вами расширения CAM API** — конкретно расширения `OperationSolver`. Здесь `SolverID` — это строковый `id` расширения, а не GUID:

   ```
   <!-- из OperationSimpleNet.xml -->
   <SCType ID="TSimpleNetOP" Caption="Simple .NET" type="TSTMillExtensionOp"
   Enabled="True">
       <GUID     DefaultValue="{BDD2CAD7-D2A9-43C4-86E3-399FCB439FC0}"/>
       <SolverID DefaultValue="Extension.Operation.Simple.Net"/>
       ...
   </SCType>
   ```

   Тот же `id` объявлен в файле настроек расширения, в записи `OperationSolver`:

   ```
   // ExtensionOperationSimpleNet.settings.json
   "extensions": [
       { "OperationSolver": { "id": "Extension.Operation.Simple.Net" } }
   ]
   ```

Рабочие компилируемые расширения `OperationSolver` лежат в репозитории `cam-api-examples` в `Operation/` (`ExtensionOperationSimpleNet`, `ExtensionOperationParamsNet`). Наследуйте такую операцию от `TSTMillExtensionOp` (или подходящей базы расширения), чтобы стандартная механика была подключена за вас.

Группировка в меню «новая операция» — `MultiGroup`

`MultiGroup` управляет тем, где операция появляется в дереве *создания новой операции*. Он позволяет одной операции отображаться в **нескольких группах сразу** и позволяет разным конфигурациям интерфейса (например, *Beginner* и *Expert*) помещать её в разные группы. Каждая запись — это `TLinkToParentMultiGroup`, чей `ID` — целевая группа; `OrderInGroup` задаёт её позицию внутри этой группы.

```
<MultiGroup>
    <SCType ID="Group3DEntry" OrderInGroup="1" type="TLinkToParentMultiGroup"/>
    <SCType ID="Roughing"     OrderInGroup="1" type="TLinkToParentMultiGroup"/>
</MultiGroup>
```

(реальный пример: Operations/MillOperations/FaceMillingOp.xml)

Используйте `MultiGroup` + `OrderInGroup` для всякого размещения и упорядочивания в меню.

4 Добавление параметров

Параметры — это просто члены типа операции. Группируйте их под плейсхолдерами инспектора через `Parent`, упорядочивайте, прикрепляйте иконку через `ImageFile` и делайте их динамическими языком выражений из языка выражений.

```
<!-- перечисление, управляющее стратегией операции -->
<SCType ID="Strategy" Caption="Strategy" type="Enumerated" DefaultValue="Spiral"
        ImageFile="$(SUPPLEMENT_FOLDER)\operations\TypeImages\FaceMillingStrategy.bmp">
    <SCType ID="Spiral"      Caption="Spiral"           type="None"/>
    <SCType ID="OptimZigzag" Caption="Optimized zigzag" type="None"/>
    <SCType ID="Zigzag"      Caption="Zigzag"           type="None"/>
    <SCType ID="OneWay"      Caption="One way"          type="None"/>
    <SCType ID="OnePass"     Caption="One pass"         type="None"/>
</SCType>

<!-- значение под Strategy, показываемое только для некоторых стратегий -->
<SCType ID="Step" Caption="Step" type="TPercentageValue"
        Parent="Strategy" Visible="[Strategy] != 4">
    <ValueType    DefaultValue="Percent"/>
```

```
    <PercentValue DefaultValue="75"/>
</SCType>
```

(реальный пример: Operations/MillOperations/FaceMillingOp.xml)

Категории и плейсхолдеры

Параметры назначаются в категории инспектора и в **плейсхолдеры** — контейнеры, унаследованные от базы, — через `Category` и `Parent`:

```
<SCType ID="LeadInDistance" Caption="Lead in" type="TPercentageValue"
        Category="TPropertiesCategoryList.Leads" Parent="LeadsPlaceHolder"

ImageFile="$(SUPPLEMENT_FOLDER)\operations\TypeImages\FaceMillingLeadInDistance.bmp">
    <PercentValue DefaultValue="10"/>
</SCType>
```

`Parent="LeadsPlaceHolder"` помещает эту строку под группирующий узел, заданный базовой операцией, так что связанные параметры собираются вместе независимо от того, где они объявлены.

Переопределение унаследованных параметров

Поскольку ваша операция наследует полный набор параметров, бóльшая часть файла операции — это *переопределения* унаследованных членов: подстройка значений по умолчанию, скрытие неприменимого или повторное включение вариантов. Используйте форму переопределения (`<MemberName .../>`) из §4.2:

```
<Stock Visible="False"/>                <!-- скрыть унаследованный параметр -->
<ConditionsSection>
    <Feeds>
        <EngageFeed  Enabled="False"/>  <!-- отключить неприменимые подачи -->
        <RetractFeed Enabled="False"/>
    </Feeds>
</ConditionsSection>
```

5 Чек-лист для новой операции

1. Сгенерируйте новый `GUID`.
2. Выберите `SolverID`: GUID встроенного решателя или `id` вашего расширения CAM API `OperationSolver` (в этом случае наследуйте от `TSTMillExtensionOp`).
3. Добавьте регистрационную запись в `OperationRegistrator` с `TypeName` = `ID` вашего типа.
4. Унаследуйте тип от ближайшей существующей базовой операции.

5. Переопределите заголовочные члены (`Name`, `Comment`, `OperationGroup`, `Image`, `Icon`, `Video`) и задайте `MultiGroup` для размещения в меню.
6. Объявите свои параметры стратегии и переопределите унаследованные по необходимости; назначьте `Category`/`Parent`, чтобы они попали в нужную группу инспектора.
7. Зарегистрируйте операцию — сохраните как `<YourName>_ExtOp.xml` в `$(COMMON_CONTAINERS_FOLDER)` либо добавьте через мастер Operations Manager (см. §7). Не редактируйте поставляемые файлы в `Supplement`.
8. Увеличивайте `Version` всякий раз, когда позже меняете структуру.

Далее: **Дескрипторы станков**

Справочник иерархии операций

> ⚙ **Сгенерировано автоматически — не редактировать вручную.** Эта страница создаётся скриптом `docs/xml-properties/gen-operation-hierarchy.py` из снимка реестра, объединённого с XML из Supplement. Перезапустите скрипт, чтобы обновить её; ручные правки перезаписываются.

Это приложение к Дескрипторам операций отображает полное дерево наследования дескрипторов технологических операций, поставляемых с CAM-системой. Используйте его, чтобы **выбрать базовый тип для наследования**: найдите поставляемую операцию, наиболее близкую к желаемой, и унаследуйте свою операцию от неё (или от одной из её баз).

Данные объединяют реестр операций, сообщаемый работающей системой, с объявлениями дескрипторов в XML-файлах, поэтому ID, подписи, родители, классы контейнеров, ID контейнеров, решатели и исходные файлы — это реальные, актуальные значения.

Как читать эту таблицу

- **Строки упорядочены в глубину** от самого абстрактного корня (`TOperationDescriptor`) вниз к конкретным операциям. Колонка **Ур.** даёт глубину наследования, а **Родитель** называет тип, от которого наследуется каждая строка; вместе с порядком они восстанавливают всё дерево.
- **ID** — ID дескриптора. Это значение `type=`, которое вы ставите на своей операции, чтобы наследоваться от него, и (для конкретных операций) `TypeName`, используемый в регистрационной записи. Показывается всегда, в том числе для абстрактных базовых типов.
- **Caption** — человекочитаемое имя, показываемое в интерфейсе.
- **Родитель** — дескриптор, от которого наследуется данный.
- **Класс контейнера / ContainerID** — Delphi-**контейнер**, который представляет операцию в дереве операций, строит экземпляр из XML-дескриптора, владеет им и либо вычисляет траекторию сам, либо делегирует решателю. В ячейке на первой строке показан класс, а на второй — его GUID класса (`ContainerID`). См. *Контейнер, решатель и траектория* в разделе 4.3. Контейнер есть у каждой операции. **Абстрактные базовые типы** (строки, не являющиеся зарегистрированными инстанцируемыми операциями) показывают контейнер, который они объявляют или наследуют; самые абстрактные дескрипторы откатываются к базовому контейнеру `TSTAbstractTechOp`. Абстрактные базы часто — лучшие цели наследования для новой операции.
- **Класс решателя / SolverID** — **решатель** траектории, показанный так же, как контейнер: реализующий Delphi-класс на первой строке и его `SolverID` на второй. **Пусто** означает, что операция использует общий фреймворк решателя по умолчанию (а когда отдельного решателя нет, её класс контейнера вычисляет траекторию напрямую). Значение здесь показывают только операции с выделенным решателем.

- **Объявлено в** — XML-файл (относительно папки `Supplement`), который объявляет дескриптор. Откройте его, чтобы изучить дескриптор целиком.

> **Выбор базы.** Для новой фрезерной операции обычные цели — `TST3DMillOp` (3D) или `TSTAbstract5DOp` (3D/5D), либо конкретный «сосед», на который вы хотите быть похожим. Для токарной — наследуйте от `AbstractLatheOP` или `LatheContouringOp`. Для операции, траектория которой вычисляется вашим расширением CAM API, наследуйте от базы расширения (например, `TSTMillExtensionOp`) и задайте в `SolverID` id вашего расширения — см. раздел 4.3.

> **Сопровождение.** Эта страница генерируется скриптом `docs/xml-properties/gen-operation-hierarchy.py` из снимка реестра (`data/WholeOpList.json`), объединённого с XML из Supplement. Перезапустите скрипт после обновления снимка из более новой сборки. Несколько внутренних/зашифрованных типов операций намеренно пропущены.

145 зарегистрированных операций + 50 абстрактных базовых типов = 195 записей.

Полная иерархия

Ур.	ID	Caption	Родитель	Класс Conta
0	`TOperationDescriptor`	Operation descriptor	`TOperationPublicDescription`	`TSTAbs {AA0BC D705E8`
1	`TAbstractTechnologicalOperation`	Abstract Technological Operation	`TOperationDescriptor`	`TSTAbs {AA0BC D705E8`
2	`TAbstractExternalOperation`	Abstract external operation	`TAbstractTechnologicalOperation`	`TSTAbs {AA0BC D705E8`
3	`AbstractOP`	Abstract operation	`TAbstractExternalOperation`	`TSTAbs {AA0BC D705E8`
4	`AbstractTurnOP`	Abstract Turn operation	`AbstractOP`	`TSCLat {E05F6 05585`

Ур.	ID	Caption	Родитель	Класс Conta
5	AbstractAxialOP	Abstract Axial OP	AbstractTurnOP	TSCLat {E05F6 05585:
6	LatheDrillOp	Lathe drilling (obsolete)	AbstractAxialOP	TSCLat {E05F6 05585:
5	AbstractLatheOP	Abstract Lathe OP	AbstractTurnOP	TSCLat {E05F6 05585:
6	TNCLatheOp	G-code based lathe	AbstractLatheOP	TNCLat {CE71F 187F0E
6	LatheContouringOp	Lathe contouring	AbstractLatheOP	TSTLat {6B3F2 B1250E
7	TLatheFaceMachining	Lathe facing	LatheContouringOp	TSTLat {6B3F2 B1250E
7	TLathePartOff	Lathe part-off	LatheContouringOp	TSTLat {6B3F2 B1250E
8	TPartOffWithTakeover	Part-off with takeover	TLathePartOff	TSTPar {B9A19 718D22
7	TLatheODAdaptive	OD Adaptive turning	LatheContouringOp	TSTLat {6B3F2 B1250E

Ур.	ID	Caption	Родитель	Класс Conta
7	TLatheODFinishing	OD Finishing	LatheContouringOp	TSTLat {6B3F2 B1250E
8	TLatheIDFinishing	ID Finishing	TLatheODFinishing	TSTLat {6B3F2 B1250E
7	TLatheODGrooving	OD Grooving	LatheContouringOp	TSTLat {6B3F2 B1250E
8	TLatheFaceGrooving	Face grooving	TLatheODGrooving	TSTLat {6B3F2 B1250E
8	TLatheIDGrooving	ID Grooving	TLatheODGrooving	TSTLat {6B3F2 B1250E
7	TLatheODTurning	OD Roughing	LatheContouringOp	TSTLat {6B3F2 B1250E
8	TLatheIDTurning	ID Roughing	TLatheODTurning	TSTLat {6B3F2 B1250E
7	TLatheODThreading	OD Threading	LatheContouringOp	TSTLat {6B3F2 B1250E
8	TLatheIDThreading	ID Threading	TLatheODThreading	TSTLat {6B3F2 B1250E
7	TLatheProfileThreading	Profile threading	LatheContouringOp	TSTLat {6B3F2 B1250E

Ур.	ID	Caption	Родитель	Класс Conta
6	LatheFacingOp	Lathe facing (obsolete)	AbstractLatheOP	TSCLat {E05F6 05585:
6	LatheFinishOp	Lathe finishing (obsolete)	AbstractLatheOP	TSCLat {E05F6 05585:
6	LatheGroovOp	Lathe grooving (obsolete)	AbstractLatheOP	TSCLat {E05F6 05585:
6	LathePartOff	Lathe part-off (obsolete)	AbstractLatheOP	TSCLat {E05F6 05585:
6	LatheRoughOp	Lathe roughing (obsolete)	AbstractLatheOP	TSCLat {E05F6 05585:
6	LatheThreadOp	Lathe threading (obsolete)	AbstractLatheOP	TSCLat {E05F6 05585:
6	LatheContouringOP		AbstractLatheOP	TSCLat {E05F6

Ур.	ID	Caption	Родитель	Класс Conta
				05585:
7	TSTLatheContouringCopy	Copy of lathe operation	LatheContouringOP	TSTLat {F355: 311A7
4	AbstractAuxiliaryOp	Auxiliary operation	AbstractOP	TSCAux {4B61A 0BD50
5	TSTAuxOpCopy	Copy of auxiliary operation	AbstractAuxiliaryOp	TSTAux {CDD78 862D4
2	TSTAbstractMillOp	Abstract mill operation	TAbstractTechnologicalOperation	TSTAbs {AA0BC D705E
3	TSTMillOpCopy	Copy of milling operation	TSTAbstractMillOp	TSTMil {E6A66 D0008
4	TSTFBMMillOpCopy	Copy of FBM operation	TSTMillOpCopy	TSTFBM {D388D 24E55
4	TSTTechOpGroupCopy	Copy of Group	TSTMillOpCopy	TSTTec {FC232 023DF
4	TSTFeatureSubOp	Copy of milling operation	TSTMillOpCopy	TSTFea {6B6DA 12BDD
4	TSTHoleMachiningOpCopy	Copy of milling operation	TSTMillOpCopy	TSTHol {6DE68 115CB
4	TSTFeatureOpGroupCopy	Copy of Multiply	TSTMillOpCopy	TSTFea {B58C4

Ур.	ID	Caption	Родитель	Класс Conta
		Group		9DC9AC
3	TNCMillOp	G-code based	TSTAbstractMillOp	TNCMil {D2921 AFF841
3	TSTMillOp	Mill operation	TSTAbstractMillOp	TSTAbs {AA0BC D705E8
4	TST3DMillOp	3D milling operation	TSTMillOp	TSTAbs {AA0BC D705E8
5	TST25DChamferOp	2.5D chamfer machining	TST3DMillOp	TST25D {5348F 819A2E
5	TSTAbstract25DOp	Abstract 25D operation	TST3DMillOp	TSTAbs {AA0BC D705E8
6	TST25DWallMachiningOp	2.5D wall machining	TSTAbstract25DOp	TST25D {1B6B6 D37F69
6	TSTAbstract25DPocketOp	Abstract 25D pocket operation	TSTAbstract25DOp	TSTAbs {AA0BC D705E8
7	TST25DFlatLandOp	2.5D flat land	TSTAbstract25DPocketOp	TST25D {1AF04 51EDC1
7	TST25DPocketingOp	2.5D pocketing	TSTAbstract25DPocketOp	TST25D {1410C E776D5
5	TSTAbstractContouringOp	Abstract contouring	TST3DMillOp	TSTAbs {AA0BC

Ур.	ID	Caption	Родитель	Класс Conta
		operation		D705E8
6	TST2DContouringOp	2D contouring	TSTAbstractContouringOp	TSTCur {6779(AC3A59
7	TST25DContouringOp	2.5D contouring	TST2DContouringOp	TSTCur {6779(AC3A59
7	KnifeOp2D	2D knife cutting	TST2DContouringOp	TSTKni {4F4D1 41D465
7	SawingOp2D	Disc cutting 2D	TST2DContouringOp	TSTSav {B321A B64637
6	TST3DContouringOp	3D contouring	TSTAbstractContouringOp	TSTCur {7A2DA A4172D
6	TSTJetCuttingOp	Jet cutting	TSTAbstractContouringOp	TSTCut {52822 8DD583
5	TSTAbstractEngraveOp	Abstract engrave operation	TST3DMillOp	TSTAbs {AA0BC D705E8
6	TSTEngraveOp	Engraving	TSTAbstractEngraveOp	TSTEng {66F71 DB337F
6	TSTPocketingOp	Pocketing	TSTAbstractEngraveOp	TSTPoc {BB586 72A595

Ур.	ID	Caption	Родитель	Класс Conta
7	TSTRestPocketingOp	Area rest milling	TSTPocketingOp	TSTPoc {BB58 72A59
5	TSTStringOp	Abstract string operation	TST3DMillOp	TSTAbs {AA0B0 D705E8
6	TSTAbstractDriveOp	Abstract drive operation	TSTStringOp	TSTAbs {AA0B0 D705E8
7	TSTFinishingDriveOp	Drive	TSTAbstractDriveOp	TSTDri {754E2 B7F73D
8	TSTRestFinishingDriveOp	Finishing drive (rest)	TSTFinishingDriveOp	TSTDri {754E2 B7F73D
7	TSTRoughingDriveOp	Roughing drive	TSTAbstractDriveOp	TSTDri {A57D3 9E6B92
6	TSTAbstractPlaneOp	Abstract plane operation	TSTStringOp	TSTAbs {AA0B0 D705E8
7	TSTFinishingPlaneOp	Finishing plane	TSTAbstractPlaneOp	TSTPla {DC54E CDB48E
8	TSTPlaneWaterlineOp	Complex	TSTFinishingPlaneOp	TSTPla {20DAF 0BD67D
9	TSTRestPlaneWaterlineFinishingOp	Complex (rest)	TSTPlaneWaterlineOp	TSTPla {20DAF 0BD67D

Ур.	ID	Caption	Родитель	Класс Conta
8	TSTComplexFinishingPlaneOp	Finishing plane	TSTFinishingPlaneOp	TSTPla {DC54E CDB48E
8	TSTFinishingPlaneOpForOptPlane	Finishing plane	TSTFinishingPlaneOp	TSTPla {DC54E CDB48E
8	TSTRestFinishingPlaneOp	Finishing plane (rest)	TSTFinishingPlaneOp	TSTPla {DC54E CDB48E
9	TSTRestComplexFinishingPlaneOp	Finishing plane (rest)	TSTRestFinishingPlaneOp	TSTPla {DC54E CDB48E
9	TSTRestFinishingPlaneOpForOptPlane	Finishing plane (rest)	TSTRestFinishingPlaneOp	TSTPla {DC54E CDB48E
8	TSTCrissCrossPlaneOp	Optimized plane	TSTFinishingPlaneOp	TSTPla {1C86(466554
9	TSTRestCrissCrossPlaneOp	Optimized plane (rest)	TSTCrissCrossPlaneOp	TSTPla {1C86(466554
7	TSTRoughingPlaneOp	Roughing plane	TSTAbstractPlaneOp	TSTPla {B87DE 50ACD4
5	TSTAbstractWaterlineOp	Abstract waterline operation	TST3DMillOp	TSTAbs {AA0BC D705E8
6	TSTFinishingWaterlineOp	Finishing waterline	TSTAbstractWaterlineOp	TSTWLF {224A6 60F403

