Руководство разработчика базы данных СПДС GraphiCS

Оглавление

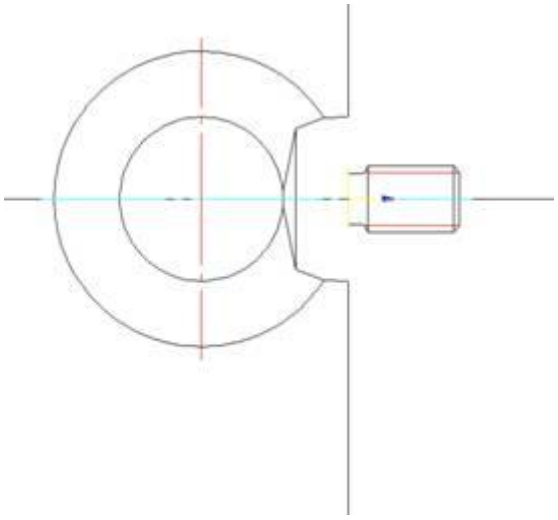
Функции, выполняемые стандартными объектами 4
Что умеют стандартные? 4
Требования, предъявляемые к стандартным деталям 7
Абстрактная модель детали 8
Двухмерная графика объекта 10
Создание образмеренных видов стандартных деталей 12
Детали в структуре БД. Зависимости 16
Контур подавления и порядок следования 18
Событийная модель объекта 19
Механизм локализации 22
Перечень требований. Оформление графики и написание скрипта объектов базы данных стандартных элементов с помощью MechWizard 23
Свойства объекта 23
Параметры 24
Скрипт 25
Оформление двухмерной графики 27
Общие требования 28
Слои 28
Размещение графики 29
Нанесение размеров 30
Нанесение штриховки 30
Параметры 30
Прочие требования 31
Синтаксис языка скриптов 31
Идентификаторы 31
Ключевые слова 31
Типы данных 32
Комментарии 33
Операторы 33
Пустой оператор 33
Составной оператор 34
Оператор присваивания 34
Оператор доступа к внутренним членам сложных типов данных 34
Арифметические и логические операторы 34
Условный оператор 34
Оператор цикла 35
Функции 36
Задание плоскостей объектов 36
Работа с таблицей 38
Диалог объекта. Оформление функции UniDialog 39
Подключение пользовательской формы 40
Установка зависимостей 40
Функция ShowValue 42
Приложение 1. Список ключевых слов и зарезервированных переменных 42
Математические функции и функции работы с плоскостями 45
Функции обработки плоскостей 51
Функции для работы с таблицами 55
Диалоги объектов БД 56
Функции для работы с зависимостями 62
Функции - обработчики событий, вызываемые ядром приложения 64
Функции для работы с идентификаторами объектов 66
Другие функции 67
Объект - центровые отверстия для валов по ГОСТ 14034-74 68
Заглушка ГОСТ 16076-70 80
Фланцы стальные приварные встык ГОСТ 12821-80 82
Двутавры ГОСТ 19425 84
Скрипт двутавра: 85
Панели HB 90

Функции, выполняемые стандартными объектами

Цель использования деталей БД программного продукта- это, как правило, создание двухмерной и трехмерной графики, облегчающей работу конструктора. Отличительной особенностью стандартных является интеллектуальный характер поведения деталей на чертеже, что резко сокращает объем дополнительных действий над ними как в двухмерной, так и в трехмерной среде проектирования.

Что умеют стандартные?

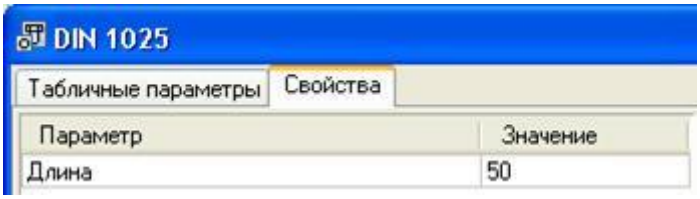
Вставка из базы. БД организована в виде иерархической структуры объектов, родственных по функциональному назначению. (Например, сгруппированы детали крепления, детали валов, арматура трубопроводов и т.д.). Кроме того, имеются отдельные виды объектов - группы, маркеры и ярлыки объектов. Кроме того, БД имеет поиск по именам стандартов. При вставке детали необходимо указать точку вставки и направление отрисовки локальной оси X детали (т.е. указать положение и ориентацию детали). Детали могут отображаться в процессе вставки полностью или в виде условного изображения начальной плоскости. Положение указывается одним щелчком, ориентация - при определенной точке вставки - вторым.



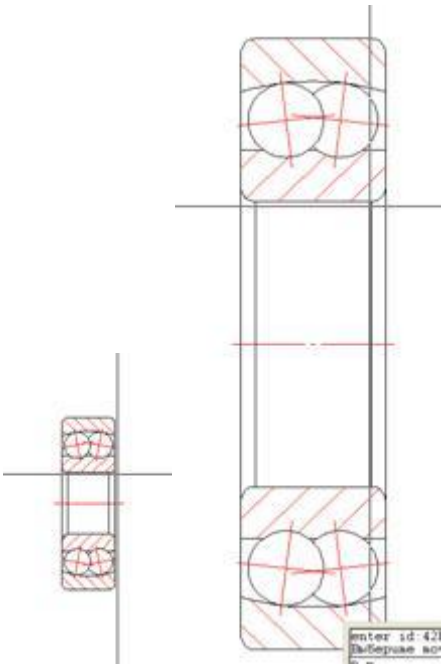
Если ГОСТ на деталь содержит таблицу параметров, то имеет смысл ограничить размеры конструктивных элементов графики значениями из таблицы. Таких таблиц может быть несколько для одной детали. (Например, если параметры одного из конструктивных элементов не зависят от других элементов - ряд фасок или скруглений не зависит от типоразмера детали крепления).