Ур.	ID	Caption	Родитель	Класс Conta
7	TSTCombineOp	Combine	TSTFinishingWaterlineOp	TSTWLS {50AB7 303D6(
7	TSTComplexFinishingWaterlineOp	Finishing waterline	TSTFinishingWaterlineOp	TSTWLF {224A(60F40:
7	TSTRestFinishingWaterlineOp	Finishing waterline (rest)	TSTFinishingWaterlineOp	TSTWLF {224A(60F40:
8	TSTRestComplexFinishingWaterlineOp	Finishing waterline (rest)	TSTRestFinishingWaterlineOp	TSTWLF {224A(60F40:
6	TSTRoughingWaterlineOp	Roughing waterline	TSTAbstractWaterlineOp	TSTWLF {E1BE2 3235D8
7	TSTFlatLandFinishingOp	Flat land	TSTRoughingWaterlineOp	TSTWLF {96618 417E88
7	TSTRestRoughingWaterlineOp	Roughing waterline (rest)	TSTRoughingWaterlineOp	TSTWLF {E1BE2 3235D8
5	TSTPocketCladdingOp	Area cladding	TST3DMillOp	TSTPoc {B91BA 9C3B9(
5	TSTCladding3DOp	Cladding 3D	TST3DMillOp	TSTCla {FFDC7 5050BE
5	TSTCurveCladdingOp	Curve cladding	TST3DMillOp	TSTCur {AF41(85C4A8

Ур.	ID	Caption	Родитель	Класс Conta
5	TSTDrillOp	Hole machining	TST3DMillOp	TSTDri {5935E 8A979{
4	TSTAbstract5DOp	Abstract 5D milling operation	TSTMillOp	TSTAbs {AA0BC D705E{
5	TSTHelicalOp	3D Helical	TSTAbstract5DOp	TSTHel {DC50D F2950{
6	TSTHelicalCladdingOp	Helical cladding	TSTHelicalOp	TSTHel {ADAAF AA505{
5	TSTUni3dOp	3D Surfacing	TSTAbstract5DOp	TSTUni {94D51 23ACB{
5	TSTWaterJetCutting4d	4D Jet cutting	TSTAbstract5DOp	TSTWat {7766{ FA041{
5	ToolEnd5DMachiningOp	5D Contouring	TSTAbstract5DOp	TSTIsc {F156{ 6B5C5{
6	ToolEnd4DMachiningOp	4D Contouring	ToolEnd5DMachiningOp	TSTIsc {F156{ 6B5C5{
6	KnifeOp5D	6D Knife cutting	ToolEnd5DMachiningOp	TSTKni {3D34E C241D{
6	ChamferingMillOp	Chamfering	ToolEnd5DMachiningOp	TSTCha {D983{ 412FA{

Ур.	ID	Caption	Родитель	Класс Conta
6	SawingOp5D	Disc cutting 6D	ToolEnd5DMachiningOp	TSTSav {C9B37 42F171
6	TCutting5DOp	Jet cutting 5D	ToolEnd5DMachiningOp	TSTCut {74C4D FE39A4
6	SawingOp	Sawing	ToolEnd5DMachiningOp	TSTSav {F4903 28321A
5	TSTMultiAxisOp	5D Roughing waterline	TSTAbstract5DOp	TIntMi {05436 D98BCD
5	TSTUni5DOp	5D Surfacing	TSTAbstract5DOp	TSTUni {4E143 813991
6	TSTUni4DOp	4D Surfacing	TSTUni5DOp	TSTUni {4E143 813991
6	TSTCladding5DOp	Cladding 5D	TSTUni5DOp	TSTCla {AF57A C41C04
7	TSTSurfaceSprayOp	Surface spraying	TSTCladding5DOp	TSurfa {9E7F6 8A6F88
6	TSTUni5DOp4D	Rotary waterline	TSTUni5DOp	TSTUni {C94FD E1B1C6

Ур.	ID	Caption	Родитель	Класс Конта
5	ToolEnd5DMachiningOpTPLinker	6D Contouring	TSTAbstract5DOp	TSTisc {430E1 C8E70(
6	TSTContourSprayOp	Contour spraying	ToolEnd5DMachiningOpTPLinker	TConto {9AAB2 3053C5
5	TSTBladeMillingOP	Blade Milling	TSTAbstract5DOp	TSTBla {ECCB3 F3334(
5	TSTCornerRestMachiningOP	Corners cleanup	TSTAbstract5DOp	TSTCor {08E47 6691DE
5	TSTDiscRoughingOp	Disc Roughing	TSTAbstract5DOp	TSTDis {9020A 0C399(
5	HoleMachiningOp	Hole machining	TSTAbstract5DOp	TSTDri {7754D 76BD02
6	LatheHoleMachiningOp	Lathe hole machining	HoleMachiningOp	TSTHol {AFB34 39C24(
5	TSTImpellerMachiningOp	Impeller Machining	TSTAbstract5DOp	TIntMi {0543(D98BCD
5	TSTFBMMillOp	Mill features machining	TSTAbstract5DOp	TFBMMi {3BE9E C8BAA3

Ур.	ID	Caption	Родитель	Класс Conta
5	TAbstractProbingOp	Mill part probing	TSTAbstract5DOp	TSTMea {CDCF3 C0F3F5
6	TMillPartProbingOP	Part probing	TAbstractProbingOp	TMillP {3357D F66079
6	TMillToolProbingOP	Tool probing	TAbstractProbingOp	TMillT {30DDF 1451A5
6	TAbstractTurnProbingOp	Turn probing	TAbstractProbingOp	TSTTur {6BC13 F1FB57
7	TTurnPartProbingOP	Turn part probing	TAbstractTurnProbingOp	TTurnP {6A35D 415F0E
7	TTurnToolProbingOP	Turn tool probing	TAbstractTurnProbingOp	TTurnT {2464E D605E6
5	TSTMorphOp	Morph	TSTAbstract5DOp	TSTMor {50A76 A0A69A
6	TSTMorphOp4D	Morph 4D	TSTMorphOp	TSTMor {AAB17 27C6E6
6	TSTMorphSprayOp	Morph spraying	TSTMorphOp	TMorph {2D717 AD3B93
5	TSTNonPlanarPrintingOp	Non-planar slicing	TSTAbstract5DOp	TSTNon {DC474 18777E

Ур.	ID	Caption	Родитель	Класс Conta
5	TSTOrthogonalMachiningOp	Orthogonal machining	TSTAbstract5DOp	TSTOrt {0FD22 F6F36E
5	TSTPencilOp	Pencil	TSTAbstract5DOp	TSTPer {2A1AD F8F846
5	TPointWeldingOp	Point welding	TSTAbstract5DOp	TSTPoi {587BA 25A2A6
5	RotaryMachiningOp	Rotary finishing	TSTAbstract5DOp	TSTRot {7362E BB096A
6	TSTRotarySprayOp	Rotary spraying	RotaryMachiningOp	TRotar {27955 E5E9BF
5	RoughingRotaryMachiningOp	Rotary roughing	TSTAbstract5DOp	TSTRou {3DACE 79E8D3
5	TSTScallopOp	Scallop	TSTAbstract5DOp	TSTSca {AFE96 D9EB62
6	TSTGeodesicOp	5D by meshes	TSTScallopOp	TSTGeo {EB52F 995C53
5	TSTSwarfOp	Swarf	TSTAbstract5DOp	TSTSwa {0F267 397194
5	TSTWaterlineOp	Undercut waterline	TSTAbstract5DOp	TSTWat {C5678 FD7AC5

Ур.	ID	Caption	Родитель	Класс Conta
5	TWeldingOp5D	Welding 5D	TSTAbstract5DOp	TSTWel {877BA 5A95CI
5	TWeldingOp6D	Welding 6D	TSTAbstract5DOp	TSTWel {6546(C7899(
4	TMillOpWithDisabledProps	Aux Op	TSTMillOp	TSTAbs {AA0BC D705E8
5	TSTAbstractPickAndPlace	Pick and place	TMillOpWithDisabledProps	TSTAbs {AA0BC D705E8
6	TPNPbySelectedNode		TSTAbstractPickAndPlace	TSTAbs {AA0BC D705E8
7	TPNPwithGripperTool		TPNPbySelectedNode	TSTAbs {AA0BC D705E8
8	TPNPCalculator		TPNPwithGripperTool	TSTAbs {AA0BC D705E8
9	TPNPwithOutputMode		TPNPCalculator	TSTAbs {AA0BC D705E8
10	TPNPwithClamps		TPNPwithOutputMode	TSTAbs {AA0BC D705E8
11	TPNPwithTurnTool	Bar feeding	TPNPwithClamps	TSTAbs {AA0BC D705E8

Ур.	ID	Caption	Родитель	Класс Conta
12	TSTBarFeeding	Bar feeding	TPNPwithTurnTool	TSTBar {C5B4I EC7D54
11	TSTPointPickAndPlaceOP	Point Pick and place	TPNPwithClamps	TSTPoi {64F4F 479984
11	TPNP2NextOp		TPNPwithClamps	TSTAbs {AA0BC D705E8
12	TPNPApproachCorrector		TPNP2NextOp	TSTAbs {AA0BC D705E8
13	TPNPwithPickCS		TPNPApproachCorrector	TSTAbs {AA0BC D705E8
14	TPNPwithActions		TPNPwithPickCS	TSTAbs {AA0BC D705E8
15	TPNPwithSafeSurf		TPNPwithActions	TSTAbs {AA0BC D705E8
16	TPNPwithEngageVectors		TPNPwithSafeSurf	TSTAbs {AA0BC D705E8
17	TPNPwithMPL		TPNPwithEngageVectors	TSTAbs {AA0BC D705E8
18	TSTPlace2Next	Place to next stage	TPNPwithMPL	TSTPla {C448I FD02DA

Ур.	ID	Caption	Родитель	Класс Conta
18	TPNPwithRemoveWorkpiece		TPNPwithMPL	TSTAbs {AA0BC D705E8
19	TSTPickAndPlace	Pick and place	TPNPwithRemoveWorkpiece	TSTPic {3D9EI 043BD9
12	TPNPwithSyncSpindles		TPNP2NextOp	TSTAbs {AA0BC D705E8
13	TPNP4SwissLathes		TPNPwithSyncSpindles	TSTAbs {AA0BC D705E8
14	TPNPwithTurnEngages		TPNP4SwissLathes	TSTAbs {AA0BC D705E8
15	TSTSubTakeoverOp	Abstract pick-n-place	TPNPwithTurnEngages	TSTSub {BC7F9 7340E4
15	TPNPwithCannedCycle		TPNPwithTurnEngages	TSTAbs {AA0BC D705E8
16	TPNPwithSyncOp		TPNPwithCannedCycle	TSTAbs {AA0BC D705E8
17	TSTTakeoverMTM	Turn take over	TPNPwithSyncOp	TSTTak {15B86 BD2392
14	TPNPwithTurnSetupLCS		TPNP4SwissLathes	TSTAbs {AA0BC D705E8

Ур.	ID	Caption	Родитель	Класс Conta
15	TTurnPNPwithPickPosition		TPNPwithTurnSetupLCS	TSTAbs {AA0BC D705E8
16	TTurnPNPwithReturn		TTurnPNPwithPickPosition	TSTAbs {AA0BC D705E8
17	TTurnPNPwithSwissPlace		TTurnPNPwithReturn	TSTAbs {AA0BC D705E8
18	TSTTurnPNP	Sub spindle working	TTurnPNPwithSwissPlace	TSTTur {89222 266FE9
5	TSTSwissWaitOp	Wait tool	TMillOpWithDisabledProps	TSTSwi {D7DCF 6063F2
4	TSTFaceMillingOp	Face Milling	TSTMillOp	TSTFac {F4DE6 ED3C82
3	TAbsWireEDMContouringOp	Wire EDM Contouring	TSTAbstractMillOp	TSTAbs {AA0BC D705E8
4	WireEDM2dContouringOp	Wire EDM 2D Contouring	TAbsWireEDMContouringOp	TSTWir {7F43E 64EB23
4	WireEDM4dContouringOp	Wire EDM 4D Contouring	TAbsWireEDMContouringOp	TSTWir {296FE AB5037
1	TSTTechOpGroup	Operations group	TOperationDescriptor	TSTTec {F3644 1DB644

Ур.	ID	Caption	Родитель	Класс Conta
2	TSTFBMRootGroup	FBM Mill	TSTTechOpGroup	TSTFBM {93B9E E368CA
2	TSTTechMillOpGroupRaw	Group	TSTTechOpGroup	TSTTec {857E6 753534
2	TSTTechMillOpGroupFin	Group	TSTTechOpGroup	TSTTec {857E6 753534
3	TSTComplexFinishingOp	Finishing complex	TSTTechMillOpGroupFin	TSTCom {F55EC B027D8
4	TSTRestComplexFinishingOp	Complex (rest)	TSTComplexFinishingOp	TSTCom {F55EC B027D8
3	TSTOptimizedPlaneOp	Finishing optimized plane	TSTTechMillOpGroupFin	TSTOpt {48878 BA9AFE
4	TSTRestOptimizedPlaneOp	Optimized plane (rest)	TSTOptimizedPlaneOp	TSTOpt {48878 BA9AFE
3	TSTRootGroup	Machine	TSTTechMillOpGroupFin	TSTRoo {815B3 D58C8(
2	TSTTechMillOpGroupRest	Group	TSTTechOpGroup	TSTTec {857E6 753534
3	TSTMultiplyGroup	Multiply group	TSTTechMillOpGroupRest	TMulti {97D77 64D2CA

Ур.	ID	Caption	Родитель	Класс Conta
2	TSTFeatureOpGroup	Multiply Group	TSTTechOpGroup	TSTFea {8C369 365714
2	TSTStageOpGroup	Setup stage	TSTTechOpGroup	TSTSta {1625: 1BAA3F
2	TSTTombOpGroup	Tomb group	TSTTechOpGroup	TSTTom {072E5 E1FC5D
2	TSTCustomPartOp		TSTTechOpGroup	TSTPar {2802A 55C526
3	TSTPartOpCopy	Copy of part	TSTCustomPartOp	TSTPar {B4017 967867
3	TSTPartOpGroup	Part	TSTCustomPartOp	TSTPar {2802A 55C526
2	TSTFBMOpGroup		TSTTechOpGroup	TSTFBM {5ED0E 6D68E9

Назад к **Дескрипторам операций** | указатель

Дескрипторы станков

Этот раздел посвящён **описанию станка** — структуре его файлов, кинематической схеме и параметрам, определяющим, как его траектория превращается в УП. Раздел опирается на фреймворк XMLProperties.

Большинство станков создаются интерактивно приложением **MachineMaker**; материалы здесь — для понимания устройства станка и для особых случаев, которые по-прежнему создаются или дорабатываются вручную.

- **Дескрипторы станков** — начните отсюда: структура файлов станка, полный простой пример, кинематическая цепочка узлов и каждая секция дескриптора (Description, Control Parameters, Schema, параметры состояния, Tooling, Machine dimensions).
- **Справочник узлов станка** — справочник-компаньон: полные таблицы свойств для каждого типа узла (`TMachineNode`, `TMachineAxis`, коннекторы, параметры состояния, `VisualProperties`), а также системы координат и единицы измерения.
- **Справочник параметров Machine Setup** — что делает каждый параметр на вкладке *Control Parameters* станка, с XML-фрагментом, который его переопределяет.
- **Продвинутые темы по станкам** — матрицы узлов, переключаемые / съёмные узлы, револьверные головы, коннекторы инструмента/заготовки, субстанки и каналы управления.
- **3D-модели станков** — подготовка, подключение и выравнивание геометрии узлов (`.osd` / `.stl`).

Общие правила (не редактировать `Supplement`, где лежат поставляемые файлы, соглашения в примерах) см. в разделе Фреймворк XMLProperties, а целевую аудиторию — в корне документации.

Дескрипторы станков

Дескриптор станка, как и операция, — это большой `ComplexType`, но его самая характерная часть — **кинематическая схема**: вложенное дерево осей и держателей, описывающее, как станок движется. Эта глава идёт по дескриптору станка **снаружи внутрь**: из каких файлов он состоит, полный простой пример, идея кинематической цепочки, а затем каждая секция по очереди. За исчерпывающим списком свойств каждого типа узла она отсылает к **справочнику узлов станка**.

> **Большинство станков строятся в MachineMaker, а не вручную.** Отдельное приложение **MachineMaker** позволяет создавать станок *интерактивно*: для распространённых типов кинематических схем оно само формирует дескриптор, без ручной правки XML. Оставьте рукописный XML (и подготовленные вручную 3D-модели) для особых, сложных или необычных станков, выходящих за рамки возможностей MachineMaker. Эта глава документирует ручной путь — но если MachineMaker может построить ваш станок, предпочтите его.

> **Сопутствующие страницы по станкам** — читайте вместе с этой главой:
>
> - **Справочник узлов станка** — полные таблицы свойств для каждого типа узла (`TMachineNode`, `TMachineAxis`, коннекторы, параметры состояния, `VisualProperties`), а также системы координат и единицы измерения.
> - **Справочник параметров Machine Setup** — что делает каждый параметр на вкладке *Control Parameters* станка (дуги, сингулярности, поворотные преобразования, 5-осевая компенсация, смена инструмента, симуляция, …), с XML-фрагментом, который его переопределяет.
> - **Продвинутые темы по станкам** — матрицы узлов, переключаемые/съёмные узлы, револьверные головы, коннекторы инструмента/заготовки, а также каналы управления и субстанки.
> - **3D-модели станков** — подготовка, подключение и выравнивание геометрии узлов (`.osd` / `.stl`).

Станок — это набор файлов

Станок — это не единый объект, спрятанный в приложении, а небольшой набор файлов, на которые можно указать:

- **дескриптор** — один `.xml`-файл, который *и есть* станок: его идентичность, параметры управления и кинематическая схема;
- **его 3D-модели** — геометрия `.osd` / `.stl`, на которую дескриптор ссылается через `ImageFile` (на каждом узле) и заголовочные члены `Image` / `Icon`; по соглашению хранится в подпапке `Images\` рядом с дескриптором (см. 3D-модели станков);

- опционально **значения по умолчанию станка** (`*.usrdef`), подстраивающие поставляемые станки.

Поставляемые файлы в `Machines\` — ваш справочник. Они задают **библиотеку типов** станков — абстрактные станки и типы-строительные-блоки, из которых собирается каждый станок, — загружаемую через `Machines/MachinesConfig.xml` в пространство имён **`Machines`**. Чаще всего вы будете читать два: `Machines/AbstractMachine.xml` (общая база, от которой наследуется каждый станок) и `Machines/MachineTypes.xml` (типы узлов, осей, держателей и параметров).

Конкретный станок, напротив, — это **отдельный файл**, загружаемый сам по себе (ниже). Как и с операциями, вы **не** редактируете `Supplement`: пользовательский станок — это отдельный файл в сканируемых папках станков (`$(SCHEMAS_FOLDER)` / `$(CUSTOM_SCHEMAS_FOLDER)`), обычно создаваемый инструментами построения станков (см. §7). Эта глава объясняет синтаксис дескриптора, из которого строятся такие станки.

Как загружается дескриптор станка

Станки загружаются иначе, чем операции, и это различие влияет на то, как их создавать.

- **Библиотека типов загружается один раз.** `MachinesConfig.xml` загружает общую **библиотеку типов** станков — абстрактные станки и строительные блоки узлов, осей, держателей и параметров (ниже). Она общая для каждого станка.
- **Каждый станок загружается изолированно.** Конкретный станок — это собственный XML-файл. Когда системе нужен конкретный станок, она строит *свежее* пространство имён `Machines` поверх общей библиотеки типов, включает **именно этот один файл станка**, а затем применяет значения по умолчанию станка (`Machines/MachinesDefaults.xml`, который подключает любые `*.usrdef`). Каждый станок получает собственное приватное пространство имён.
- **Идентификаторы локальны для одного станка.** Поскольку станки никогда не делят пространство имён, два разных файла станков могут безопасно переиспользовать одни и те же `ID` у `SCType`. Глобально уникальные имена членов не нужны — только уникальные *внутри* вашего станка.
- **Библиотека ленивая.** Установки могут содержать **тысячи** станков, поэтому библиотека станков лишь *сканирует* предопределённые папки станков на предмет лёгких метаданных (достаточных, чтобы их перечислить и искать) и полностью загружает дескриптор станка только тогда, когда этот станок действительно открывают. Помните: файл станка разбирается по требованию, изолированно, а не весь сразу при старте.

Практическое следствие для авторов: ваш файл станка — это самодостаточный `<SCCollection>` в пространстве имён `Machines`, наследующийся от библиотечного типа и переопределяющий/расширяющий его. Он лежит в сканируемой папке станков, а не в `Supplement`.

Строительные блоки

`MachineTypes.xml` определяет переиспользуемые типы, из которых собирается станок. Их всего несколько; **справочник узлов** перечисляет каждое свойство каждого из них.

Тип	Роль
`TMachineNode`	База для всего в дереве схемы — *жёсткая* часть (станина, колонна, стол). Имеет `ID`, матрицу преобразования `Matrix`, визуальные свойства и необязательный `ImageFile` (геометрия `.osd`/`.stl`).
`TMachineAxis`	Узел, который **движется**: `AxisType` (`Linear`/`Rotary`), `Direction` (только линейные оси), `Scale`, `RapidFeed` и `ParameterName`, связывающий его с параметром состояния.
`TMachineStateParameter`	Управляемое **значение**, которым приводится ось (положение, шпиндель, …): `Address`, пределы, режим управления. Не узел схемы — оси ссылаются на него по имени.
`TToolHolderNode` / `TWorkpieceHolderNode`	Гнёзда, несущие инструмент / заготовку, с `SupportedToolTypes`. Специализации: `TMillToolHolder`, `TLatheCutterHolder`, `TJetCutterHolder`, …

> **О наименовании.** ID типов `TToolHolderNode` и `TWorkpieceHolderNode` отображаются в **интерфейсе** как **«коннектор инструмента»** и **«коннектор заготовки»**. Это одно и то же — гнездо, где инструмент, соответственно заготовка, крепится к кинематической цепочке.

Существует также несколько предопределённых типов осей и параметров состояния (`TToolAxisX/Y/Z`, `TLatheSpindle`, `TRotaryTable`, `TAxisXPosition`, …), но они **унаследованные и необязательные**: введённые рано для экономии набора через наследование, они экономят мало и заставляют выяснять, какие умолчания они скрывают. **Предпочитайте объявлять оси и параметры явно** из `TMachineAxis` / `TMachineStateParameter`, чтобы значимые значения были видны прямо в вашем файле станка. (Шаблонные типы вроде револьверных в §3 — исключение: там наследование действительно оправдано.)

Полный простой станок

Конкретные станки наследуются от `AbstractMachine` (или более близкой базы вроде `AbstractMillMachine`), которая уже предоставляет каждую секцию станка:

Секция	Содержимое
`DescriptionPlaceHolder`	Идентичность и метаданные: `Name`, `Comment`, `Group`, `Developer`, `NCSystem`, файл постпроцессора (`SPPFile`), файл интерпретатора, система единиц.
`ControlData`	Параметры управления/поведения: поддержка дуг, поворотные преобразования, 5-осевая компенсация, обработка локальной СК (ORIGIN), формат вывода системы координат инструмента.
`MachineStateParameters`	Список управляемых параметров (состояние осей).
`Schema`	Само кинематическое дерево.
`Coolants`	Доступные каналы СОЖ.
`Simulation`	Метод/разрешение симуляции и настройки столкновений.
`ToolChange`	Позиция смены инструмента, тайминги и режим вывода.
`MachineDimensions`	Опрятное место для сбора ключевых **определяющих параметров** схемы.

Две секции, которые вы пишете почти для каждого станка, — это `MachineStateParameters` (значения, которыми управляет ЧПУ) и `Schema` (геометрия, движущаяся вместе с ними). Вот полный небольшой 3-осевой фрезерный станок, в котором **стол перемещается по X**, а **инструментальная голова несёт Y и Z** — распространённая компоновка станинного типа и хорошая иллюстрация того, что двигаться могут *обе* ветви:

```
<MachineStateParameters>
    <SCType ID="AxisXPos" Caption="X" type="TMachineStateParameter">
        <Address DefaultValue="X"/><Group DefaultValue="LinearAxis"/>
        <Min DefaultValue="-300"/><Max DefaultValue="300"/>
    </SCType>
    <SCType ID="AxisYPos" Caption="Y" type="TMachineStateParameter">
        <Address DefaultValue="Y"/><Group DefaultValue="LinearAxis"/>
        <Min DefaultValue="-200"/><Max DefaultValue="200"/>
    </SCType>
    <SCType ID="AxisZPos" Caption="Z" type="TMachineStateParameter">
        <Address DefaultValue="Z"/><Group DefaultValue="LinearAxis"/>
        <Min DefaultValue="0"/><Max DefaultValue="400"/><InitialValue DefaultValue="400"/>
    </SCType>
    <SCType ID="SpindlePos" Caption="Spindle" type="TMachineStateParameter">
        <Address DefaultValue="S"/><Group DefaultValue="RotaryAxis"/>
<AxisControl DefaultValue="Manual"/>
```

```
    </SCType>
</MachineStateParameters>

<Schema>
    <SCType ID="Base" Caption="Bed" type="TMachineNode" IsFloor="True">
        <ImageFile DefaultValue="Images\bed.osd"/>

        <!-- ВЕТКА ИНСТРУМЕНТА: Y несёт Z, тот несёт шпиндель, тот несёт инструмент -->
        <SCType ID="AxisY" Caption="Axis Y" type="TMachineAxis">
            <ParameterName DefaultValue="AxisYPos"/>
            <AxisType DefaultValue="Linear"/>
            <Direction><X DefaultValue="0"/><Y DefaultValue="1"/><Z
DefaultValue="0"/></Direction>
            <SCType ID="AxisZ" Caption="Axis Z" type="TMachineAxis">
                <ParameterName DefaultValue="AxisZPos"/>
                <AxisType DefaultValue="Linear"/>
                <Direction><X DefaultValue="0"/><Y DefaultValue="0"/><Z
DefaultValue="1"/></Direction>
                <SCType ID="Spindle" Caption="Spindle" type="TMachineAxis">
                    <ParameterName DefaultValue="SpindlePos"/>
                    <AxisType DefaultValue="Rotary"/>
                    <SCType ID="Tool" Caption="Tool" type="TMillToolHolder">
                        <SpindleParamID DefaultValue="SpindlePos"/>
                    </SCType>
                </SCType>
            </SCType>
        </SCType>

        <!-- ВЕТКА ЗАГОТОВКИ: стол движется по X (физически -X, см. примечание ниже) -->
        <SCType ID="AxisX" Caption="Axis X" type="TMachineAxis">
            <ParameterName DefaultValue="AxisXPos"/>
            <AxisType DefaultValue="Linear"/>
            <Direction><X DefaultValue="-1"/><Y DefaultValue="0"/><Z
DefaultValue="0"/></Direction>
            <SCType ID="Table" Caption="Table" type="TMachineNode">
                <ImageFile DefaultValue="Images\table.osd"/>
                <SCType ID="Workpiece" Caption="Workpiece" type="TWorkpieceHolderNode"/>
            </SCType>
        </SCType>
    </SCType>
</Schema>
```

Чтение этого дерева: **станина** — фиксированная база (`IsFloor="True"`, см. кинематическую схему). Её **ветка инструмента** — Y → Z → шпиндель → держатель инструмента: подвиньте Y, и всё, что ниже, сдвинется, а листовой `TMillToolHolder` — это место крепления режущего инструмента. Её

ветка заготовки — X → стол → заготовка: ось X несёт стол, а `TWorkpieceHolderNode` на столе — место удержания детали. Эти два листа — *коннектор инструмента* / *коннектор заготовки* интерфейса. Каждый движущийся узел называет управляющий им параметр состояния через `ParameterName`, и все четыре параметра объявлены в `MachineStateParameters`.

> Обратите внимание на `Direction` оси X — **(−1, 0, 0)**. Она находится в ветке **заготовки**, поэтому её `Direction` — это физическое перемещение *стола* — мировой **−X**. Перемещение детали в −X эквивалентно перемещению инструмента в +X, так что получается привычная станочная ось X с правой тройкой осей. (Ось из ветки инструмента, напротив, использует своё направление перемещения напрямую.) Это правило знака для ветки заготовки подробно описано в §4.

Остальная часть главы проходит по каждой секции по очереди.

Идентичность и метаданные

`AbstractMachine` несёт заголовочные члены, нужные каждому станку — `GUID`, `Priority`, `Name`, `Comment`, `Group`, `Image`, `Icon` — и поля идентичности `DescriptionPlaceHolder` (`Developer`, `NCSystem`, постпроцессор `SPPFile`, файл интерпретатора и систему **Measurements**, фиксирующую единицы станка). Производный станок переопределяет их так же, как операция:

```
<SCType ID="AbstractMillMachine" Caption="Abstract Mill Machine" type="AbstractMachine">
    <GUID     DefaultValue="{FA164E91-DF7D-474B-B335-8EE7F2C34E41}"/>
    <Priority DefaultValue="1000"/>
    <Group    DefaultValue="Milling"/>
    ...
</SCType>
```

(реальный пример: `Machines/AbstractMachine.xml`)

`Group` — это перечисление `TMachineGroup`: `Milling`, `Lathe`, `LatheMilling`, `JetCutter`, `WireEDM`, `Robot`, `Unknown`.

Система **Measurements** (метрическая или дюймовая) фиксирует единицы каждого переноса `Matrix` и всей геометрии, на которую есть ссылки — см. Системы координат и единицы.

Параметры управления

`ControlData` — это глубокая секция переключателей, сообщающих движку и постпроцессору, что станок умеет. Обычно вы переопределяете только относящиеся к вашему станку части. **Каждый из этих параметров подробно объяснён — с XML-фрагментом, который его переопределяет — в Справочнике параметров Machine Setup;** этот раздел — лишь обзор того, что там находится:

- **Дуги** — разрешён ли вывод дуг и в каких плоскостях, мин. длина, макс. радиус, пространственные дуги.
- **Поворотные преобразования** — доступность полярной/цилиндрической интерполяции, поддержка TCPM и умолчания.
- **5-осевая компенсация** — компенсация вершины инструмента / нуля заготовки / системы координат для индексных и непрерывных режимов.
- **Локальная СК (ORIGIN)** — как ведут себя начала координат в стиле G54.
- **Вывод системы координат инструмента** — формат поворота/ориентации, ожидаемый постпроцессором (вектор нормали, кватернион, варианты Эйлера, …).

Многие из этих умолчаний вычисляются языком выражений, чтобы оставаться согласованными — например, последовательность поворотов, выводимая из выбранного формата системы координат инструмента через большой `SWITCH` (см. §5).

Параметры состояния станка

`MachineStateParameters` перечисляет величины, которыми управляет ЧПУ. Вы добавляете их, переопределяя унаследованную секцию и объявляя члены `TMachineStateParameter` (или предопределённой специализации):

```
<MachineStateParameters>
    <SCType ID="AxisXPos" type="TAxisXPosition"/>
    <SCType ID="AxisYPos" type="TAxisYPosition"/>
    <SCType ID="AxisZPos" type="TAxisZPosition">
        <InitialValue DefaultValue="100"/>
    </SCType>
    <SCType ID="AxisSPos" Caption="Spindle Position" type="TMachineStateParameter">
        <Address  DefaultValue="S"/>
        <Group    DefaultValue="RotaryAxis"/>
        <Priority DefaultValue="1"/>
        <Order    DefaultValue="4"/>
    </SCType>
</MachineStateParameters>
```

(реальный пример: `Machines/AbstractMachine.xml`)

Чаще всего вы задаёте `Address`, `Group` и `Min`/`Max`/`Incr`; ещё несколько управляют поведением симуляции и смены инструмента (`AxisControl`, `HasBrake`, `SupportShortestPathRotation`, `DesignTimeValue`, `ControlWithMap`, …). **Полная таблица свойств с примечаниями по каждому** — в справочнике узлов → `TMachineStateParameter`.

Два места для объявления параметра состояния

`TMachineStateParameter` может объявляться в **одном из** двух мест:

- в **глобальной секции** `MachineStateParameters` (как выше), либо
- **внутри описания узла или оси** в `Schema`.

Объявление его на узле и включает **переключаемые (необязательные) узлы**: когда узел может присутствовать или отсутствовать, его параметр состояния путешествует вместе с ним. Содержимое `MachineStateParameters` тогда формируется **динамически** — параметры появляются или исчезают в зависимости от того, какие узлы реально загружены для текущей конфигурации станка. Помещайте параметр в глобальную секцию, когда он существует всегда; помещайте на узел, когда он должен появляться и исчезать вместе с этим узлом.

Кинематическая схема

`Schema` — это место, где геометрия станка оживает. Это **дерево узлов, вложенных по принципу вложенности**: дочерний узел сидит на родителе и движется вместе с ним. Вложенность *и есть* кинематическая цепочка. У схемы всегда **две ветви от базы** — одна несёт **инструмент**, другая удерживает **заготовку** — и обе обязательны (см. полный пример выше).

Ключевые моменты при построении схемы:

- Каждый движущийся узел привязывается к параметру состояния через `ParameterName` (шпиндель — это просто поворотная ось, чей параметр состояния в группе `RotaryAxis`, а держатель инструмента указывает на него через `SpindleParamID`). Все параметры, которые называет схема, должны существовать в `MachineStateParameters`.
- `AxisType` — это `Linear` или `Rotary`. **`Direction` применяется только к линейным осям** — там это направление перемещения. Для **поворотных** осей `Direction` *не используется*: вращение всегда вокруг **оси Z собственной локальной системы координат оси**, поэтому ориентируйте узел (через его `Matrix`), чтобы поместить этот Z туда, где должна быть ось вращения.
- **`Scale`** — это **безразмерный** множитель значения параметра состояния оси (длина для линейной оси, угол в **радианах** для поворотной); **отрицательный** `Scale` инвертирует ось. Полный список свойств `TMachineAxis`, включая `RapidFeed`, `Channel` и `DesignTimeAxisValue`, см. в справочнике узлов.
- Цепочка **должна заканчиваться держателем**, чтобы инструменту (а на стороне заготовки — детали) было где разместиться. `XAxisID`/`YAxisID`/`ZAxisID` и `ToolAxisID`/`ToolAxisValue` держателя инструмента несут дополнительный смысл — см. §4 и справочник коннекторов.
- Визуальный `Color`/геометрия каждого узла задаётся через унаследованный `VisualProperties` (см. справочник узлов и 3D-модели станков).

Один параметр, несколько узлов — приём с общей осью

Ось живёт в **двух местах** с разными ролями: **узел оси** (`TMachineAxis`) в `Schema` — *геометрия* (где он сидит, его `Direction`, `Matrix`, 3D-модель) — и **параметр состояния** (`TMachineStateParameter`) —

единственное *управляемое значение* (его `Address`, пределы, режим управления). Они связаны через совпадение `ParameterName` узла с `ID` параметра.

Мощное следствие: **несколько узлов осей могут делить один параметр состояния**, так что одно управляемое значение приводит сразу несколько частей геометрии. Классический случай — **3-кулачковый патрон**: три кулачка, каждый — собственный узел оси, направленный радиально в своём повёрнутом направлении, и все приводятся одним `JawPos`:

```
<!-- один общий параметр состояния -->
<MachineStateParameters>
    <SCType ID="JawPos" Caption="Jaws" type="TMachineStateParameter">
        <AxisControl DefaultValue="Manual"/>
        <Min DefaultValue="0"/><Max DefaultValue="100"/><InitialValue DefaultValue="50"/>
    </SCType>
</MachineStateParameters>

<!-- три узла-кулачка, все управляются ОДНИМ JawPos, но повёрнуты на 0°, 120°, 240° -->
<SCType ID="Chuck" Caption="Chuck" type="TMachineNode">
    <SCType ID="Jaw1" type="TMachineAxis">
        <ParameterName DefaultValue="JawPos"/><AxisType DefaultValue="Linear"/>
        <Direction><X DefaultValue="1"/><Y DefaultValue="0"/><Z
DefaultValue="0"/></Direction>
        <Matrix><SCType ID="T1" type="TRotateZ" DefaultValue="0"/></Matrix>
    </SCType>
    <SCType ID="Jaw2" type="TMachineAxis">
        <ParameterName DefaultValue="JawPos"/><AxisType DefaultValue="Linear"/>
        <Direction><X DefaultValue="1"/><Y DefaultValue="0"/><Z
DefaultValue="0"/></Direction>
        <Matrix><SCType ID="T1" type="TRotateZ" DefaultValue="120"/></Matrix>
    </SCType>
    <SCType ID="Jaw3" type="TMachineAxis">
        <ParameterName DefaultValue="JawPos"/><AxisType DefaultValue="Linear"/>
        <Direction><X DefaultValue="1"/><Y DefaultValue="0"/><Z
DefaultValue="0"/></Direction>
        <Matrix><SCType ID="T1" type="TRotateZ" DefaultValue="240"/></Matrix>
    </SCType>
</SCType>
```

Каждый кулачок перемещается вдоль **своего локального X** (своего `Direction`), но его `Matrix` поворачивает эту локальную систему на 0° / 120° / 240°, так что три кулачка радиально сходятся и расходятся **вместе** при каждом изменении единственного значения `JawPos`. Один параметр — три движущихся узла.

Привязка схемы к полу — `IsFloor`

Пометьте узел, представляющий **пол** (куда направлена гравитация), логическим атрибутом `IsFloor="True"` (как на узле `Base` в примере выше). Он не влияет на кинематику; он сообщает графическому окну, как строить **стандартные виды**, чтобы «верх» выглядел естественно для пользователя, даже когда мировой Z станка не направлен вверх — например, у **потолочного робота**, базовый Z которого направлен вниз.