Например, стандарт на стальные двутавры

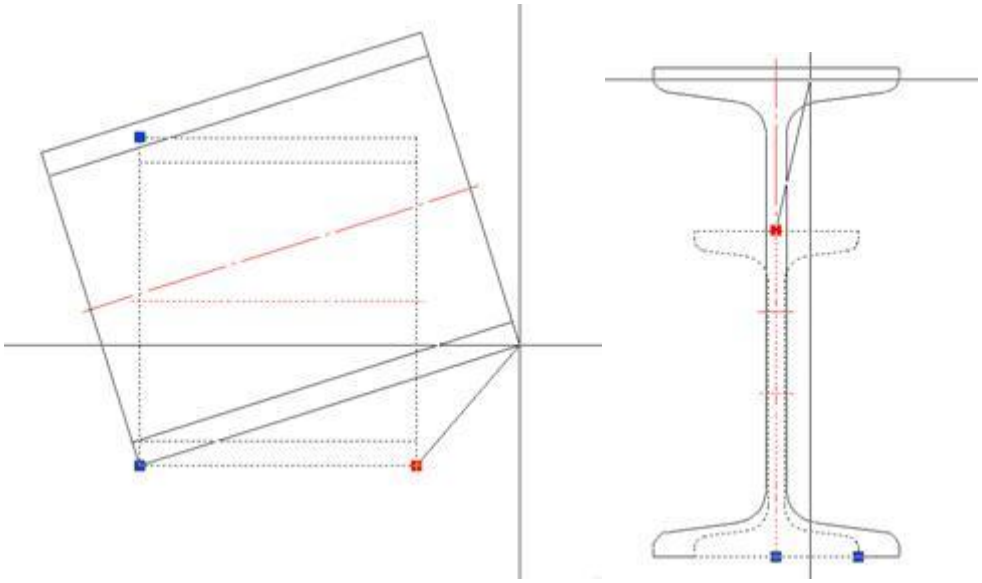
Если параметры графики объекта не заданы ГОСТом или рекомендациями, то их значение указывает пользователь при вставке объекта. Они отображаются на второй закладке свойств диалога вставки. Такими параметрами, например, является длина двутавра.



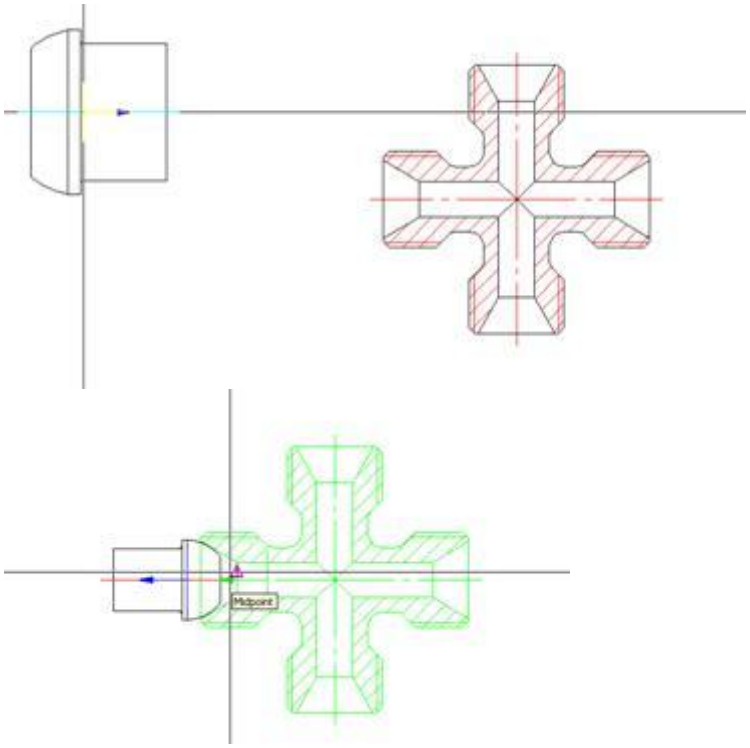
В ряде случаев удобно указывать значение параметров детали не ручным вводом или выбором из таблицы, а визуально на чертеже, относительно других элементов оформления. В этом случае используется динамический выбор параметров детали. Он включается переключателем *Динамический выбор параметров*. Как правило, параметры выбираются после указания точки вставки по положению курсора относительно точки вставки. Динамический выбор может осуществляться из табличных и произвольных параметров. Иногда на динамически выбираемые значения накладываются дополнительные ограничения (вроде максимального и минимального значения) для обеспечения корректной отрисовки детали.



Поскольку ручки являются мощным инструментом редактирования объектов, то наличие ручек дает объектам дополнительный функционал. Помимо того, что за них можно объект перемещать, поворачивать, отражать, с помощью ручек можно изменять типоразмер детали и задавать значения параметров детали визуально, а также определять направление отрисовки детали.