Частые ошибки схемы

При загрузке схемы система проверяет её и сообщает о проблемах. Сообщения, которые вы вероятнее всего встретите:

- **"Axes (X,Y,Z) must define the right coordinate system (repaired)"** — выбранные направления перемещения осей не образуют **правую** тройку. Система исправляет это, но проверьте векторы `Direction` и знаки веток инструмента/заготовки (§4).
- **`Tool connector "…" has no correct X/Y/Z axis`** — `XAxisID` / `YAxisID` / `ZAxisID` коннектора инструмента не называют существующие оси.
- **"Possibly incorrect ToolAxisID …"** — `ToolAxisID` инструмента или субстанка не указывает на корректную ось (обычно это должна быть ось шпинделя или револьверной головы).
- **"The same machine axis is specified as a X, Y or Z axis more than once in the submachine"** — субстанок указывает одну ось в двух из своих слотов `XAxisID`/`YAxisID`/`ZAxisID`.
- **"Incorrect redundant robot axis type"** — неподдерживаемое значение в `RedundantAxis`.

> **Продвинутые темы схемы** — как позиционировать узлы **матрицами** (относительно родителя или мира), строить **переключаемые/съёмные узлы** (блоки револьверной головы, присоединяемые головки), создавать **револьверные головы**, подключать **коннекторы инструмента/заготовки** и настраивать **каналы управления и субстанки** для многоканальных станков — описаны в **Продвинутых темах по станкам**.

Оснастка, СОЖ и симуляция

Остальные секции дескриптора меньше и в основном самоочевидны:

- **`Coolants`** — каналы СОЖ, доступные на станке, каждый с кодом, который постпроцессор выводит для его включения/выключения.
- **`Simulation`** — метод и разрешение симуляции, а также настройки столкновений, используемые при проверке проекта на этом станке.
- **`ToolChange`** — позиция смены инструмента, закон(ы) тайминга (на которые ссылается `ToolChangeTimeCalcLawIndex` коннектора) и режим вывода смен инструмента.

Размеры станка

`MachineDimensions` — место, где собираются важные управляющие параметры схемы, чтобы пользователь видел их все в одном компактном месте — например, расположение позиционера, размер ячейки или любое значение, от которого параметризованы другие узлы. Параметр попадает в эту секцию одним из двух способов: объявляется **напрямую** как её дочерний (для значений, принадлежащих базовым узлам) или **прикрепляется** к ней атрибутом переподчинения `Parent` (для значений, живущих на переключаемых/присоединяемых узлах, но которые всё равно должны появляться здесь). О `Parent` см. §8.4.

Обнаружение

Станки **нигде** не регистрируются — они обнаруживаются по расположению. Файл станка находится просто за счёт размещения в одной из сканируемых папок станков (`$(SCHEMAS_FOLDER)` / `$(CUSTOM_SCHEMAS_FOLDER)`); библиотека станков подхватывает его при сканировании папок и загружает по требованию (выше). Регистрационную запись писать не нужно — в отличие от операций, станкам не нужна запись в регистраторе.

Чек-лист для нового станка

1. Определите ближайшую базу (`AbstractMillMachine`, абстрактный робот/токарный, ...) или напрямую `AbstractMachine`.
2. Сгенерируйте `GUID`; задайте `Group`, `Name`, `Comment`, `Image`, `Icon`, постпроцессор `SPPFile`.
3. Определите управляемые величины в `MachineStateParameters`.
4. Постройте дерево `Schema`: вложите оси в их реальном кинематическом порядке и завершите каждую ветвь подходящим держателем; свяжите каждую ось с её параметром состояния через `ParameterName`.
5. Установите нужные переключатели `ControlData` (дуги, поворотные преобразования, 5-осевое, вывод системы координат инструмента).
6. Поместите файл в сканируемую папку станков (`$(SCHEMAS_FOLDER)` / `$(CUSTOM_SCHEMAS_FOLDER)`), а не в `Supplement` — библиотека обнаружит его автоматически, регистрационная запись не нужна; `ID` членов должны быть уникальны лишь в пределах этого одного файла (выше).

Далее: **Использование XML-свойств из CAM API**

Справочник узлов станка

Эта страница — **справочник свойств** для каждого типа узла, используемого в кинематической схеме станка, а также системы координат и единицы, в которых они все выражены. Это справочник-компаньон к **Дескрипторам станков**, где разбирается, как узлы сочетаются друг с другом; приходите сюда, когда нужен полный список свойств конкретного типа узла.

Типы узлов образуют небольшое дерево наследования:

- `TMachineNode` — база всего в схеме. Каждый другой тип узла ниже наследует его `ID`, `Matrix`, `VisualProperties` и `ImageFile`.
  - `TMachineAxis` — узел, который *движется* (линейно или поворотно).
  - `TToolHolderNode` / `TWorkpieceHolderNode` — гнёзда, где крепятся инструмент и заготовка (*коннектор инструмента* / *коннектор заготовки* интерфейса).

Отдельный тип, `TMachineStateParameter`, вообще не является узлом схемы — это управляемое *значение*, которым приводится узел оси. Узлы осей ссылаются на него по имени.

Системы координат и единицы измерения

- **Мировая система координат.** Если смотреть на обычный фрезерный станок спереди: **Z вверх, X вправо, Y от наблюдателя**. Стройте схему (и моделируйте геометрию) в этой системе.
- **Единицы.** Все переносы `Matrix` — и вся CAD-геометрия — заданы в единицах *Measurements* станка (см. Идентичность и метаданные): **миллиметры** для метрического станка, **дюймы** для дюймового. Если активная система единиц отличается, система пытается пересчитать масштаб станка **один раз при загрузке**, но для сильно параметризованных схем это может сработать некорректно — поэтому **готовьте схему сразу в тех единицах, в которых она будет использоваться**.
- `Direction` **не обязан быть единичным вектором.** Для обычной ортогональной оси это нормализованный вектор перемещения. Но на станках с **неортогональными** осями — например, наклонная опора Y на токарно-фрезерном — модуль `Direction` **не** равен 1; его компоненты следуют синусам/косинусам наклона между осями. Поставляемый токарно-фрезерный *Mori Seiki* (`Machines\LatheMilling\Turret\Mori Seiki\MoriSeiki.xml`) задаёт своей оси Y `Direction = (√3, 1, 0)` — записанное точно как `(1.7320508…, 1, 0)`, наклон 30° с модулем 2 — для математически точного описания.
- **Конвенция оси инструмента. Z** инструмента идёт от режущей вершины вдоль оси вращения инструмента к точке его крепления в шпинделе/держателе. Для наружных (OD) токарных резцов то же по **ISO 13399**: Z от вершины к креплению держателя, рабочая плоскость пластины в плоскости **ZX**; **X** выбран так, чтобы у **правого** инструмента бóльшая часть режущей геометрии была со стороны **−X** (у **левого** — на **+X**). CAM-система всё вычисляет в

предположении этого размещения; лишь при финальном выводе CLData необязательный `AdditionalTransform` (вывод системы координат инструмента) может применить последнее смещение/поворот — обычно нулевое для станков и используемое на роботах, чтобы развернуть Z так, чтобы он указывал из фланца (и из инструмента в сторону, противоположную креплению).

TMachineNode

`TMachineNode` — база для всего в дереве схемы. Каждый тип узла наследует эти четыре члена:

Свойство	По умолч.	Смысл
`ID`	`""`	Идентификатор узла, уникальный в пределах этого файла станка.
`Matrix`	единичная	Размещение узла относительно родителя (или мира — см. позиционирование узлов матрицами).
`VisualProperties`	—	Как отрисовывается 3D-модель узла (см. ниже).
`ImageFile`	`""`	3D-модель узла, файл `.osd` / `.stl` (см. 3D-модели станков).

Голый `TMachineNode` — это **жёсткая** часть конструкции (станина, колонна, стол): он несёт геометрию и позиционирует своих потомков, но сам не движется.

TMachineAxis

`TMachineAxis` расширяет `TMachineNode` (поэтому у него есть `ID`, `Matrix`, `VisualProperties` и `ImageFile`, как у любого узла) и добавляет специфичные для оси свойства ниже:

Свойство	По умолч.	Смысл
`AxisType`	`Linear`	`Linear` или `Rotary`.
`ParameterName`	`""`	ID `TMachineStateParameter`, который приводит эту ось.
`Direction`	`(0, 0, 1)`	Направление перемещения (**только линейные оси** — см. примечание ниже).
`Scale`	`1`	Множитель от значения параметра состояния к физическому перемещению (см. ниже); отрицательный инвертирует

Свойство	По умолч.	Смысл
		направление.
`RapidFeed`	`10000`	Скорость ускоренного перемещения в минуту (мм/мин для линейной оси в мм, град/мин для поворотной оси в градусах).
`Channel`	`-1`	Канал управления (§6); `-1` = общий.
`WorkpieceIndex`	`-1`	Если ≥ 0, ось управляется только для этой заготовки.
`DesignTimeAxisValue`	`0`	Значение оси, используемое во время проектирования для вычисления мировых матриц узлов.

`Direction` — линейная или поворотная. Для **линейной** оси `Direction` — это направление перемещения. Для **поворотной** оси `Direction` *не используется*: вращение всегда вокруг **оси Z собственной локальной системы координат оси**, поэтому ориентируйте узел (через его `Matrix`), чтобы поместить этот Z туда, где должна быть ось вращения.

`Scale` — это **безразмерный** множитель, применяемый к значению параметра состояния оси — физическая единица заключена в *значении*, а не в `Scale`: длина для линейной оси (мм или дюйм в дюймовой системе) и угол в **радианах** для поворотной оси. Используйте `Scale`, чтобы переотобразить значение — например, **индексная** револьверная голова хранит простой *индекс* позиции, а `Scale`, равный `360 / BlocksCount`, превращает каждый шаг индекса в поворот на одну позицию. **Отрицательный** `Scale` инвертирует направление оси (а значит, переворачивает и базовую СК, в которую он вносит вклад — см. §4).

`VisualProperties`

Каждый узел (`TMachineNode` и его потомки — оси, держатели, узлы схемы) имеет блок `VisualProperties`, управляющий тем, как отрисовывается его 3D-модель:

Свойство	По умолч.	Смысл
`Visible`	`True`	Показывается ли модель узла; при выключении узел скрыт.
`Color`	`(0.5, 0.5, 0.5)`	Цвет узла как `R` / `G` / `B`, каждый `0…1`.
`Metallic`	`False`	Отрисовка с металлическим видом.
`IsTransparent`	`False`	Сделать узел прозрачным (включает `Transparence`).

Свойство	По умолч.	Смысл
`Transparence`	`0.5`	Прозрачность, `0` = непрозрачно … `1` = полностью прозрачно.
`DisplayMode`	`Default`	Режим отрисовки — `Wire`, `Shade`, `EdgeShade` или `Default`.
`VisMatrix`	единичная	Матрица, размещающая 3D-модель узла независимо от его кинематической `Matrix` (см. §1).
`Use3DModelColors`	`False`	При `True` отрисовывать узел цветами, хранящимися в его 3D-модели, вместо `Color` выше. (Атрибут на `VisualProperties`.)

```
<VisualProperties Use3DModelColors="False">  <!-- атрибут на самом элементе -->
    <Visible DefaultValue="True"/>
    <Color><R DefaultValue="0.5"/><G DefaultValue="1"/><B DefaultValue="0.5"/></Color>
    <DisplayMode DefaultValue="Shade"/>
</VisualProperties>
```

> Подготовка самих 3D-моделей (файлы `.osd` / `.stl`, где они лежат, единицы и выравнивание) описана в **3D-моделях станков**.

TMachineStateParameter

`TMachineStateParameter` — это управляемая величина, которой управляет ЧПУ: положение оси, шпиндель, ход кулачка. Узлы осей ссылаются на него через `ParameterName`. Он объявляется либо в глобальной секции `MachineStateParameters`, либо на узле в `Schema` (см. параметры состояния станка).

Свойство	По умолч.	Смысл
`Enabled`	`True`	Активен ли параметр.
`ID`	`""`	Его идентификатор; оси и держатели ссылаются на него по этому имени (часто параметризованному, например `[Prefix]…`).
`Address`	`""`	Адресная буква, которую постпроцессор выводит для него (например, `X`, `Z`, `C`, `S`).
`AxisControl`	`Continues`	Как он приводится — `Continues` (непрерывно), `Indexed` или `Manual` (см. примечания).

Свойство	По умолч.	Смысл
`Group`	`Other`	Классификация — `LinearAxis`, `RotaryAxis` или `Other`.
`Incr`	`0.001`	Наименьший инкремент / разрешение.
`Min` / `Max`	`-1E9` / `+1E9`	Пределы перемещения.
`InitialValue`	`0`	Значение оси при сбросе станка в исходное состояние в проекте, до запуска любой операции.
`DesignTimeValue`	`0`	Смещение нуля оси — положение, в котором была нарисована её 3D-модель (см. примечания).
`HasBrake`	`False`	У оси есть тормоз. При установке вокруг каждого изменения этой оси в CLData выдаётся команда `AXESBRAKE On|Off` (чтобы постпроцессор отпустил/зажал тормоз).
`SupportShortestPathRotation`	*вычисляется*	Только периодические поворотные оси — влияет на симуляцию ускоренных перемещений (см. примечания).
`Priority`	`0`	Порядок параметра только в списках/окнах пользователя (на вывод не влияет).
`Order`	`[Priority]`	Порядок, в котором оси записываются в команды CLData.
`ParameterType`	`Geom`	**Устарело** — сейчас не действует (когда-то группировало отображение на геометрические и технологические).
`ControlWithMap`	`False`	Разрешение избыточной оси через интерактивную *карту оси* (см. примечания).
`ClampID`	*вычисляется*	Id зажимного приспособления; на него ссылаются операции передачи/перехвата детали, которые зажимают и разжимают его.

Чаще всего вы задаёте `Address`, `Group` и `Min`/`Max`/`Incr`.

Примечания по отдельным свойствам:

- `AxisControl` — *Continuous*: несколько непрерывных осей могут меняться в пределах одного кадра УП. *Indexed*: не может двигаться синхронно с другими осями — позиционируется и фиксируется в собственном кадре(ах) УП до основной обработки. *Manual*: не управляется программой — оператор задаёт её вручную, поэтому она может меняться только в объектах **Setup** (не в операциях внутри установа), а постпроцессор может выводить её как комментарий/инструкцию оператору.
- `ControlWithMap` — для **избыточных** осей. Включает интерактивную *карту оси*: график значения избыточной оси (по вертикали) против длины пути (по горизонтали), на котором вы рисуете, как эта ось должна двигаться вдоль траектории, наблюдая за движением всего станка и видя столкновения узлов как запретные зоны на карте. Это интерактивный способ разрешить избыточность обратной кинематики. Включайте его **только для *дополнительных* избыточных осей, о которых виртуальный субстанок ещё не знает**: основные избыточные внешние оси — например, 3 линейные оси рельсов робота и 2 поворотные оси позиционера, или ось C 5-осевого станка (для обхода сингулярностей) — появляются на карте **автоматически**, без этого флага.
- `SupportShortestPathRotation` — для периодических поворотных осей с диапазоном более 180°. Он **не** меняет CLData (целевое значение, например `A+370`, по-прежнему выводится); он влияет только на **симуляцию ускоренных перемещений**: при включении ускоренное перемещение идёт по кратчайшему эквивалентному углу (`A+10` вместо `A+370`); при выключении делает полный оборот. Рабочие перемещения всегда выполняют полный ход.
- `DesignTimeValue` — используйте, когда 3D-модель оси была нарисована в ненулевом положении: укажите здесь это положение, и нуль оси сместится на противоположную величину, чтобы модель совпала со значением оси 0.

Коннекторы инструмента и заготовки

Коннектор — это место, где инструмент или заготовка крепятся к кинематической цепочке — *коннектор инструмента* (`TToolHolderNode`) и *коннектор заготовки* (`TWorkpieceHolderNode`) интерфейса. Оба наследуются от общего гнезда и разделяют свойства **шпинделя и зажима** ниже. На **станках** (не роботах) **инструментальный** коннектор дополнительно несёт свойства, сообщающие решателю кинематики, как позиционируется и выбирается его инструмент.

Таблицы здесь — это справочник свойств; *поведение* за этими свойствами — как выводится базовая система координат, как коннектор делается активным, оговорка о размещении заготовки — объяснено в §4 Коннекторы инструмента.

Свойства, специфичные для инструмента (`TToolHolderNode`)

Свойство	По умолч.	Смысл
`XAxisID` / `YAxisID` / `ZAxisID`	`""`	Оси станка, позиционирующие вершину этого инструмента в пространстве; они также определяют **базовую систему координат** коннектора (§4).
`ToolAxisID`	`""`	Ось, которую нужно переместить, чтобы сделать этот коннектор активным (индексация магазина / револьверной головы). Может быть пустой, когда активный субстанок предоставляет ось инструмента.
`ToolAxisValue`	`0`	Значение `ToolAxisID`, которое вводит эту позицию / инструмент.
`ToolNumber`	`0`	Позиция в магазине (номер инструмента), которую выводит постпроцессор. `0` → не используется; вместо этого УП выводит номер из собственных свойств режущего инструмента.
`SupportedToolTypes`	—	Применения, которые может выполнять инструмент в этом коннекторе — `MillTool`, `LatheCutter`, `JetCutter`, `Punch`, `Wire`, `Cutter6D`, `Welder`, `AdditiveTool`, `Painter`, `HeatTreatment`, `Gripper`. Используется для фильтрации списков операций/инструментов, предлагаемых для коннектора.
`Channel`	`-1`	Канал управления, к которому принадлежит этот коннектор (§6).
`ToolChangeTimeCalcLawIndex`	`0`	Индекс закона времени смены инструмента (заданного в секции *Tool Change* станка), применяемого к этому коннектору (`0` = закон по умолчанию).

Шпиндель и зажим (общие для обоих коннекторов)

Свойство	По умолч.	Смысл
`SpindleParamID`	`""`	ID параметра состояния станка, приводящего поворотную ось **шпинделя**, которая вращает инструмент или деталь. Это ID *параметра состояния*, **а не** `ID` узла оси.

Свойство	По умолч.	Смысл
`HolderType`	`Unknown`	`Unknown`, `LeftLatheSpindle` или `RightLatheSpindle`; помечает гнездо токарного шпинделя.
`DefaultClampID`	*вычисляется*	ID зажима по умолчанию для этого гнезда, на который ссылаются операции передачи/перехвата детали; по умолчанию выводится из `HolderType` (левый шпиндель → `1`, правый → `2`, иначе `-1`).

Коннектор заготовки (`TWorkpieceHolderNode`)

Коннектор заготовки — это место удержания **детали** — на патроне, шпинделе или приспособлении. Он несёт общие свойства *шпинделя и зажима* выше (важнее всего `SpindleParamID` для токарного шпинделя и `HolderType` / `DefaultClampID` для зажима), но **ни одного** из полей позиционирования инструмента (`XAxisID`/`YAxisID`/`ZAxisID`), активации (`ToolAxisID`/`ToolAxisValue`) или `ToolNumber` / `SupportedToolTypes` — те описывают, как позиционируется и выбирается *инструмент*, что неприменимо к держателю детали.

> **Оговорка о размещении.** Хотя `TWorkpieceHolderNode` наследует `Matrix` узла, эта матрица **игнорируется** для коннектора заготовки — в отличие от обычных узлов, осей и держателей инструмента. Коннектор заготовки всегда принимает размещение своего **родительского узла**. Поэтому, чтобы сместить положение детали, добавьте дополнительный родительский узел и поместите размещение (его `Matrix`) на *этот* узел. Это сделано намеренно: положение детали должно определяться **установом заготовки** в проекте (внутри CAM-системы), а не фиксироваться при создании станка.

Назад к разбору: **Дескрипторы станков** · Указатель станков

Справочник параметров Machine Setup

Эта страница объясняет параметры, показываемые на вкладке **Machine Setup** станка — секции *Description*, *Control Parameters* и смежные — в том виде, как вы видите их в инспекторе параметров. Для каждого параметра описано **что он делает и когда его менять**, и приведён **XML-фрагмент**, переопределяющий его значение по умолчанию в дескрипторе станка.

Эти параметры — обращённая к человеку сторона секции `ControlData` дескриптора станка и соседних секций (XML-структура описана в Дескрипторах станков). Построители станков обычно задают их инструментом **MachineMaker**, но понимать их необходимо при тонкой настройке станка или написании постпроцессора/интерпретатора, потому что они определяют, как траектория превращается в УП и как станок симулируется.

> **Станок vs. робот.** Некоторые параметры и целые группы различаются по типу кинематики — у 5-осевого фрезерного станка иные опции, чем у 6-осевого шарнирного робота. Где это важно, различия отмечены ниже.

> **Откуда берутся значения.** У каждого параметра здесь задаётся **значение по умолчанию** уровня станка; многие из них просто «засевают» соответствующую опцию, которую затем оператор видит (и может изменить) внутри операции. В тексте отмечается, когда значение Machine Setup — это «значение по умолчанию для свойства операции».

Как переопределить параметр в XML

Каждый фрагмент ниже помещается **внутрь тела типа вашего станка**, переопределяя значение, унаследованное от базового станка, формой переопределения из §4.2 — вы заново указываете член (и путь секции, который его содержит) и задаёте новый `DefaultValue`:

```
<SCType ID="MyMachine" Caption="My 5-axis mill" type="AbstractMillMachine">
    <ControlData>
        <LocalCS>
            <AutoLCSRotationLaw DefaultValue="SnapToToolCS"/>
            <!-- SnapToMachineCS, SnapToWorkpieceCS, ExcludeA, ExcludeB, ExcludeC -->
        </LocalCS>
    </ControlData>
</SCType>
```

Фрагменты, показанные для каждой группы ниже, для краткости опускают внешний `<SCType …>` — они начинаются с нужной секции (`<ControlData>`, `<ToolChange>`, …). **Вложенность секций важна**: член должен быть обёрнут в те же элементы, что его содержат. Комментарии после значения перечисляют другие принимаемые варианты перечисления. Почти каждый параметр объявлен в

MachineTypes.xml и Machines/AbstractMachine.xml; полные готовые станки поставляются в $(SCHEMAS_FOLDER) и их стоит изучать как примеры.

Описание (Description)

Общая, в основном информационная идентичность станка:

- **Name**, **Group** — отображаемое имя и категория станка, к которой он относится.
- **Developer** — кто создал станок/схему.
- **NC System name** — семейство стойки/ЧПУ, для информации.
- **Postprocessor file** — постпроцессор по умолчанию, используемый для генерации УП для этого станка.
- **Interpreter file** — используется для обратного чтения УП и восстановления траектории внутри станка.
- **Measurements** — система единиц, в которой создавался станок (мм / дюймы), и производные единицы (линейные, скорость резания, подача, подача/об, подача/зуб, обороты). На вкладке они только для просмотра.

```
<!-- тело типа станка (прямые потомки) -->
<Name      DefaultValue="My 5-axis mill"/>
<Group     DefaultValue="Milling"/>  <!-- Unknown, Lathe, LatheMilling, JetCutter, WireEDM,
Robot -->
<Developer DefaultValue="Acme"/>
<NCSystem  DefaultValue="Heidenhain TNC640"/>
<SPPFile   DefaultValue="$(PROGRAM_PERSONAL)\Postprocessors\MyPost.sppx"/>
<Image     DefaultValue="Images\MyMachine.png"/>
<Icon      DefaultValue="Images\MyMachine_ico.bmp"/>
```

Оснастка (Tooling)

Для станков с револьверными головами здесь настраивается компоновка блоков револьверной головы.

Параметры управления (Control Parameters)

Основная часть вкладки. Они сообщают движку и постпроцессору, что умеет стойка и как должен формироваться вывод. Всё в этой группе живёт под элементом <ControlData>.

Дуги

Управляет тем, как круговые перемещения выводятся в УП.

- **Use arcs** — главный переключатель. Если выключен, каждая дуга, которая появилась бы в траектории, заменяется отрезками, вычисленными по допуску, заданному в операции.
- **Use arcs in XY / YZ / ZX plane** — ограничить вывод дуг плоскостями, которые стойка действительно поддерживает; дуги в отключённой плоскости заменяются отрезками.
- **Circles division** — разбиваются ли полные окружности: *Do not break*, *Into quadrants* (90°) или *Into halves* (180°).
- **Minimal Arc Length** / **Maximal Arc Radius** — защита от вырожденных дуг. Дуга короче минимума или с радиусом больше максимума (почти прямая) выводится отрезками.
- **Spatial arcs** — дуги, которые **не** лежат в ортогональной плоскости XY/YZ/ZX, обычно заданные тремя произвольными точками. Часто встречаются на промышленных роботах. Если выключено, такие дуги становятся отрезками; если включено, формируется специальная многоточечная команда дуги для постпроцессора.
  - **Supported** — включает вывод пространственных дуг.
  - **Minimal distance between start and end points** — дугу по 3 точкам нельзя задать, когда начало и конец совпадают (полная дуга 360°). Если расстояние начало/конец меньше этого значения, вместо неё выводятся отрезки.
  - **Reorientation mode** — как моделируется *ориентация инструмента* при прохождении пространственной дуги: *Relative to base CS* (сферическая интерполяция ориентации в базовой системе координат дуги) или *Relative to path* (ориентация интерполируется относительно касательной к пути).
  - **Consider middle point orientation** — дуга по 3 точкам несёт положение *и* ориентацию в начале, середине и конце. Выкл.: ориентация середины игнорируется, и ориентация интерполируется начало→конец (средняя точка всё равно задаёт форму дуги в пространстве). Вкл.: ориентация интерполируется начало→середина, затем середина→конец. Установите в соответствии с тем, как ведёт себя реальная стойка.

```
<ControlData>
    <UseArc           DefaultValue="true"/>
    <UseArcInXY       DefaultValue="true"/>
    <UseArcInYZ       DefaultValue="false"/>
    <UseArcInZX       DefaultValue="false"/>
    <CirclesDivision DefaultValue="DoNotBreak"/>  <!-- Quadrants, Halves -->
    <MinArcLength     DefaultValue="0.05"/>
    <MaxArcRadius     DefaultValue="10000"/>
    <SpatialArcs>
        <Supported            DefaultValue="False"/>
        <MinDistance          DefaultValue="0.2"/>
        <ReorientationMode    DefaultValue="rmBaseCS"/>  <!-- rmPath -->
        <ConsiderMiddleOrient DefaultValue="False"/>
    </SpatialArcs>
</ControlData>
```

Сингулярности

Сингулярность — это поза, в которой значения суставов математически не определены или неоднозначны. Содержимое этой группы зависит от текущей кинематики.

- **Станок (поворотный стол).** В схеме с поворотным столом, когда ось инструмента становится вертикальной, поворот стола не определён (бесконечно много решений). Таких зон следует избегать.
  - **Allowed axis deviation** (например, *Wrist angle*) — насколько инструмент может быть наклонён от идеальной ориентации, чтобы обойти зону сингулярности, а не проходить сквозь неё.
- **Робот.** У робота больше сингулярностей (кисть, база/плечо, локоть), потому что у него больше суставов. Значения углов по суставам задают, **где начинается каждая зона сингулярности** (угол, с которого сустав считается «в» зоне), а *J2–J3 free area* / карты описывают запретные области, используемые для обхода.

```
<ControlData>
    <Singularities>
        <Wrist DefaultValue="0.01"/>   <!-- допустимое отклонение оси, градусы -->
        <!-- только робот, дополнительные суставы: -->
        <Elbow DefaultValue="0"/>
        <Base><Angle DefaultValue="5"/></Base>
    </Singularities>
</ControlData>
```

Переворот стола / конфигурация станка по умолчанию

Когда поза достижима более чем одним способом (например, наклонить голову влево или вправо — то же соотношение инструмент/деталь, противоположные знаки осей), **переворот** выбирает, какое решение использовать. Machine Setup хранит только **состояние по умолчанию** каждого переворота; сама операция предоставляет фактические органы управления переворотом.

- **Станки:** один переворот *Flip Table* по умолчанию.
- **Роботы:** состояние по умолчанию для каждого переворота сустава — *Flip base (J1)*, *Flip elbow (J3)*, *Flip wrist (J5)*, плюс перевороты/позиции внешних осей.

```
<ControlData>
    <DefaultFlips>
        <Flip5x DefaultValue="False"/>   <!-- станок: переворот стола/кисти по умолчанию
(Boolean) -->
        <!-- только робот: FlipA1, FlipA3, FlipA5, ... -->
```

```
    </DefaultFlips>
</ControlData>
```

Параметры списка начал координат

Как нумеруются рабочие системы координат.

- **Origin prefix** (например, `G`), **Initial number** (например, `54`), **Increment value** (например, `1`) — дают `G54`, `G55`, `G56`, … по мере создания новых начал координат. Роботы используют иные соглашения.
- **Create new origin for part group** — получает ли каждая группа деталей собственное начало координат.

```
<ControlData>
    <OriginParams>
        <Prefix                 DefaultValue="G"/>
        <InitialNum             DefaultValue="54"/>
        <IncStep                DefaultValue="1"/>
        <CreateNewOriginForPart DefaultValue="True"/>
    </OriginParams>
</ControlData>
```

Параметры коррекции на радиус

- **Sharp Corner** — угловой порог (например, `135°`), разделяющий *острые* и *тупые* углы при построении пути с коррекцией на радиус инструмента. На остром угле вставляется дополнительная хорда, чтобы инструмент не уходил слишком далеко от угла; на тупом угле путь просто идёт к точке пересечения.

```
<ControlData>
    <RadiusCompensationParams>
        <SharpCorner DefaultValue="135"/>   <!-- градусы -->
    </RadiusCompensationParams>
</ControlData>
```

Поворотные преобразования

Поддержка полярной и цилиндрической интерполяции.

- **Polar interpolation is available** / **Cylindrical interpolation is available** — предлагает ли стойка эти интерполяции вообще.
- **CNC supports polar interpolation** / **CNC supports cylindrical interpolation** — если **включено**, стойка сама выполняет преобразование, и CAM-система работает как в обычных декартовых

координатах; если **выключено**, CAM-система *симулирует* интерполяцию и выводит получившиеся значения поворотных осей, чтобы стойка им следовала.
- **Start from C=0** — совместимость с устаревшими постпроцессорами, требовавшими сброса оси C в ноль перед включением интерполяции. На новых станках держите **выключенным**; путь теперь отслеживается корректно и без этого.

```
<ControlData>
    <RotaryTrans>
        <PolarAvailable     DefaultValue="true"/>
        <CNCSupPolar        DefaultValue="true"/>
        <CylindAvailable    DefaultValue="true"/>
        <CNCSupCylind       DefaultValue="true"/>
        <OldPPCompatibility DefaultValue="false"/>  <!-- «Start from C=0»; держите false -->
    </RotaryTrans>
</ControlData>
```

Вывод системы координат инструмента

Как **ориентация инструмента** записывается в каждом кадре УП (в основном для 5-осевого / multi-goto вывода).

- **Format** (комбобокс) — формат ориентации: *Normal vector (Nx, Ny, Nz)* (по умолчанию для станков), различные последовательности *углов Эйлера* (часто на роботах), *Quaternion* или *Axis-angle*. Выберите то, что ожидает стойка.
- **Additional transformation** — дополнительный поворот собственной системы координат инструмента, в основном для роботов. По умолчанию Z инструмента направлен от вершины вверх к шпинделю; некоторые роботы калибруются с Z, направленным в другую сторону, или с X вдоль инструмента. Три угла позволяют согласовать с реальной системой координат инструмента.

> **Известное ограничение — прочитайте перед использованием на роботах.** Сейчас существует **один глобальный** `AdditionalTransform`, и он применяется к **двум разным системам одновременно**: к финальной системе координат *инструмента* **и** к системе координат *фланца 6-й оси* робота. На реальных ячейках эти две **не** всегда совпадают, а в ячейке может быть **несколько** рабочих органов, тогда как эта настройка остаётся единственной и глобальной — поэтому одно преобразование не может скорректировать обе независимо. Это известное архитектурное ограничение (в будущей версии настройку, как ожидается, разделят между схемой станка и режущим инструментом). Если ориентация инструмента или фланца робота выглядит неверно и один `AdditionalTransform` не может удовлетворить обе — вот почему; не отлаживайте это снова и снова.

- **Set machine state flags** — выводить состояние системы координат станка (ноль / путь / база робота) в каждом кадре. Выкл. для обычных станков (не используется); обычно вкл. для роботов, где ориентация в каждой точке критична.
- **Swap Tool and Workpiece frames** — для роботов, которые держат **деталь**, пока инструмент неподвижен. Тогда каждый кадр выражает ориентацию детали относительно инструмента (обратную к обычной матрице «инструмент относительно детали»). Если стойка сама выполняет это обращение, оставьте выключенным; иначе CAM-система сама обращает матрицу перед передачей ориентации постпроцессору.
- **Set tool contact surface normal vectors** — для **5-осевой компенсации инструмента**. На чистовых проходах известны точка контакта и её нормаль к поверхности; вывод этой нормали позволяет стойке сместить инструмент вдоль неё на разницу между реальным и запрограммированным инструментом (чтобы слегка иной инструмент не зарезал). При включении в CL data в каждом кадре с контактом поверхности записывается специальная команда **`TLCONTACT`**, несущая эту нормаль контакта; постпроцессор должен её обрабатывать (см. команду `TLCONTACT` в руководстве по вашему постпроцессору).

```
<ControlData>
    <ToolFrameOutput>
        <Format DefaultValue="NormalVector"/>
            <!-- EulerZYX, EulerXYZ, FixedABC, Quaternion, AxisAngleDeg, AxisAngleRad, ... -
->
        <SetMachineStateFlags       DefaultValue="False"/>
        <SwapToolAndWorkpieceFrames DefaultValue="False"/>
        <SetContactNormal           DefaultValue="False"/>
        <AdditionalTransform>           <!-- доп. поворот системы координат инструмента
(роботы) -->
            <A DefaultValue="0"/>
            <B DefaultValue="0"/>
            <C DefaultValue="0"/>
        </AdditionalTransform>
    </ToolFrameOutput>
</ControlData>
```

Избыточная ось

Выбирает правило разрешения **избыточной** оси при решении обратной кинематики (используется только схемами, у которых действительно есть избыточная ось). По умолчанию *None*.

```
<ControlData>
    <RedundantAxis DefaultValue="None"/>
```

```
        <!-- KukaLBRiiwa, NITIProgress, NITIProgress3, NonSphericalWrist, ScaraSimple -->
</ControlData>
```

Связь суставов робота

(Схемы роботов.) Флаги вроде **Joint 3 is linked with joint 2 (parallelogram)** моделируют механически связанные суставы — включены по умолчанию для роботов параллелограммного типа, выключены для прочих. При установке значения связанных суставов выводятся как связанные.

```
<ControlData>
    <IsParallelJoints23 DefaultValue="false"/>  <!-- сустав 3 связан с суставом 2 -->
    <IsConnected46Joints DefaultValue="False"/>
    <IsConnected56Joints DefaultValue="False"/>
</ControlData>
```

Индексное и непрерывное 5-осевое управление

Две почти одинаковые группы, **Indexed 5-axis machining** и **Continuous 5-axis machining**, каждая содержит *те же три переключателя компенсации*. Какую группу использует операция, выбирается переключателем операции **Tool Center Point Management (TCPM)**: TCPM **выкл** → операция использует *индексную* группу; TCPM **вкл** → использует *непрерывную* группу.

Три переключателя компенсации

Они управляют тем, как вершина инструмента и система координат заготовки пересчитываются по мере движения инструмента и детали при 5-осевой обработке:

- **5 Axis tooling point compensation** — когда **включено**, вершина инструмента жёстко привязана к инструменту (стойка пересчитывает положение вершины при каждом повороте). Когда **выключено**, запрограммированная точка остаётся неподвижной, пока инструмент поворачивается — используется в редком случае, когда кинематика поворотной головы стойке *не* известна (например, ручная голова); тогда CAM-система вычисляет путь сама. Обычно оставляйте включённым.
- **5 Axis workpiece zero point compensation** — движется ли **начало** системы координат заготовки вместе с деталью при повороте осей.
- **5 Axis coordinate system compensation** — поворачивается ли **ориентация осей** системы координат заготовки вместе с деталью.

Значения по умолчанию различаются по группам: *индексная* = компенсация вершины вкл, компенсация заготовки выкл; *непрерывная* = все три вкл (в TCPM стойка компенсирует всю

кинематику, поэтому путь программируется так, как если бы система была жёстко привязана к детали).

```
<ControlData>
    <!-- используется, когда TCPM операции ВЫКЛ (индексный): -->
    <Indexed5AxisCompensationMode>
        <IsToolTipCompensated        DefaultValue="true"/>
        <IsWorkpieceZeroCompensated  DefaultValue="false"/>
        <IsCoordinateAxesCompensated DefaultValue="false"/>
    </Indexed5AxisCompensationMode>
    <!-- используется, когда TCPM операции ВКЛ (непрерывный): -->
    <TCPM5AxisCompensationMode>
        <IsToolTipCompensated        DefaultValue="true"/>
        <IsWorkpieceZeroCompensated  DefaultValue="true"/>
        <IsCoordinateAxesCompensated DefaultValue="true"/>
    </TCPM5AxisCompensationMode>
</ControlData>
```

Локальная система координат (ORIGIN) — индексный режим

Индексная обработка наклонного элемента требует явной команды *повернуть рабочую систему координат* в УП (например, команды плоскости/цикла). Этими параметрами она управляется:

- Доступность **Local coordinate system** — *Unavailable* (у стойки нет такой команды — тогда операция вообще не предлагает опцию Local CS), *Off* (доступна, но по умолчанию выключена) или *Auto*. Выбранное значение становится **значением по умолчанию** для свойства Local CS операции.
- **Auto** вычисляет локальную СК так, чтобы её **Z был вдоль оси инструмента**, сохраняя начало там, где был задан рабочий ноль (например, `G54`).
- **Auto LCS rotation law** — Z вдоль инструмента оставляет поворот *вокруг* Z неопределённым; этот закон его фиксирует: *Snap to Tool CS*, *Snap to Machine CS*, *Snap to Workpiece CS* или *Exclude A/B/C* (принудительно обнулить один пространственный угол, чтобы результат был просто двумя поворотами). Правильный выбор обычно зависит от того, где в кинематике находятся поворотные оси.
- **Local CS positioning mode** — что делают оси станка при выводе команды поворота СК:
  - *Stay* — СК поворачивается, но **ни одна ось не движется**; вершина инструмента остаётся на месте, поэтому её координаты относительно новой СК меняются. Команды поворота осей вы должны добавить сами.
  - *Turn* — дополнительно поворачивает поворотную ось, чтобы инструмент снова совместился с Z новой СК (например, на некоторых стойках за командой поворота СК сразу следует команда «выровнять поворотные»). **Самый частый** — одной команды поворота достаточно.