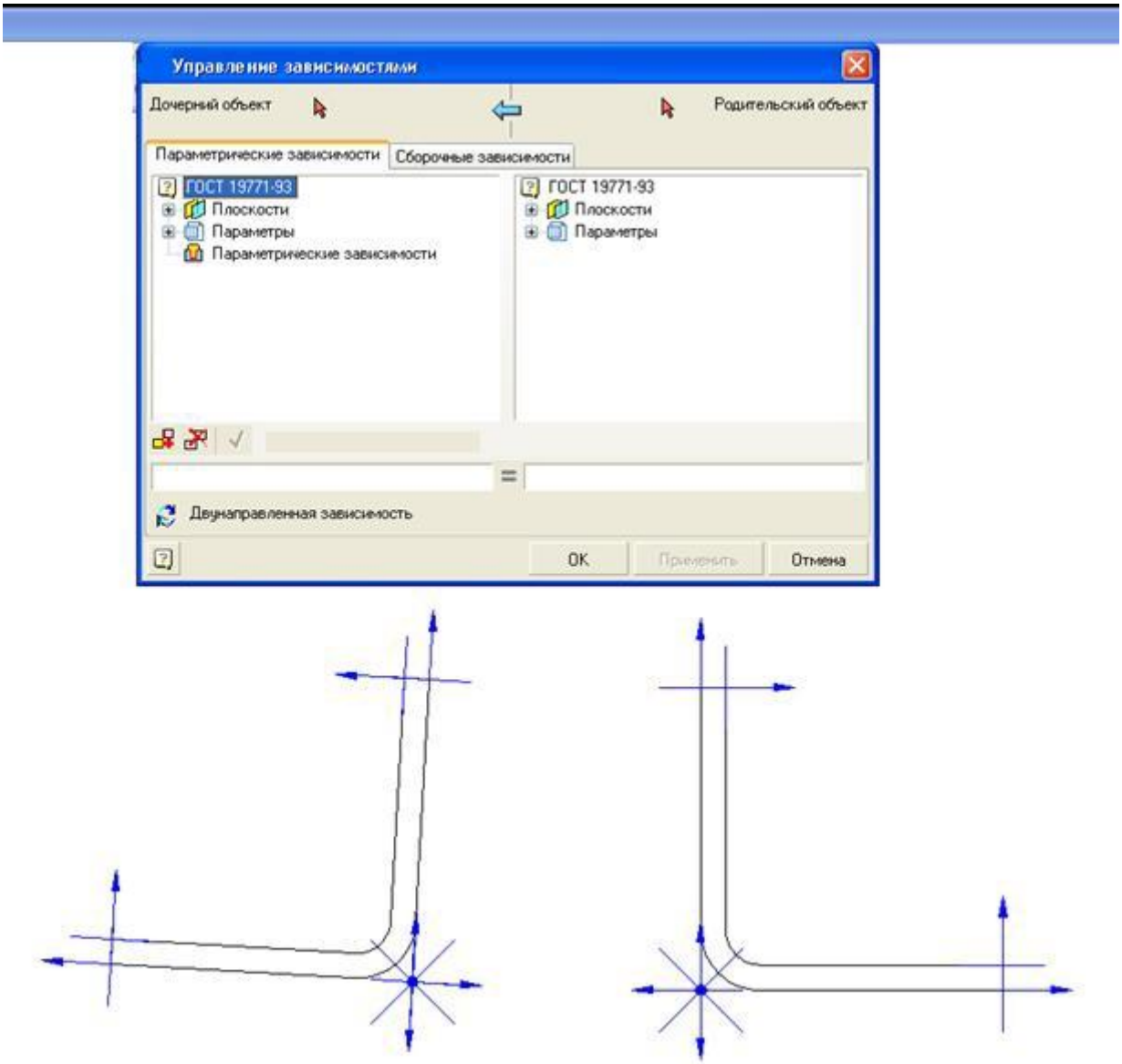


Отличительной особенностью стандартных БД программного продукта является их интеллектуальный характер поведения по отношению к другим объектам, уже присутствующим на чертеже. Например, при вставке из базы арматуры трубопроводов по внутреннему конусу крестовины, вставляемый вслед за ней ниппель будет автоматически к ней привязываться.



Реализуется это посредством установки параметрических и геометрических зависимостей между деталями. Разумеется, зависимости должны устанавливаться только для деталей в соответствие со стандартом.

При желании пользователь может сам устанавливать зависимости на уже находящиеся на чертеже объекты (например, если автоматический коннект деталей не предусмотрен) - это пользовательские зависимости, которые хранятся в чертеже.



Кроме выше описанных функций, стандартные БД могут выполнять дополнительные функции, которые необходимы для использования их в различных инструментах (Вставка болтового соединения, диалоги арматуры трубопроводов).

Требования, предъявляемые к стандартным деталям

Обеспечение требуемого функционала означает, что детали должны вставляться из БД, их параметры должны задаваться, меняться, детали должны коннектиться и т.п.

Детали должны *соответствовать стандартам* (ГОСТ, ОСТ ИСО и т.д.) и инженерной практике. То есть, помимо значений табличных параметров, необходимо соблюдение требований оформления по ЕКСД, СПДС и т.п. Помимо этого, отрисовываемые детали не должны нарушать общепринятых принципов проектирования и оформления двухмерной и трехмерной документации на изделие.

Унификация. В целях упрощения разработки и классификации стандартных деталей необходимо выполнять сходные стандартные с