    - *Move* — как *Turn*, и дополнительно перемещает линейные оси, чтобы вершина инструмента сохраняла те же координаты в повёрнутой СК. Обычно нежелательно.
- **Euler angles type (rotation sequence)** — угловое соглашение для вывода поворота СК (например, `XYZ`, `ZXZ`). Заметьте, что CL data несёт *и* пространственные (эйлеровы) углы, *и* физические значения осей; постпроцессор решает, что выводить. Эти два могут сильно различаться на неортогональной кинематике (например, на столе со скосом 45°), поэтому уточните у интегратора станка, что ожидает стойка.
    - **Rotations around movable axes** — соглашение со штрихами или без (поворот вокруг уже повёрнутых осей или вокруг фиксированных).
    - **Angles in degrees** — градусы или радианы.
- **Move auto LCS with table** — поворачивается ли **начало** авто-СК вместе с деталью (вкл) или остаётся в исходном откалиброванном месте (выкл).
- **Rotatable Workpiece CS** — поворачивается ли СК заготовки вместе с деталью **на момент установки/калибровки** (когда вы задаёте `G54` с уже повёрнутой осью). Выкл. для станков (СК заготовки всегда калибруется в фиксированной ориентации), вкл. для роботов (она жёстко связана с деталью).

```
<ControlData>
    <LocalCS>
        <DefaultState           DefaultValue="Auto"/>   <!-- Unavailable, Off, Auto -->
        <PositioningMode        DefaultValue="Stay"/>    <!-- Turn, Move -->
        <IsSpatial              DefaultValue="True"/>
        <AutoLCSRotationLaw     DefaultValue="SnapToMachineCS"/>
            <!-- SnapToToolCS, SnapToWorkpieceCS, ExcludeA, ExcludeB, ExcludeC -->
        <TurnAutoLCSRotationLaw DefaultValue="ZAlongTurn_XAlongTool"/>
            <!-- ZAlongTurn_MinusXAlongTool, ZAlongTool_XbySnapRule -->
        <MoveAutoLCSWithTable   DefaultValue="true"/>
        <RotatableWCS           DefaultValue="false"/>   <!-- роботы: true -->
        <EulerAnglesType>
            <RotationsSequence          DefaultValue="XYZ"/>  <!-- ZXZ, ZYX, XZY, ... -->
            <RotationsAroundMovableAxes DefaultValue="False"/>
            <AnglesInDegrees            DefaultValue="True"/>
        </EulerAnglesType>
    </LocalCS>
</ControlData>
```

Непрерывная 5-осевая обработка (TCPM)

- **TCPM mode is available** — предлагает ли стойка непрерывное 5-осевое управление / tool centre point management (часто отдельно лицензируемая опция). Если недоступно, операция не показывает переключатель TCPM.
- **TCPM mode default state** — значение по умолчанию для переключателя TCPM операции.

- **Disable TCPM when rotate back** — для поворотных осей с ограниченным диапазоном перемещения, которые нужно «отматывать». Отматывание с активным TCPM заставляет каждую ось двигаться, удерживая вершину инструмента неподвижной — большое рискованное перемещение. При включении система выходит из TCPM, поворачивает обратно только одну поворотную ось, затем снова включает TCPM и продолжает.

Эти три живут под `RotaryTrans` (группа *Continuous 5-axis machining* — лишь группировка в интерфейсе):

```
<ControlData>
    <RotaryTrans>
        <TCPMAvailable         DefaultValue="False"/>
        <TCPMDefault           DefaultValue="False"/>
        <DisableTCPMifOverturn DefaultValue="True"/>  <!-- «Disable TCPM when rotate back» -
->
    </RotaryTrans>
</ControlData>
```

Смена инструмента (Tool Change)

- **Go to tool change position** — *Only if tool change is needed* (отводить в позицию смены только когда последовательные операции используют разные инструменты), *Always at the end of operation* или *Never* (устаревший режим, который просто меняет инструмент без отвода, эмулируя старое поведение «инструмент, парящий в пространстве» из времён до появления симуляции станка).
- **Output mode** — как записываются координаты позиции смены:
  - *Machine coordinates (ISO G53)* — перемещения по физическим осям (например, `G53 Z0`); безопасно, малые числа, не зависит от рабочего нуля.
  - *Reference point (ISO G28)* — отвод через референтную точку станка; часто на токарно-фрезерных / токарных центрах; избегает явного указания позиции смены в рабочей СК.
  - *Tool tip coordinates* — фактические координаты вершины относительно рабочей СК; даёт большие числа и требует, чтобы станок был откалиброван, иначе можно наехать на концевые выключатели.
- **Tool change position** — сама позиция.
- **Heavy axes** — оси, которые планировщик подхода/отвода с обходом столкновений по возможности должен не двигать (ручные/индексные/неудобные оси, например колонна). Перечисляет ID узлов станка.
- **Tool change time calculation** — как оценивается время смены: *Undefined* (встроенная константа, для совместимости) или явный закон, задающий длительность; можно задать несколько законов.

```
<!-- тело типа станка -->
<ToolChange>
    <Using      DefaultValue="Auto"/>     <!-- Anyway, DoNotUse -->
    <OutputMode DefaultValue="RefPoint"/> <!-- Tooltip, Machine -->
</ToolChange>
<ToolChangeMachineState DefaultValue=""/> <!-- позиция смены инструмента -->
<Leads>
    <MotionPlannerOptions>
        <HeavyAxes DefaultValue=""/>       <!-- ID узлов станка через пробел -->
    </MotionPlannerOptions>
</Leads>
```

СОЖ (Coolants)

Какие каналы СОЖ есть у станка — *Flood*, *Mist*, *Tool* (через инструмент) и дополнительные нумерованные каналы — каждый с флагом *Supported* и необязательным **временем переключения**. Включённые каналы появляются на вкладке вывода операции.

```
<Coolants>
    <Coolant5>                          <!-- Coolant1..Coolant20 -->
        <Supported  DefaultValue="True"/>
        <SwitchTime DefaultValue="1"/>
    </Coolant5>
</Coolants>
```

Симуляция (Simulation)

Параметры симуляции обработки/заготовки:

- **Simulation method** — *Voxel 5d* (по умолчанию; быстро, менее точно — воксель это 3D-пиксель), *Voxel 3d* или *Solid* (поверхностная модель; точнее и не зависит от размера вокселя, но медленнее на длинных путях и сложных формах). Вокселные режимы плохо подходят для очень тонких инструментов (например, проволоки EDM), где размер инструмента приближается к размеру вокселя.
- **Model resolution** — число вокселей, на которое делится заготовка (высокое / стандартное / низкое).
- **Gouge detection tolerance** — насколько точно обнаруживаются зарезы в деталь.
- **Revolution bodies simulation** — когда патрон быстро вращается, его кулачки становятся телом вращения; вкл. — это тело вычисляется из 3D-модели кулачка для отображения и столкновений (может замедлить симуляцию); выкл. — кулачки показываются/сталкиваются как есть.

- **Check inappropriate tool and spindle direction** — предупреждает, когда вращение инструмента/шпинделя неверно (например, правый инструмент с реверсированным шпинделем), что испортило бы поверхность или сломало инструмент.
- **Collisions to ignore** — пары узлов, столкновения которых проверять не нужно. Смежные узлы игнорируются автоматически; перечисляйте здесь только несмежные пары, которые никогда не могут столкнуться, чтобы ускорить проверку.
- **Show active part only** — для многоканальных / станков швейцарского типа, обрабатывающих два экземпляра детали сразу; скрывает второй экземпляр детали, когда он мешает.

```
<Simulation>
    <Simulation_Method       DefaultValue="smVoxel5d"/>  <!-- smSolid, smVoxel -->
    <Simulation_Resolution   DefaultValue="srStandard"/> <!-- srLow, srHigh -->
    <GougeDetectionTolerance DefaultValue="0.05"/>
    <RevBodySim              DefaultValue="True"/>
    <CheckSpindleDirection   DefaultValue="true"/>
    <ShowActivePartOnly      DefaultValue="False"/>
</Simulation>
```

Системные настройки → Main

- **Rotation angles type (rotation sequence)** — соглашение об углах Эйлера по умолчанию, используемое везде, где вводятся углы (например, при создании системы координат). Это лишь значение по умолчанию; каждый случай всё равно может его переопределить.

```
<SystemSettings>
    <Main>
        <RotationAnglesType>
            <RotationsSequence DefaultValue="XYZ"/>  <!-- ZXZ, ZYX, ... -->
        </RotationAnglesType>
    </Main>
</SystemSettings>
```

Параметры состояния станка, Schema, размеры станка

Остальная часть вкладки — это собственная кинематика станка:

- **Machine state parameters** — управляемые величины (положения осей X/Y/Z/B/C, ...); см. Параметры состояния станка.
- **Schema** — кинематическое дерево узлов; см. Кинематическую схему.
- **Machine dimensions** — значения калибровки ячейки, устанавливаемые при настройке ячейки: например, положение поворотного приспособления/стола относительно мировой системы

координат ячейки или установленное положение инструмента. Особенно важно для роботизированных ячеек.

Назад к **Дескрипторам станков** | указатель станков

Продвинутые темы по станкам

Эта страница покрывает части создания станка, выходящие за пределы базовой кинематической схемы из Дескрипторов станков: как **позиционировать узлы матрицами**, как строить **переключаемые / съёмные узлы** (блоки револьверной головы, присоединяемые головки, ...), полный разбор **револьверной головы** и **каналы управления и субстанки** для многоканальных станков.

Большинство станков создаются в **MachineMaker** (см. Дескрипторы станков); приёмы здесь — для особых случаев, которые по-прежнему создаются или дорабатываются вручную. Типы-строительные-блоки находятся в `Machines/MachineTypes.xml`, шаблоны револьверной головы/патрона — в `Machines/TurretTypes.xml`, а шаблон многоканального примера — в `Machines/SwissTemplate.xml`. Полные станки поставляются в `$(SCHEMAS_FOLDER)` и являются лучшим справочником.

---

1 Позиционирование узлов матрицами

Каждый узел в `Schema` (`TMachineNode` и его потомки) несёт **`Matrix`**, размещающую его **относительно своего родительского узла**. Вся кинематическая цепочка — это произведение этих матриц вниз по дереву. Заполнить `Matrix` можно двумя способами, плюс есть отдельная матрица, используемая только для 3D-модели.

Тип матрицы (`TMatrix`) имеет селектор **`BaseCS`** — `Parent` (по умолчанию) или `World` — и строится либо из *серии шагов преобразования*, либо из *явного базиса* (`OrtX`/`OrtY`/`OrtZ` + `Move`).

Способ 1 — относительно СК родителя (серия перемещений и поворотов)

Обычный способ. Оставьте базовую матрицу единичной и **добавьте серию шагов переноса и поворота**, применяемых в порядке объявления. Типы шагов — `TTranslateX` / `TTranslateY` / `TTranslateZ` и `TRotateX` / `TRotateY` / `TRotateZ` (градусы). Каждый шаг — это член, который вы объявляете внутри `Matrix`:

```
<SCType ID="MyNode" type="TMachineNode">
    <Matrix>
        <SCType ID="T1" type="TTranslateZ" DefaultValue="100"/>  <!-- затем... -->
        <SCType ID="T2" type="TRotateX"    DefaultValue="90"/>
    </Matrix>
</SCType>
```

Именно так устроены шаблоны револьверной головы, со смещениями, управляемыми выражениями (см. §3):

```
<!-- TurretTypes.xml: блок револьверной головы, размещённый на голове -->
<Matrix>
    <SCType ID="T1" type="TTranslateX" DefaultValue="-[TurretParameters.Radius]"/>
    <SCType ID="T2" type="TRotateZ"
            DefaultValue="-
[TurretParameters.RotationDirection]*360/[TurretParameters.BlocksCount]*
([Parameters.BlockNumber]-1)"/>
</Matrix>
```

`BaseCS` остаётся `Parent` (по умолчанию), поэтому шаги интерпретируются в системе координат родительского узла. Порядок важен — сначала применяется `T1`, затем `T2`.

Способ 2 — относительно мировой СК (пространственные углы)

Когда узел проще описать в **мировой** системе координат ячейки, установите `BaseCS` в `World` и задайте одно пространственное преобразование (`TComplexTransformation3d`): перенос плюс три угла в выбранном **соглашении** (Эйлер / фиксированные оси / кватернион / axis-angle).

```
<Matrix>
    <BaseCS DefaultValue="World"/>
    <SCType ID="T1" type="TComplexTransformation3d">
        <Translation><X DefaultValue="0"/><Y DefaultValue="0"/><Z
DefaultValue="300"/></Translation>
        <Rotation>
            <Convention DefaultValue="EulerZYX"/>  <!-- FixedXYZ, Quaternion, AxisAngleDeg,
... -->
            <R1 DefaultValue="0"/>
            <R2 DefaultValue="45"/>
            <R3 DefaultValue="0"/>
            <R4 DefaultValue="0"/>   <!-- 4-е значение только для кватерниона / axis-angle -
->
        </Rotation>
    </SCType>
</Matrix>
```

> Базис можно также задать явно через `OrtX`/`OrtY`/`OrtZ` (три вектора осей) и `Move` (начало координат), но на практике вы будете использовать две формы выше.

Матрица 3D-модели (`VisualProperties.VisMatrix`)

Матрица узла выше задаёт его **кинематическую** систему. **3D-модель**, прикреплённая к узлу (`ImageFile`, файл `.osd` или `.stl`), позиционируется *отдельной* матрицей `VisMatrix` внутри `VisualProperties`. Используйте её, чтобы выровнять импортированную модель с системой узла, не нарушая кинематику:

```
<SCType ID="MyNode" type="TMachineNode">
    <ImageFile DefaultValue="Images\head.osd"/>
    <VisualProperties>
        <VisMatrix>
            <SCType ID="T1" type="TRotateZ" DefaultValue="180"/>
        </VisMatrix>
    </VisualProperties>
</SCType>
```

(Визуализация станка обрабатывается отдельно от кинематики; `VisMatrix` влияет только на то, как отрисовывается модель.)

---

2 Переключаемые / съёмные узлы

У реальных станков есть взаимозаменяемые части — блоки револьверной головы, присоединяемые фрезерные головки, адаптеры субшпинделя, варианты патрона. В пределах **XML-схемы станка** существует **два механизма** моделирования «узла, который может присутствовать или нет», с разными компромиссами.

> **Область применения.** Этот раздел покрывает только два механизма *XML-схемы*. Другие виды съёмного содержимого обрабатываются **на уровне проекта отдельными, не-XML механизмами** и не связаны с описанной здесь схемой — например, **приспособления** (оснастка заготовки, хранящаяся в собственном бинарном контейнере вместе с 3D-моделями) и **сборки инструмента** (строятся в собственных структурах данных и подключаются к коннекторам магазина станка). Не используйте `TCaseNode` или скрытие узлов для их моделирования.

Механизм A — `TCaseNode` (одна ветвь загружается при разборе)

Узел, наследующийся от `TCaseNode`, содержит несколько альтернативных дочерних ветвей, но предоставляет **`ActiveNode`**, называющий ту, что «в работе». При разборе дескриптора станка **загружается только активная ветвь**; остальные полностью пропускаются.

```
<SCType ID="HeadBlockSelector" type="TCaseNode">
    <ActiveNode DefaultValue="EmptyBlock"/>   <!-- какую дочернюю ветвь загрузить -->
```

```
    <Parameters>
        <SCType ID="BlockNumber" type="Integer" DefaultValue="1"/>
    </Parameters>
    <SCType ID="EmptyBlock"  Caption="Empty"  type="TAbstractTurretHeadBlock"/>
    <SCType ID="DrillBlock"  Caption="Drill"  type="TTurretDrillToolHolder"/>
    <!-- ...другие альтернативы... -->
</SCType>
```

- **Плюсы:** неактивные ветви не потребляют память — критично, когда у станка много необязательных блоков (револьверная голова с десятками позиций, инструментальный магазин, ...).
- **Ограничение:** активная ветвь выбирается **один раз, при загрузке/разборе**. Проект владеет **единственным экземпляром станка**, поэтому переключить активный блок *во время* проекта нельзя — только инициализировать в начале. Это классический механизм для **револьверных голов** (см. §3).

`TCaseNode` несёт `ActiveNode` (ID активного потомка) и необязательный блок `Parameters`; оба пропускаются загрузчиком, когда он спускается в выбранную ветвь.

Механизм B — скрытие неактивных узлов (переключаемые во время работы)

Здесь загружаются **все** узлы, но узел **показывается/активен только когда он принадлежит текущему выбранному коннектору инструмента или заготовки**. Поскольку ничего не выгружается, активный блок может меняться **посреди проекта**. Этим управляют три атрибута:

Атрибут	Тип	Узел с ним активен только когда...
`VisToolHolderID`	строка	его значение равно **ID активного коннектора инструмента**
`VisWorkpieceHolderID`	строка	его значение равно **ID активного коннектора заготовки**
`HideIfInactive`	лог.	узел лежит на **ветви активного инструмента или заготовки** (более простой переключатель, чем явное указание ID)

```
<!-- этот адаптер показывается только когда активен коннектор инструмента "Tool3" -->
<SCType ID="Adapter3" type="TMachineNode" VisToolHolderID="Tool3">
    ...
</SCType>

<!-- эта ветвь показывается только пока несёт активный инструмент/заготовку -->
<SCType ID="OptionalHead" type="TMachineNode" HideIfInactive="True">
```

```
    ...
</SCType>
```

`HideIfInactive` — это простой эквивалент `VisToolHolderID` / `VisWorkpieceHolderID`, когда вы не хотите жёстко прописывать ID коннекторов (которые могут меняться): вместо указания ID вы просто помечаете узел, и он активен всякий раз, когда является предком текущего коннектора инструмента или заготовки.

Что выбрать

	`TCaseNode` **(A)**	**Скрытие (B)**
Загружается в память	только активная ветвь	все ветви
Когда делается выбор	один раз, при разборе/загрузке	в любой момент, посреди проекта
Память при многих вариантах	низкая	выше
Типичное применение	блоки револьверной головы, большие наборы опций	присоединяемые головки/адаптеры, переключаемые во время работы

---

3 Револьверные головы

Револьверная голова — хрестоматийное применение §1 (параметрические матрицы), §2 Механизм A (выбор блока через `TCaseNode`) и **параметризации** (чтобы ID узлов оставались уникальными среди одинаковых блоков). Поставляемые шаблоны — в `Machines/TurretTypes.xml`; некоторые старые станки строят револьверную голову без шаблона, но по тому же принципу (шаблон добавили позже).