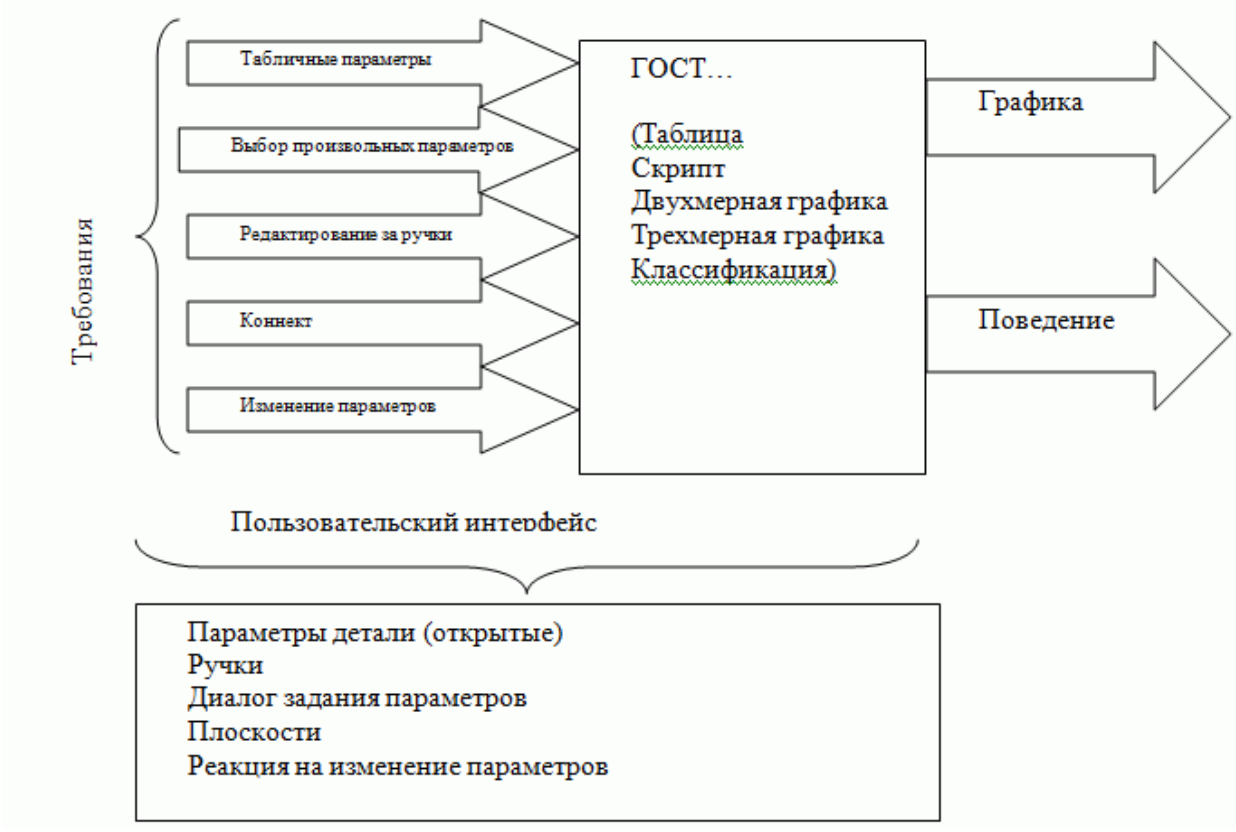
помощью одинакового оформления графики и , что очевидно, одинаковые по графике детали должны функционировать, как правило, одинаково.

Поскольку база данных стандартных используется в мультиязычных приложениях, то она должна быть *локализуемой*. Т.е., части стандартных относящиеся к пользовательскому интерфейсу, должны корректно отображаться как в русском, так в английском и немецком языке.

Часто возникает ситуация, при которой один ГОСТ разрабатывают несколько человек. Унификация в первую очередь позволяет передавать разработку между разработчиками и пользователями. Во вторую очередь, подход к разработке и оформление скриптов должны быть понятными для всех участников. (Это требование *прозрачности* очень актуально при доработке стандартных деталей)

Абстрактная модель детали

Для пользователя деталь должна представлять собой черный ящик, реализующий определенный выше функционал.



Задавать типоразмер детали или ее графику необходимо с помощью параметров (например, для болта - диаметр резьбы и длина стержня). Поэтому необходимо определить эти параметры. Но не все параметры графики должны быть доступны пользователю (например, размер фаски или радиусы закругления не будут существенными для болта), но они должны быть известны для определения графики. В этой связи различают открытые (Public) параметры - это видимые пользователем и закрытые, защищенные (Protected) параметры, которые определяют

графику, но пользователю невидимы и недоступны. Очевидно, что параметры эти могут быть как табличными (т.е. иметь дискретные, определенные ГОСТом значения) или произвольными.

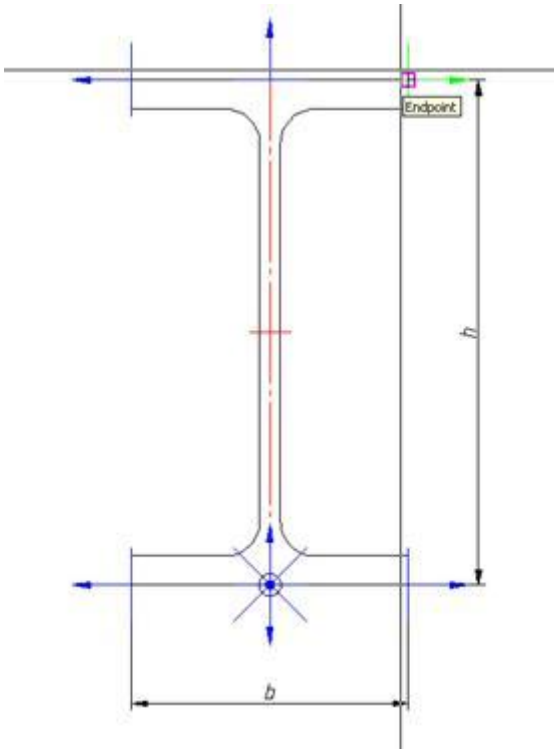
При редактировании за ручки должно быть определено общее их количество, какие из параметров детали они задают, и возможно, какие из ручек могут задавать точку вставки и ориентацию детали.

В пользовательский интерфейс детали также входит вид диалога вставки (или форма данных). Его вид и поведение также входит во «внутренности» черного ящика детали, понятно, что форма или диалог должны обеспечивать адекватность и удобство выбора типоразмера детали и задания ее параметров.

Установка геометрических зависимостей обеспечивается плоскостями детали, определенными как совокупность точки и вектора. Количество и положение этих плоскостей с точки зрения установки зависимостей также должно быть определено. Кроме того, плоскости могут быть как изменяемыми т.е. определяющими положение детали при воздействии зависимости от другой детали, так и неизменяемыми (если к этой плоскости присоединяется другая деталь), не меняющими положение точки вставки детали по действию зависимостей.

Плоскости определяются на основании локальной системы координат детали. Точка вставки (pntOrigin) является ее началом. Вектор ориентации (vecDirection) и ортогональный ему вектор в плоскости чертежа (vecPlane) соответствуют осям абсцисс и ординат. В трехмерном пространстве вектор аппликат определяется как векторное произведение vecDirection на vecPlane.

Математически подсвечиваемая плоскость будет определяться точкой, находящейся на расстоянии b/2 от точки вставки по горизонтали и расстоянии h по вертикали. Вектор этой плоскости будет коллинеарен вектору направления vecDirection. Очевидно, что этих данных достаточно для определения нового положения детали при изменении плоскости по зависимости.



Помимо перечисленных особенностей абстрактной модели, детали при чистом изменении параметров (например, при действии параметрических зависимостей, или при указании значения параметра непосредственно из диалога свойств) необходимо определить, каким конкретно образом присваивать новые значения для параметров, как выбирать из таблицы новые значения, менять плоскости и т.п.

Выставка дискретных значений параметров детали производится с помощью таблицы. Она организуется таким образом, что одной записи (строке) таблицы соответствует один типоразмер. Поля (столбцы) таблицы содержат параметры детали. Параметры могут иметь строковый, целый и действительный типы. Параметры имеют комментарий, который используется в диалоге вставки. Поля и строки таблиц имеют уникальные идентификаторы, сохраняемые при импорте и экспорте таблицы для обеспечения соответствия типоразмеров уже имеющихся на чертеже деталей (наличия выбранных строк и параметров в таблице при редактировании объекта).

Двухмерная графика объекта

Для отображения объекта на чертеже вводятся ортогональные виды. Виды различаются, помимо ориентации, еще по стилю отображения на обычные, виды с разрезом, упрощенные и виды с размерами. Виды с размерами являются системными и скрытыми. Их графика реализуется при вставке образмеренного вида из контекстного меню.

Локальная система координат в двухмерных видах задается с таким расчетом, что на фронтальном виде локальная ось абсцисс направлена строго вправо, ось ординат - вверх. На остальных видах направления координатных осей определяются проекционными зависимостями.

Параметрическое распознавание вида заключается в определении параметров и относительного положения (горизонтальность, вертикальность, параллельность, перпендикулярность) графических примитивов, а так же их взаимодействия (точек пересечения, касательностей и т.д.). Кроме того, необходимо задать положение точки вставки детали (локального начала координат). Это необходимо для построения текстового описания вида (параметризация графики).

Рекомендации к построению двухмерных видов.

- Использовать объектную привязку к концам отрезков и пересечениям.
- Использовать полярное отслеживание или ортогональность.
- Скругления и касательные дуги должны быть образмерены.
- Использовать вспомогательную геометрию, облегчающую распознавание (окружности в центрах касательных дуг, линии горизонтального и вертикального отношения узлов).
- Размеры следует наносить таким образом, чтобы их трактовка была однозначной.
- Иногда, при создании сложной графики, применяют разбиение вида по частям. И в последствии устанавливают параметр нулевого расстояния между частями.

Следует помнить, что цель распознавания - определение положения всех примитивов относительно точки вставки. От нее начинается определение параметров и отсчет размеров.

Требования к оформлению графики изложены в соответствующих рекомендациях.

Для того, чтобы двухмерная графика была параметрически управляемой, на размерах вместо номинала выставляются соответствующие геометрические параметры детали. Кроме того, некоторые параметры относятся к самой графике.

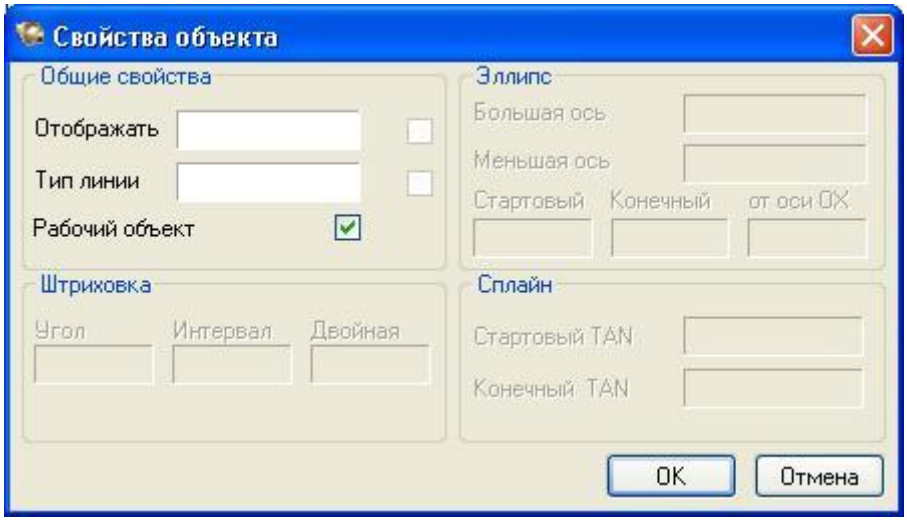
Пример чертежа

Общие свойства: «Отображать» - логическое выражение или переменная, определяющая, отрисовывать или нет данный примитив.

Тип линии:

- 0 - основная.
- 2 - тонкая.
- 4 - штриховая.

Флажок «Рабочий объект» - объект участвует в распознавании, но в итоговую графику не включается. Это вспомогательные построения.



Параметры штриховки - угол наклона, шаг штриховки и признак двойной штриховки (логическое условие)

Параметры эллипса - бОльшая полуось (параметр размера полуоси в направлении локальной для эллипса оси Ox), меньшая полуось (тоже в направлении оси Oy). Стартовый и конечный углы эллипса - если отрисовывается не весь эллипс, а только эллиптическая дуга. Угол наклона относительно оси OX - это угол, на который поворачивается локальная система координат эллипса относительно системы координат вида объекта.

Параметры сплайна - стартовый угол касания и конечный угол касания. Все точки сплайна при этом должны быть образмерены.

Если при вставке вида **контур штриховки** не следует предполагаемым границам, то создается замкнутая полилиния, на которую устанавливается параметр - контур штриховки.