Составные части

1. Голова револьверной головы — поворотная индексная ось. Она содержит блок `TurretParameters` (префикс, число позиций, радиус, ID линейных осей, на которых едет голова, направление вращения) и индексный параметр состояния. `Scale` отображает один шаг индекса в `360 / BlocksCount` градусов:

```
<SCType ID="TAbstractTurretHead" Caption="Turret head" type="TMachineAxis">
    <ID            DefaultValue="[TurretParameters.Prefix]TurretHead"/>
    <ParameterName DefaultValue="[TurretParameters.Prefix]TurretAxisPos"/>
```

```
    <AxisType      DefaultValue="Rotary"/>
    <Scale         DefaultValue="360/[TurretParameters.BlocksCount]"/>
    <SCType ID="TurretParameters" Caption="Turret parameters" type="ComplexType">
        <SCType ID="Prefix"            type="String"  DefaultValue=""/>
        <SCType ID="BlocksCount"       type="Integer" DefaultValue="12"/>
        <SCType ID="Radius"            type="Double"  DefaultValue="120"/>
        <SCType ID="XAxisID"           type="String"  DefaultValue="[Prefix]AxisX"/>
        <SCType ID="ZAxisID"           type="String"  DefaultValue="[Prefix]AxisZ"/>
        <SCType ID="RotationDirection" type="Integer" DefaultValue="1"/>
    </SCType>
    <SCType ID="TurretAxisPos" type="TMachineStateParameter">
        <ID          DefaultValue="[TurretParameters.Prefix]TurretAxisPos"/>
        <Address     DefaultValue="[TurretParameters.Prefix]T"/>
        <AxisControl DefaultValue="Indexed"/>
        <Group       DefaultValue="RotaryAxis"/>
    </SCType>
</SCType>
```

2. Блок — размещён на голове параметрической матрицей. Каждая позиция повёрнута на свой угол и вынесена наружу на радиус головы (§1, Способ 1):

```
<SCType ID="TAbstractTurretHeadBlock" Caption="Turret head block" type="TMachineNode">
    <Matrix>
        <SCType ID="T1" type="TTranslateX" DefaultValue="-[TurretParameters.Radius]"/>
        <SCType ID="T2" type="TRotateZ"
                DefaultValue="-
[TurretParameters.RotationDirection]*360/[TurretParameters.BlocksCount]*
([Parameters.BlockNumber]-1)"/>
    </Matrix>
</SCType>
```

3. Держатель инструмента на блоке — с параметрическими уникальными ID. Каждый блок должен давать **уникальные** ID узлов и номера инструмента; параметризация от `[Parameters.BlockNumber]` и `[TurretParameters.Prefix]` это гарантирует:

```
<SCType ID="TTurretToolHolder" Caption="Tool holder" type="TToolHolderNode">
    <ID           DefaultValue="[TurretParameters.Prefix]Tool[Parameters.BlockNumber]"/>
    <ToolAxisID   DefaultValue="[TurretParameters.Prefix]TurretHead"/>
    <ToolAxisValue DefaultValue="[TurretParameters.RotationDirection]*
([Parameters.BlockNumber]-1)"/>
    <ToolNumber   DefaultValue="[Parameters.BlockNumber]"/>
    <XAxisID      DefaultValue="[TurretParameters.XAxisID]"/>
```

```
    <ZAxisID       DefaultValue="[TurretParameters.ZAxisID]"/>
</SCType>
```

4. Селектор блока — `TCaseNode`. Каждая позиция — это селектор, чей `Parameters.BlockNumber` отличает её, а `ActiveNode` выбирает установленный тип блока (пусто, токарный резец, приводное сверло, ...). Загружается только активный блок каждой позиции:

```
<SCType ID="TAbstractTurretHeadBlockSelector" type="TCaseNode">
    <ActiveNode DefaultValue="EmptyBlock"/>
    <Parameters>
        <SCType ID="BlockNumber" type="Integer" DefaultValue="1"/>
    </Parameters>
    <SCType ID="EmptyBlock" Caption="Empty" type="TAbstractTurretHeadBlock"/>
    <!-- конкретный станок добавляет: держатель токарного резца, держатель сверла, ... как
альтернативы -->
</SCType>
```

Почему здесь необходима параметризация

У револьверной головы много одинаковых позиций. Если бы у их узлов были фиксированные ID, они бы конфликтовали. Управляя каждым `ID`, `ToolNumber`, `Address` и смещением матрицы от `[Parameters.BlockNumber]` (на позицию) и `[TurretParameters.Prefix]` (на голову, чтобы две головы на одном станке не сталкивались), один шаблон чисто инстанцируется любое число раз. Это тот же язык выражений `[...]`, что и везде (см. язык выражений); дескрипторы станков сильно на него опираются.

> Шаблоны патрона/кулачков в том же файле следуют идентичному образцу — параметрический блок `ChuckParameters` и матрицы по кулачкам, управляемые `[JawIndex]`.

Собираем вместе — сборка револьверной головы из шаблона

Теперь объединим части шаблона в полную переиспользуемую револьверную голову. Всё наследуется от типов `TurretTypes.xml`, поэтому шаговый угол головы, матрицы блоков и все ID создаются автоматически из параметров.

1. Определите блоки станка — каждый блок наследуется от `TAbstractTurretHeadBlock` (наследуя его матрицу радиального/углового позиционирования) и содержит подходящий параметрический держатель:

```
<SCType ID="LatheCutterBlock" Caption="Lathe cutter holder" type="TAbstractTurretHeadBlock">
    <SCType ID="ToolHolder" type="TTurretLatheCutterHolder"/>
```

```
</SCType>
<SCType ID="DrivenDrillBlock" Caption="Driven drill holder" type="TAbstractTurretHeadBlock">
    <SCType ID="ToolHolder" type="TTurretDrillToolHolder"/>
</SCType>
```

> Держатели выше — это готовые к револьверной голове держатели шаблона, тоже объявленные в [TurretTypes.xml]: **`TTurretToolHolder`** (от обобщённого коннектора `TToolHolderNode`), **`TTurretLatheCutterHolder`** (от `TLatheCutterHolder`) и **`TTurretDrillToolHolder`** (от `TMillToolHolder`). Каждый лишь *добавляет параметрические ID инструмента револьверной головы* — `ID = [TurretParameters.Prefix]Tool[Parameters.BlockNumber]`, плюс `ToolAxisID`, `ToolNumber` и связанные `XAxisID`/`YAxisID`/`ZAxisID` — поверх своего типа коннектора. Именно это позволяет блоку просто вставить один из них и автоматически унаследовать уникальные, префиксованные имена.

2. Определите селектор блоков станка — унаследуйте его от шаблонного селектора `TAbstractTurretHeadBlockSelector` (`TCaseNode`, уже предоставляющий `ActiveNode`, `Parameters.BlockNumber` и унаследованную альтернативу `EmptyBlock`) и добавьте блоки, которые этот станок поддерживает:

```
<SCType ID="TMyBlockSelector" Caption="Turret block selector"
type="TAbstractTurretHeadBlockSelector">
    <SCType ID="LatheCutter" Caption="Lathe cutter" type="LatheCutterBlock"/>
    <SCType ID="DrivenDrill" Caption="Driven drill" type="DrivenDrillBlock"/>
    <!-- "EmptyBlock" унаследован от TAbstractTurretHeadBlockSelector -->
</SCType>
```

3. Постройте револьверную голову как тип — унаследуйте её от `TAbstractTurretHead` (который даёт поворотную индексную ось и вычисляет за вас `Scale = 360 / BlocksCount`), задайте её `TurretParameters` и навесьте 12 селекторов позиций. `BlockNumber` каждой позиции делает её ID уникальными; её `ActiveNode` говорит, какой блок там установлен:

```
<SCType ID="TMyTurret" Caption="Turret" type="TAbstractTurretHead">
    <TurretParameters>
        <BlocksCount DefaultValue="12"/>   <!-- Scale головы становится 360/12 = 30° -->
        <Radius      DefaultValue="120"/>
    </TurretParameters>

    <SCType ID="T1"  Caption="Position 1"  type="TMyBlockSelector">
        <ActiveNode DefaultValue="LatheCutter"/>   <!-- позиция 1: токарный резец -->
        <Parameters><BlockNumber DefaultValue="1"/></Parameters>
    </SCType>
```

```
    <SCType ID="T2"  Caption="Position 2"  type="TMyBlockSelector">
        <ActiveNode DefaultValue="DrivenDrill"/>   <!-- позиция 2: приводное сверло -->
        <Parameters><BlockNumber DefaultValue="2"/></Parameters>
    </SCType>
    <SCType ID="T3"  Caption="Position 3"  type="TMyBlockSelector">
        <ActiveNode DefaultValue="EmptyBlock"/>    <!-- позиция 3: пусто (унаследованный
блок) -->
        <Parameters><BlockNumber DefaultValue="3"/></Parameters>
    </SCType>

    <!-- ...позиции T4 … T11, каждая — TMyBlockSelector со своим
         ActiveNode и BlockNumber = 4 … 11 ... -->

    <SCType ID="T12" Caption="Position 12" type="TMyBlockSelector">
        <ActiveNode DefaultValue="LatheCutter"/>
        <Parameters><BlockNumber DefaultValue="12"/></Parameters>
    </SCType>
</SCType>
```

Вот и вся идея от начала до конца:

- **Один тип селектора** перечисляет все возможные блоки; **двенадцать его экземпляров** (`T1`…`T12`) — это физические позиции на голове.
- **`ActiveNode`** каждой позиции выбирает *один* загружаемый там блок — остальные альтернативы не стоят памяти (§2 A).
- **`BlockNumber`** каждой позиции вместе с `TurretParameters` головы параметризует угол матрицы блока и его `ID`/`ToolNumber` инструмента, так что двенадцать одинаковых позиций дают двенадцать различных, неконфликтующих инструментов.
- Конфигурация фиксируется при разборе станка; чтобы переоснастить позицию, пользователь редактирует станок, который перезагружается.

Размещение револьверной головы(голов) в станке — зачем нужен `Prefix`

`TMyTurret` теперь — переиспользуемый тип. Разместите его в `Schema` станка в конце суппорта, который его несёт, чтобы он двигался с этими осями. С **одной** револьверной головой её `Prefix` можно оставить пустым:

```
<Schema>
    <SCType ID="AxisX" type="TToolAxisX">
        <SCType ID="AxisZ" type="TToolAxisZ">
            <SCType ID="Turret" Caption="Turret" type="TMyTurret"/>
        </SCType>
```

```
    </SCType>
</Schema>
```

Но токарный/токарно-фрезерный станок может нести **две револьверные головы** (верхнюю/нижнюю), а станок загружается в **единое пространство имён**, поэтому каждый ID узла должен быть уникален *в пределах станка* (как загружается станок). Две копии `TMyTurret` каждая выдала бы `TurretHead`, `Tool1`…`Tool12`, `TurretAxisPos`, … — конфликт. Именно поэтому `TAbstractTurretHead` параметризует каждое генерируемое имя **`Prefix`**-ом: дайте каждому экземпляру свой префикс, и все его ID станут уникальными:

```
<Schema>
    <!-- нижняя револьверная голова на нижнем суппорте X/Z -->
    <SCType ID="LowerX" type="TMachineAxis"> <ParameterName DefaultValue="LowerAxisXPos"/>
      <SCType ID="LowerZ" type="TMachineAxis"> <ParameterName DefaultValue="LowerAxisZPos"/>
        <SCType ID="LowerTurret" Caption="Lower turret" type="TMyTurret">
            <TurretParameters>
                <Prefix  DefaultValue="Lower"/>   <!-- -> LowerTurretHead, LowerTool1,
LowerTurretAxisPos -->
                <XAxisID DefaultValue="LowerX"/>
                <ZAxisID DefaultValue="LowerZ"/>
            </TurretParameters>
        </SCType>
      </SCType>
    </SCType>

    <!-- верхняя револьверная голова на собственном суппорте -->
    <SCType ID="UpperX" type="TMachineAxis"> <ParameterName DefaultValue="UpperAxisXPos"/>
      <SCType ID="UpperZ" type="TMachineAxis"> <ParameterName DefaultValue="UpperAxisZPos"/>
        <SCType ID="UpperTurret" Caption="Upper turret" type="TMyTurret">
            <TurretParameters>
                <Prefix  DefaultValue="Upper"/>   <!-- -> UpperTurretHead, UpperTool1,
UpperTurretAxisPos -->
                <XAxisID DefaultValue="UpperX"/>
                <ZAxisID DefaultValue="UpperZ"/>
            </TurretParameters>
        </SCType>
      </SCType>
    </SCType>
</Schema>
```

Каждый экземпляр переиспользует то же определение `TMyTurret` (все 12 позиций и их блоки), но благодаря своему отличному `Prefix` выдаёт собственные неконфликтующие ID — `LowerTurretHead` / `LowerTool1…` против `UpperTurretHead` / `UpperTool1…` — и ссылается на свои оси суппорта через

параметризованные `XAxisID` / `ZAxisID`. (В случае одной револьверной головы `Prefix` можно оставить пустым.) На многоканальных станках две головы обычно также сидят на разных `Channel` — см. §6.

---

4 Коннекторы инструмента — оси позиционирования и активация

Коннектор — это место, где инструмент или заготовка крепятся к кинематической цепочке — *коннектор инструмента* (`TToolHolderNode`) и *коннектор заготовки* (`TWorkpieceHolderNode`) интерфейса (см. строительные блоки). Оба наследуются от общего гнезда и потому разделяют свойства шпинделя/зажима из Шпиндель и зажим. На **станках** (не роботах) **инструментальный** коннектор дополнительно несёт свойства, сообщающие решателю кинематики, как его инструмент движется и как он выбирается.

> **Полные таблицы свойств** коннекторов собраны в справочнике узлов → Коннекторы инструмента и заготовки; этот раздел объясняет *поведение* за этими свойствами.

Оси позиционирования и базовая СК — `XAxisID` / `YAxisID` / `ZAxisID`

`XAxisID`, `YAxisID`, `ZAxisID` называют оси станка, которые **позиционируют вершину этого инструмента в пространстве**. Они дают решателю пригодную цепочку *даже без явного субстанка* (§5) — по сути каждый коннектор уже описывает небольшой «виртуальный субстанок». Они также определяют **базовую систему координат** этой цепочки.

Явные субстанки позже добавили `OriginG54BaseNode` / `OriginG54`, чтобы задавать эту базовую СК напрямую. Когда их нет, базовая СК выводится:

- её **начало** совпадает с **началом мировой СК** станины станка;
- её **направления осей** берутся из **направлений перемещения** `XAxisID` / `YAxisID` / `ZAxisID`.

Результат **должен быть правой тройкой**, поэтому задавайте направления перемещения осей аккуратно. **Ветвь**, в которой сидит ось, имеет значение:

- ось в **инструментальной** ветви вносит своё направление перемещения **напрямую** (положительно);
- ось в **заготовочной** ветви вносит **противоположное** направление (отрицательно) — перемещение детали в одну сторону эквивалентно перемещению инструмента в другую.

Отрицательный `Scale` оси (справочник узлов) тоже переворачивает направление перемещения, поэтому он также влияет на эту базовую СК.

Активация коннектора — `ToolAxisID` / `ToolAxisValue`

Это основной механизм за **инструментальными магазинами вообще и револьверными головами в частности**. Коннектор заявляет: *чтобы сделать инструмент в этом коннекторе активным, переместите ось `ToolAxisID` в значение `ToolAxisValue`*. Это работает одинаково для **поворотных** магазинов (револьверная голова индексируется к углу позиции) и **линейных** (магазин сдвигается в позицию).

```
<SCType ID="Tool5" type="TMillToolHolder">
    <ToolAxisID    DefaultValue="TurretHead"/>   <!-- ось, индексирующая магазин -->
    <ToolAxisValue DefaultValue="4"/>            <!-- значение, вводящее эту позицию -->
    <ToolNumber    DefaultValue="5"/>
</SCType>
```

В шаблоне револьверной головы они параметрические — `ToolAxisID = [TurretParameters.Prefix]TurretHead` и `ToolAxisValue = [TurretParameters.RotationDirection]*([Parameters.BlockNumber]-1)` — так что каждая позиция автоматически указывает на ось револьверной головы и свой индекс.

> **`ToolAxisID` можно опустить.** Когда активный субстанок (§5) определяет ось инструмента, система использует *её*; к собственному `ToolAxisID` коннектора она откатывается, только когда у субстанка её нет. Так что коннектор может оставить `ToolAxisID` пустым и позволить активному субстанку её предоставить.

Номер инструмента и поддерживаемые применения

- **`ToolNumber`** — **позиция в магазине** этого коннектора, обычно известная как **номер инструмента**; именно его постпроцессор выводит в УП. В шаблоне револьверной головы он параметризован индексом позиции. Если оставить `0` (не задано), номер коннектора **не** используется, и УП вместо этого выводит номер, взятый из **собственных свойств режущего инструмента**.
- **`SupportedToolTypes`** (*поддерживаемые применения*) — виды обработки, которые может выполнять инструмент, установленный в этот коннектор: `MillTool` (фрезерование), `LatheCutter` (токарная обработка), `JetCutter` (струйная резка), `Punch`, `Wire` (проволочная EDM), `Cutter6D` (6D-резка), `Welder` (сварка), `AdditiveTool` (аддитивный), `Painter` (окраска распылением), `HeatTreatment`, `Gripper` (захват). Система использует этот набор, чтобы **фильтровать списки операций и инструментов**, предлагаемых для коннектора, до тех, что поддерживаются его применениями.
- **`Channel`** — канал управления, к которому принадлежит этот коннектор (§6).
- **`ToolChangeTimeCalcLawIndex`** — индекс закона времени смены инструмента (заданного в секции *Tool Change* станка), применяемого к этому коннектору (`0` = закон по умолчанию).

Шпиндель и зажим (общие для обоих коннекторов)

Поскольку коннекторы инструмента и заготовки наследуются от одного гнезда, эти свойства применимы к **обоим**:

- `SpindleParamID` — ID **параметра состояния станка** (параметры состояния станка), приводящего поворотную ось **шпинделя**, которая вращает инструмент или деталь. Это ID *параметра состояния*, а не `ID` узла оси.
- `HolderType` — `Unknown`, `LeftLatheSpindle` или `RightLatheSpindle`; помечает гнездо токарного шпинделя.
- `DefaultClampID` — ID зажима по умолчанию для этого гнезда, на который ссылаются операции передачи/перехвата детали; по умолчанию выводится из `HolderType` (левый шпиндель → `1`, правый → `2`, иначе `-1`).

Коннекторы заготовки

Коннектор заготовки (`TWorkpieceHolderNode`) удерживает **деталь** (на патроне, шпинделе или приспособлении) и несёт только общие свойства шпинделя/зажима выше — ни одного из полей позиционирования инструмента, активации или `ToolNumber` / `SupportedToolTypes`, которые описывают, как позиционируется и выбирается *инструмент*, и неприменимы к держателю детали. Его полный список свойств и **оговорку о размещении** (`Matrix` узла игнорируется — коннектор заготовки всегда принимает размещение своего родительского узла) см. в справочнике узлов → Коннектор заготовки.

---

5 Субстанки (попеременная работа)

> **Каналы vs. субстанки — ключевое различие.** *Каналы* (§6) описывают части станка, работающие **одновременно**. *Субстанки* описывают альтернативные кинематические цепочки, используемые **по одной за раз** (попеременно) — они решают, *какие* оси приводят инструмент. Они встречаются даже на **одноканальных** станках, поэтому это отдельное понятие.