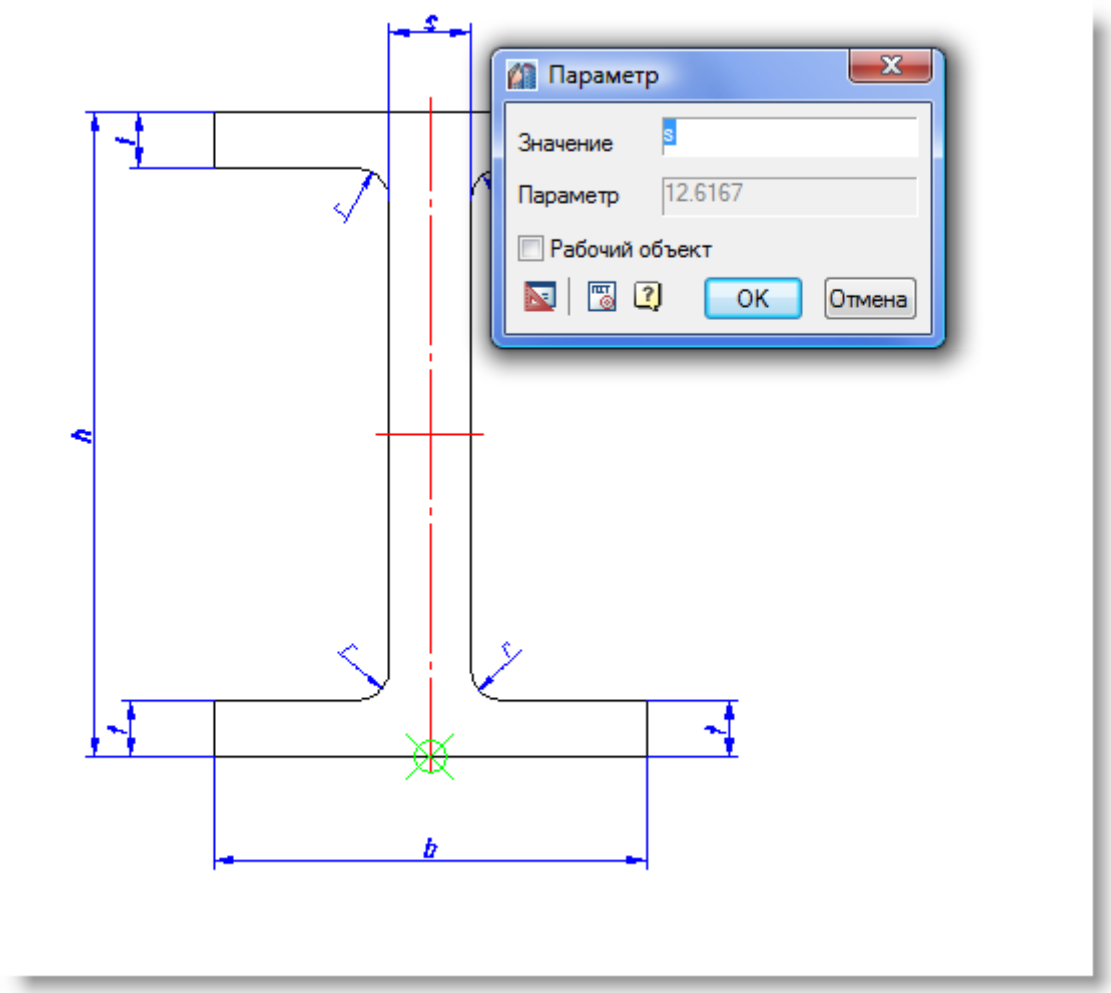
Иногда также возникает необходимость установить принудительно **контур подавления** для детали (если автоматически создаваемый контур не соответствует требованиям, предъявляемым к детали). В этом случае, аналогично, на замкнутую полилинию устанавливается параметр контур подавления.

Создание образмеренных видов стандартных деталей

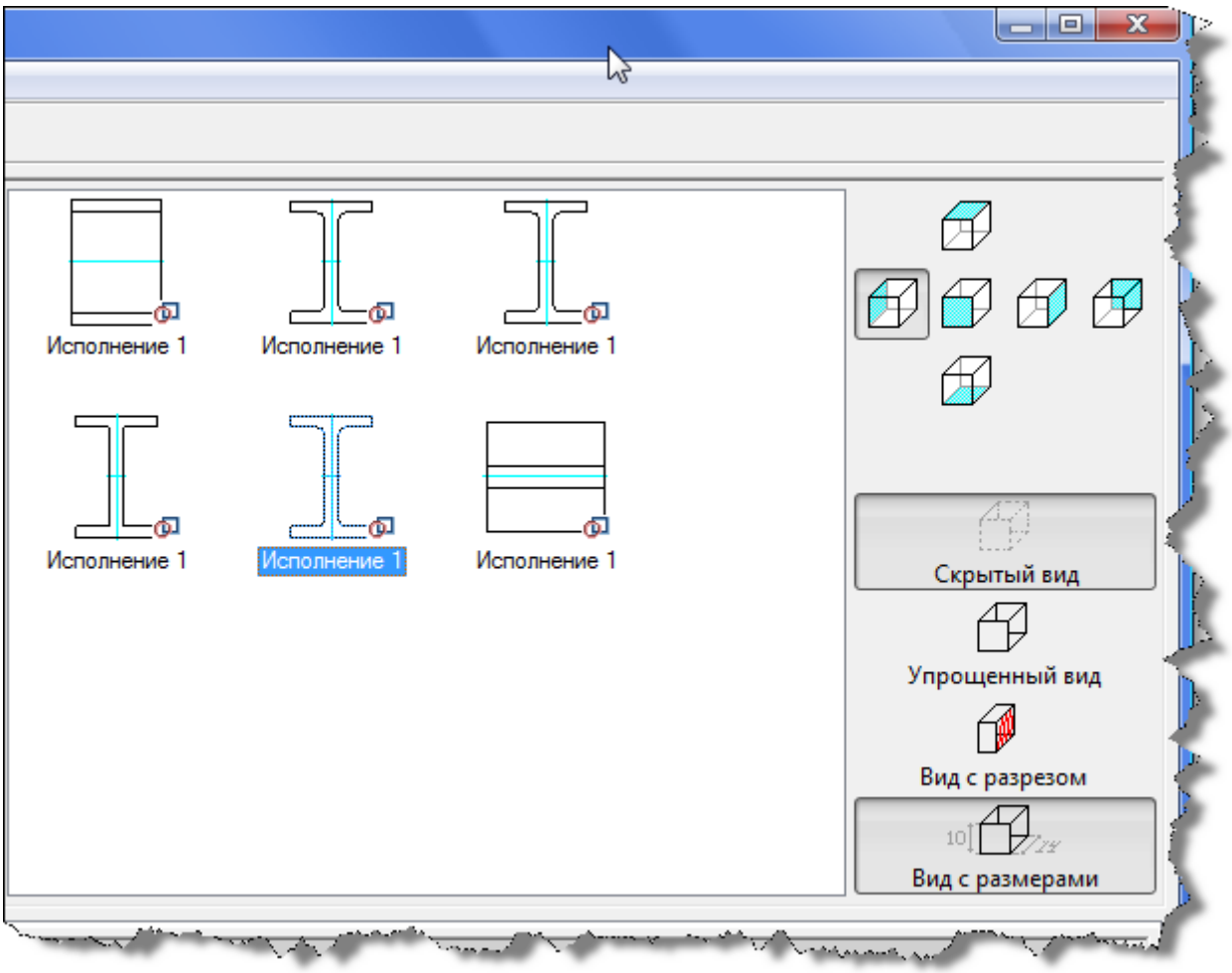
Для создания образмеренных видов стандартных деталей используется та же графика, что и для распознавания основного вида.

В графику образмеренного следует включать только те элементы, которые необходимы.

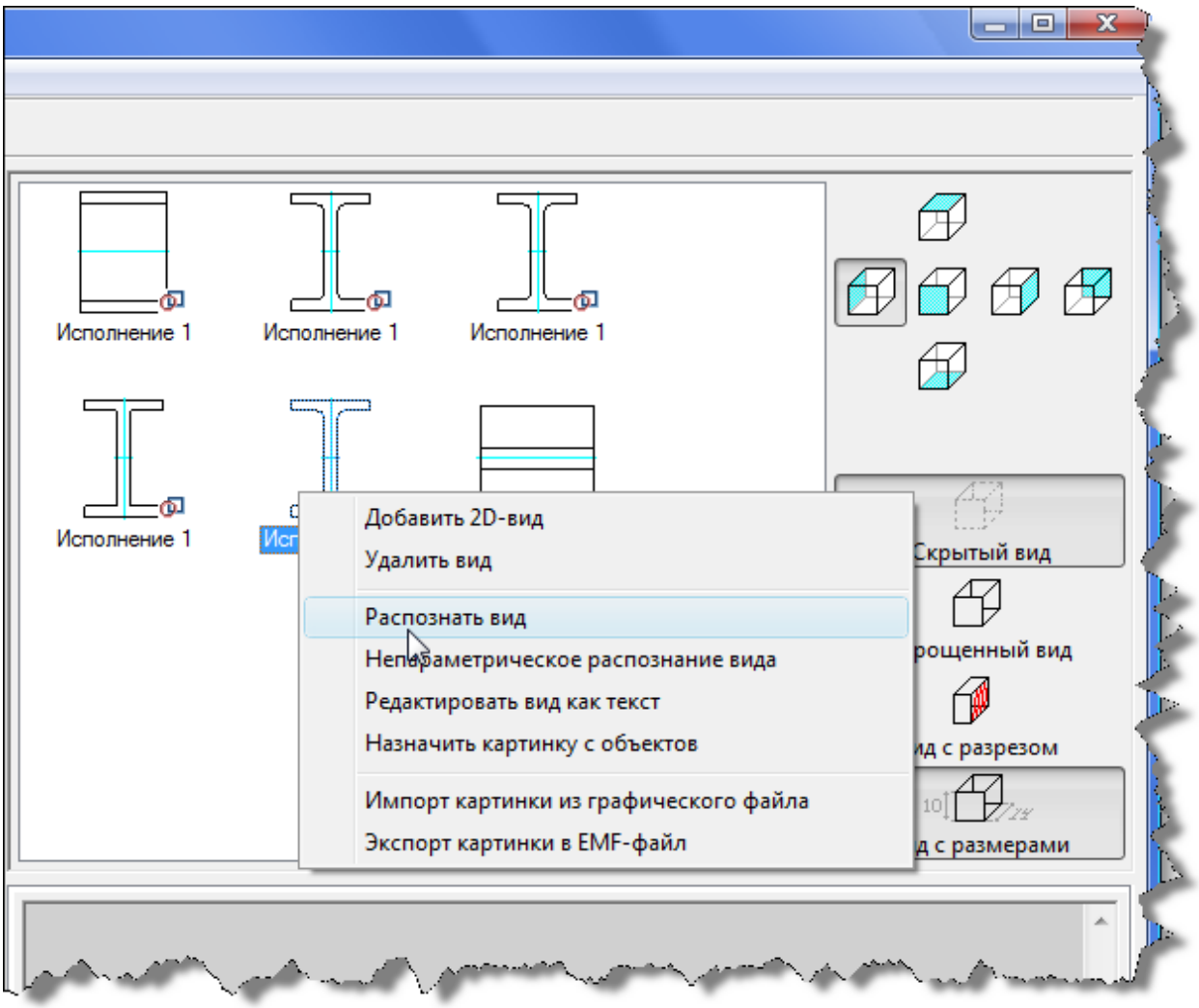
Для того, чтобы размеры понимались при распознавании как конструктивные объекты, с них необходимо снять флажок *Рабочий объект* с помощью инструмента **Установить параметр**.