Решатель обратной кинематики (OK) работает с цепочкой **фиксированной формы**: три линейные оси плюс **две** поворотные оси (`X Y Z` + `RotaryAxis1` + `RotaryAxis2`). Когда дерево узлов станка предлагает *больше* — дополнительные поворотные оси или несколько ветвей инструмента/заготовки (шпиндели, револьверные головы) — цепочка становится **избыточной**, и решатель не может выбрать. **Субстанок** устраняет неоднозначность, называя **активную ветвь**: узел инструмента, узел заготовки и ровно то, какие линейные и поворотные оси участвуют.

Канонический пример — 6-осевой фрезерный (XYZ + ABC)

У распространённого обрабатывающего центра есть линейные `X Y Z` плюс **три** поворотные оси `A`, `B`, `C` — но решатель ОК всегда приводит лишь две поворотные. Три поворотные избыточны. Вы разрешаете это, объявив **два субстанка**, каждый закрепляет пригодную цепочку *XYZ + две поворотные*:

- `XYZ + A + C`
- `XYZ + B + C`

Затем операция (или пользователь) выбирает, какой субстанок использовать для данной работы. Используйте вариант `SubMachine5x`, добавляющий два поля поворотных осей:

```
<SubMachinesList>
    <SCType ID="XYZ_AC" Caption="A + C" type="SubMachine5x">
        <ToolNode DefaultValue="Spindle"/>   <!-- узел, несущий инструмент -->
        <WrkNode  DefaultValue="Table"/>      <!-- узел, несущий деталь -->
        <XAxisID  DefaultValue="AxisX"/>
        <YAxisID  DefaultValue="AxisY"/>
        <ZAxisID  DefaultValue="AxisZ"/>
        <R1AxisID DefaultValue="AxisA"/>      <!-- первая поворотная -->
        <R2AxisID DefaultValue="AxisC"/>      <!-- вторая поворотная -->
    </SCType>
    <SCType ID="XYZ_BC" Caption="B + C" type="SubMachine5x">
        <ToolNode DefaultValue="Spindle"/>
        <WrkNode  DefaultValue="Table"/>
        <XAxisID  DefaultValue="AxisX"/>
        <YAxisID  DefaultValue="AxisY"/>
        <ZAxisID  DefaultValue="AxisZ"/>
        <R1AxisID DefaultValue="AxisB"/>
        <R2AxisID DefaultValue="AxisC"/>
    </SCType>
</SubMachinesList>
```

Каждый субстанок — это одна решаемая цепочка XYZ + 2 поворотные; вместе они покрывают весь 6-осевой станок, ни разу не прося решатель обработать три поворотные сразу. Этот станок — **одноканальный**: субстанки нужны независимо от каналов.

Второй пример — двухшпиндельный / двухревольверный токарно-фрезерный

Очень распространённый токарно-фрезерный центр имеет **два носителя инструмента** (верхнюю и нижнюю револьверные головы) и **два носителя детали** (левый и правый шпиндели). Любая револьверная голова может обрабатывать деталь, удерживаемую любым шпинделем, поэтому

пригодные пары **инструмент + заготовка** перечисляются как субстанки — плюс ещё одна запись для работы шпиндель-в-шпиндель:

- **SubMachine 1** — UpperTurret → LeftSpindle
- **SubMachine 2** — UpperTurret → RightSpindle
- **SubMachine 3** — LowerTurret → LeftSpindle
- **SubMachine 4** — LowerTurret → RightSpindle
- **SubMachine 5** — RightSpindle *как держатель инструмента* → LeftSpindle *как держатель заготовки* — для осевого сверления или передачи/перехвата детали между двумя шпинделями.

```
<SubMachinesList>
    <SCType ID="UpperOnLeft" Caption="Upper turret → Left spindle" type="SubMachine">
        <ToolNode   DefaultValue="UpperTurret"/>   <!-- ГОЛОВА револьверной головы, не
отдельный держатель -->
        <WrkNode    DefaultValue="LeftSpindle"/>    <!-- узел шпинделя, удерживающий деталь
-->
        <XAxisID    DefaultValue="UpperX"/>
        <ZAxisID    DefaultValue="UpperZ"/>
        <ToolAxisID DefaultValue="UpperTurret"/>    <!-- ось индексации револьверной головы
-->
        <Channel    DefaultValue="0"/>
    </SCType>
    <!-- ...UpperOnRight, LowerOnLeft, LowerOnRight — остальные пары револьверная
голова×шпиндель... -->
    <SCType ID="RightToLeft" Caption="Right spindle → Left spindle" type="SubMachine">
        <ToolNode DefaultValue="RightSpindle"/>   <!-- правый шпиндель выступает как
носитель инструмента -->
        <WrkNode  DefaultValue="LeftSpindle"/>     <!-- левый шпиндель удерживает деталь -->
    </SCType>
</SubMachinesList>
```

> `ToolNode` / `WrkNode` **называют узел-*носитель*, а не отдельный держатель.** Указывайте **ближайшего общего родителя** задействованных держателей — **голову револьверной головы** (родитель всех её держателей инструмента) или узел **шпинделя** — *а не* отдельный `TToolHolderNode` / `TWorkpieceHolderNode`. В этом и весь смысл (настоящих) субстанков: оси ветви (`XAxisID` / `ZAxisID` / `ToolAxisID`) указываются **один раз, на субстанке**, а не повторяются на каждом держателе инструмента, как требовал более старый «виртуальный субстанок» (коннекторные `XAxisID`… из §4). Полный реальный список см. в поставляемом `Machines\LatheMilling\MultiChannel\Puma MX2100ST\DOOSAN_PUMA_MX2100ST.xml` (например, `ToolNode=AxisT` для револьверной головы, `WrkNode=AxisC` для шпинделя).

Итак, каждая запись называет, **какой носитель держит инструмент, а какой — деталь**, давая решателю одну однозначную пару; оператор (или операция) выбирает активный субстанок. Последняя запись показывает, что **сам шпиндель может быть носителем инструмента** — например, контршпиндель надвигает деталь на неподвижное сверло или перенимает деталь у главного шпинделя. Там, где случай XYZ+ABC использовал субстанки для разрешения **избыточных поворотных осей**, этот случай использует их, чтобы выбрать, **какие из нескольких носителей инструмента/заготовки** образуют активную пару.

Объявление и поля

Субстанки перечисляются в `SubMachinesList` станка; каждая запись — это `SubMachine` или вариант (более старая форма-массив `SubMachines` устарела).

Поле	Смысл
`ToolNode` / `WrkNode`	Узлы-**носители**, несущие инструмент и заготовку (ближайший общий родитель держателей — голова револьверной головы, шпиндель), **а не** отдельный держатель.
`XAxisID` / `YAxisID` / `ZAxisID`	Линейные оси этой цепочки.
`ToolAxisID`	Ось инструмента / шпинделя.
`OriginG54BaseNode`	Узел, к которому привязана базовая система координат субстанка.
`OriginG54`	Произвольная **ориентация** (преобразование) для этой базовой СК — обычно используется на станках с **контршпинделем**, где **Z базовой СК нужно инвертировать**.
`ApproachRule` / `ReturnRule`	Общее правило подхода / возврата для этой ветви (см. ниже).
`DetailedApproachRules` / `DetailedReturnRules`	Варианты этих правил по режиму обработки, выбираемые автоматически по типу операции (см. ниже).
`Channel`	Канал, к которому принадлежит этот субстанок — актуально только на многоканальных станках (§6).

Типы субстанков. Какой тип `SubMachine` использовать, зависит от **решателя кинематики**, который нужен ветви, или от дополнительных опций, специфичных для вида оборудования. Обычный `SubMachine` несёт общие поля выше; специализированные типы добавляют поля, которые требует их решатель. Набор **открыт и будет расти** со временем; на данный момент существуют две специализации помимо базовой:

- `SubMachine5x` — для **5-осевых** станков. Добавляет `R1AxisID` / `R2AxisID`, две поворотные оси, которыми управляет 5-осевой решатель (использовано в примере XYZ+ABC выше).
- `TrevisanSubMachine` — для станков с функцией **точения осью U** (типичные представители — станки типа Trevisan). Добавляет радиальную `UAxisID`.

Правила подхода и возврата

`ApproachRule` / `ReturnRule` (и их детальные формы) описывают, **как станок движется между точкой смены инструмента и телом траектории** для этого субстанка: правило **подхода** от позиции смены инструмента к *началу* траектории операции и правило **возврата** от её *конца* обратно к позиции смены инструмента.

Правило — это упорядоченная последовательность перемещений осей, записанная как группы, разделённые `;`, которые выполняются по очереди; буквы, сгруппированные вместе, движутся вместе, а `Axis(value)` перемещает ось в заданное значение. Например, `ApproachRule = "BC; Y Z(10); X"` означает: сначала повернуть **B и C**, затем переместить **Y** и **Z** (в 10), затем подвести **X**.

Шаг может быть и **командным токеном**, а не перемещением оси. В частности, `LCS` включает или выключает **локальную систему координат** в этой точке последовательности — или, для соответствующего типа 5-осевой обработки, режим **TCPM** — как видно в вариантах `MillLCS` / `MillTCPM` ниже (`BC;Z;X;LCS;XY;Z`).

- `ApproachRule` **/** `ReturnRule` дают **одно общее** правило.
- `DetailedApproachRules` **/** `DetailedReturnRules` дают варианты **по режиму обработки**, так что нужное правило выбирается **автоматически** по типу текущей операции:
  - `TurnRadial` / `TurnAxial` — радиальное / осевое **точение**;
  - `MillRadial` / `MillAxial` — радиальное / осевое обычное **3-осевое фрезерование**;
  - `MillLCS` — **индексная** 5-осевая обработка (локальная СК);
  - `MillTCPM` — **непрерывная** 5-осевая обработка (TCPM).

```
<SCType ID="MainMill" type="SubMachine">
    ...
    <ApproachRule DefaultValue="BC; Y Z(10); X"/>
    <ReturnRule   DefaultValue="Z(10); X; Z"/>
    <DetailedApproachRules>
        <TurnRadial DefaultValue="BS;Z;X"/>
        <TurnAxial  DefaultValue="BS;Z(10);X;"/>
        <MillRadial DefaultValue="BC;Z;X"/>
        <MillAxial  DefaultValue="BC;Z;X"/>
        <MillLCS    DefaultValue="BC;Z;X;LCS;XY;Z"/>
        <MillTCPM   DefaultValue="BC;Z;X;LCS;XY;Z"/>
    </DetailedApproachRules>
```

```
    <DetailedReturnRules> ... </DetailedReturnRules>
</SCType>
```

(реальный пример: `Machines\LatheMilling\MultiChannel\Puma MX2100ST\DOOSAN_PUMA_MX2100ST.xml`)

Выбирайте тип, соответствующий кинематике ветви; один станок может смешивать типы среди своих субстанков.

6 Каналы управления (одновременная работа)

Канал — это независимый поток движения/УП, выполняющийся **одновременно** с другими. На многоканальном станке (двухшпиндельные / многоревольверные токарные, токарно-фрезерные центры, станки швейцарского типа) разные узлы, оси и параметры состояния принадлежат **разным каналам**, а некоторые оси **общие**. Принадлежность вы назначаете целочисленным членом `Channel` на соответствующих узлах/осях/параметрах состояния (и `ChannelWorkpiece`, где сторона заготовки отличается):

```
<MachineStateParameters>
    <SCType ID="AxisX1Pos" type="TMachineStateParameter"><Channel DefaultValue="0"/>
... </SCType>
    <SCType ID="AxisX2Pos" type="TMachineStateParameter"><Channel DefaultValue="1"/>
... </SCType>
</MachineStateParameters>
```

`Channel="0"` и `Channel="1"` помещают два поперечных суппорта на разные каналы; ось, оставленная на общем канале, является общей. Многоканальные станки — **не** только швейцарского типа — см. поставляемые примеры в `$(SCHEMAS_FOLDER)\LatheMilling\MultiChannel` (например, *IndexG160*, *Puma MX2100ST*, *PumaTT2000SY*). `SwissTemplate.xml` — один распространённый готовый частный случай (2-канальный швейцарский токарный), который также показывает селекторы `TCaseNode` револьверной головы/адаптера в контексте; станки, построенные на нём — например, токарные станки швейцарского типа Hanwha в `$(SCHEMAS_FOLDER)\LatheMilling\SwissType` — реальные примеры использования этого шаблона.

> **Коротко:** *каналы* работают **одновременно** и говорят, *какому независимому потоку* принадлежит узел; *субстанки* используются **попеременно** и говорят, *какое подмножество узлов/осей образует одну решаемую цепочку инструмент+заготовка*. Одноканальный станок всё равно может нуждаться в субстанках (фрезерный XYZ+ABC выше); многоканальный токарно-фрезерный использует оба — субстанки внутри каждого канала.

Назад к **Дескрипторам станков** | указатель станков

3D-модели станков

Кинематическая схема станка — это дерево узлов (Дескрипторы станков); каждый узел может нести **3D-модель**, которая отрисовывается — и проверяется на столкновения — в положении этого узла в кинематической цепочке. Эта страница объясняет, как эти модели подключаются, готовятся и выравниваются. XML-сторона мала (три вещи: `ImageFile`, `VisMatrix`, `Use3DModelColors`); остальное — про корректную подготовку геометрии.

1 Прикрепление модели к узлу

Модель узла именуется его свойством **`ImageFile`** (унаследованным от `TMachineNode`) и позиционируется для отображения матрицей **`VisMatrix`** внутри `VisualProperties`:

```
<SCType ID="Column" Caption="Column" type="TMachineNode">
    <ImageFile DefaultValue="Images\column.osd"/>
    <VisualProperties>
        <VisMatrix>                          <!-- подстраивает только модель, не кинематику
-->
            <SCType ID="T1" type="TRotateZ" DefaultValue="90"/>
        </VisMatrix>
        <Color><R DefaultValue="0.4"/><G DefaultValue="0.4"/><B
DefaultValue="0.45"/></Color>
    </VisualProperties>
</SCType>
```

Узел отрисовывается в **своей кинематической системе** (произведение цепочки `Matrix` от корня до него, см. §1), а затем **`VisMatrix`** применяется **поверх, только к модели**. Так что если CAD-модель экспортирована в неудобной ориентации, поправьте её через `VisMatrix`, не нарушая то, как узел движется.

> **Один узел = своя геометрия.** Каждый движущийся узел должен нести **только** ту геометрию, что движется вместе с ним. Поэтому правильно подготовленный станок — это **сборка** отдельных моделей (по одной на узел), а не единая монолитная модель. Именно это позволяет частям корректно двигаться друг относительно друга.

2 Где лежат файлы моделей

- Файлы моделей обычно лежат **рядом с XML-файлом станка** или в подпапке **`Images\`** для порядка, и на них ссылаются **относительными путями** (например, `Images\column.osd`). То же относится к растровым изображениям, показываемым в библиотеке станков.
- Поэтому полный станок — это **папка из нескольких файлов**: XML-дескриптор плюс его модели и изображения.

- **Держите папку самодостаточной и переносимой. Не** ссылайтесь на файлы в папке соседнего станка — это сломается при перемещении или передаче станка. (Схемы станков хранятся, как папки, внутри проектов именно для того, чтобы проекты оставались переносимыми.)
- Эти папки **не** обязаны находиться под стандартными псевдонимами дистрибутива вроде `$(SCHEMAS_FOLDER)`: библиотеке станков можно указать сканировать любые пользовательские папки в поисках станков.

3 Единицы и система координат

- **Единицы должны совпадать с дескриптором станка.** Геометрию модели нужно сохранять в тех же единицах, что и *Measurements* станка — миллиметры для метрического станка, дюймы для дюймового — ровно как переносы `Matrix` (см. системы координат и единицы). Если активная система единиц отличается, система пытается масштабировать станок один раз при загрузке, но для сильно параметризованных схем это может сработать некорректно — поэтому **готовьте модели в тех единицах, в которых будет использоваться станок**.
- **Система координат.** Геометрия показывается в глобальной СК, в которой была нарисована, в сочетании с матрицами иерархии узлов и затем `VisMatrix`. Иными словами, моделируйте деталь в мировой СК станка (Z вверх, X вправо, Y от наблюдателя — системы координат) и используйте `VisMatrix` только чтобы поправить модель, вышедшую несовмещённой.
- **Часто это вопрос нулевого положения, а не `VisMatrix`.** Модель часто выглядит «не так» просто потому, что ось, на которой она сидит, была смоделирована в **ненулевом положении**. Тогда `VisMatrix` обычно **не** нужен — просто задайте у этой оси `DesignTimeAxisValue` (справочник узлов) равным положению, в котором была нарисована модель, и узел (модель *и* кинематика вместе) выровняется корректно во время проектирования. Прибегайте к `VisMatrix`, только когда сама модель повёрнута/смещена относительно системы своего узла.

4 Форматы OSD и STL

Принимаются два формата:

- **OSD** (*OpenGL Stream Data*) — собственный формат системы и предпочтительный. Это триангулированная сетка, хранимая как поток OpenGL-примитивов/команд, и в отличие от STL может нести также **рёбра и нормали поверхностей** (для более красивой отрисовки) и **дополнительные точки привязки/ключевые точки** (например, центры окружностей из исходного CAD, которые иначе теряются при триангуляции).
- **STL** — распространённый обменный формат сеток; тоже поддерживается, но без дополнительной информации о рёбрах/нормалях/точках привязки.

Как получить OSD

1. Импортируйте CAD-модель (система читает многие распространённые CAD-форматы) в CAM-систему.

2. На странице **Model** выберите нужный узел (подпапку).
3. Используйте **Save as** и выберите формат **OSD**.

Тот же экспорт доступен через **CAM API** и **CAM IPC** (см. репозиторий `cam-api-examples`), так что подготовку моделей можно автоматизировать скриптами.

5 Визуальная и коллизионная геометрия — держите её лёгкой

Сейчас **одна и та же модель используется и для отображения, и для проверки столкновений** (в будущем может измениться). Значит, детализация модели — это компромисс между видом и скоростью — **предпочитайте скорость**: не стройте схемы из чрезмерно детализированного CAD. И CAM-система, и MachineMaker предоставляют инструмент, который интерактивно помогает удалить невидимые, внутренние или очень мелкие поверхности, чтобы упростить модель. (Фиксированного норматива по числу треугольников нет; проще — лучше.)

Это также связано с настройками *Simulation — Revolution bodies simulation* и *Collisions to ignore* — описанными в Machine Setup → Симуляция.

6 Цвета — `Use3DModelColors`

`VisualProperties` принимает необязательный логический атрибут **`Use3DModelColors`**:

- **`False`** (по умолчанию) — узел отрисовывается с `Color`, заданным в его `VisualProperties` (побеждает XML-дескриптор).
- **`True`** — узел отрисовывается **цветами, хранящимися в самой 3D-модели**.

```
<VisualProperties Use3DModelColors="True">
    <!-- этот узел сохраняет цвета, запечённые в его .osd-модели -->
</VisualProperties>
```

7 Готовые модели

У **MachineMaker** есть онлайн-библиотека устройств, роботов, станков и ячеек. Она поставляет **готовые наборы** (модели *плюс* дополнительную информацию дескриптора), а не отдельные файлы моделей, и вставляет их в описание ячейки за вас — их нельзя загрузить как отдельные файлы. Поэтому, когда вы готовите схему **вручную**, вы обычно готовите модели индивидуально для этого станка, получая самодостаточную переносимую папку, описанную в §2.

Назад к **Дескрипторам станков** | указатель станков