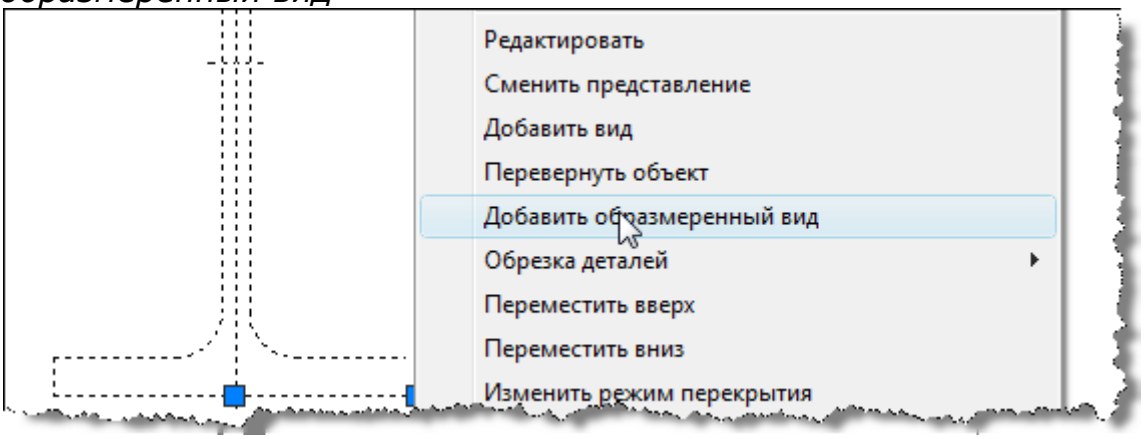
Далее нужно создать вид в необходимом исполнении объекта и задать для него свойства *Скрытый вид* и *Вид с размерами*.



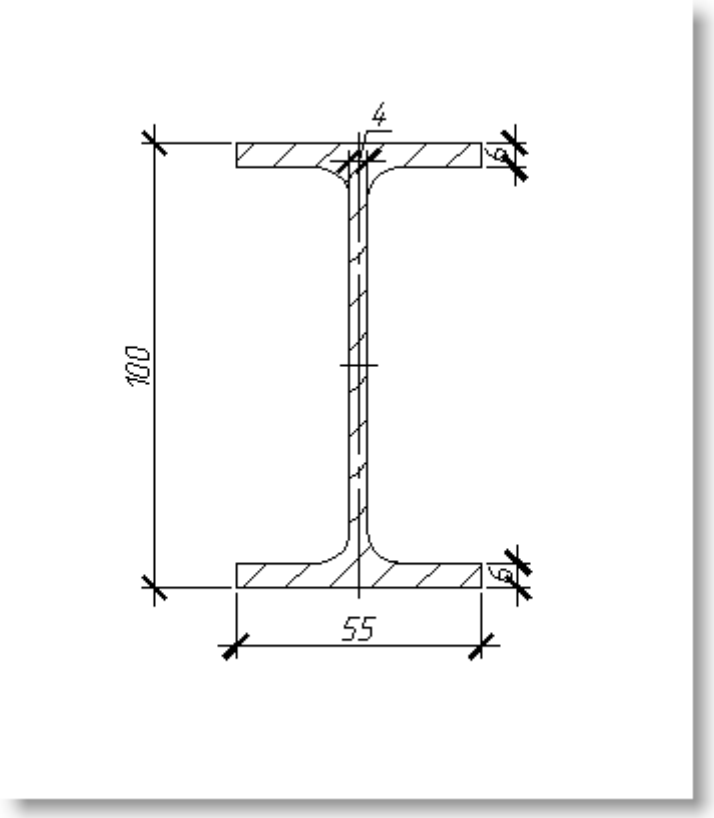
После этого в контекстном меню вида выбрать *Распознать вид*.



Указать графику на чертеже и подтвердить выбор. Сохранить объект. Для тестирования образмеренного вида нужно вставить объект на чертеж с тем направлением вида, для которого был создан вид с размерами. В контекстном меню этого вида выбрать опцию *Добавить образмеренный вид*



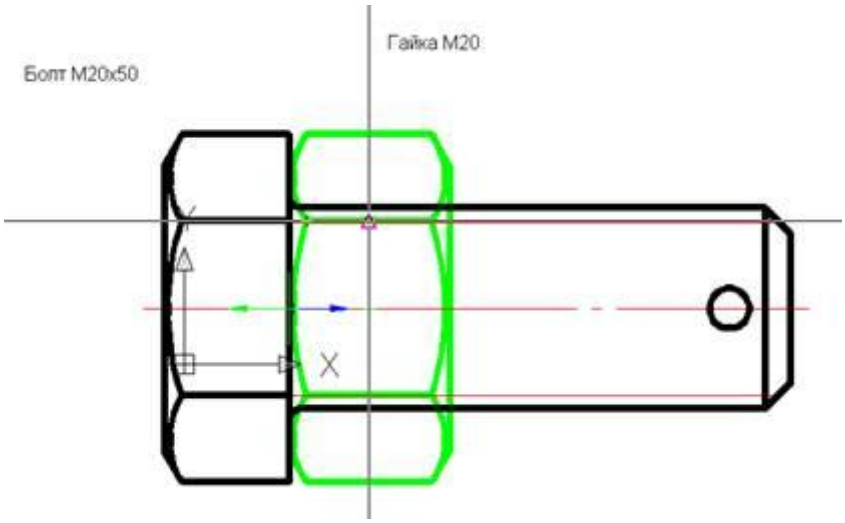
И указать точку вставки образмеренного вида на чертеже.



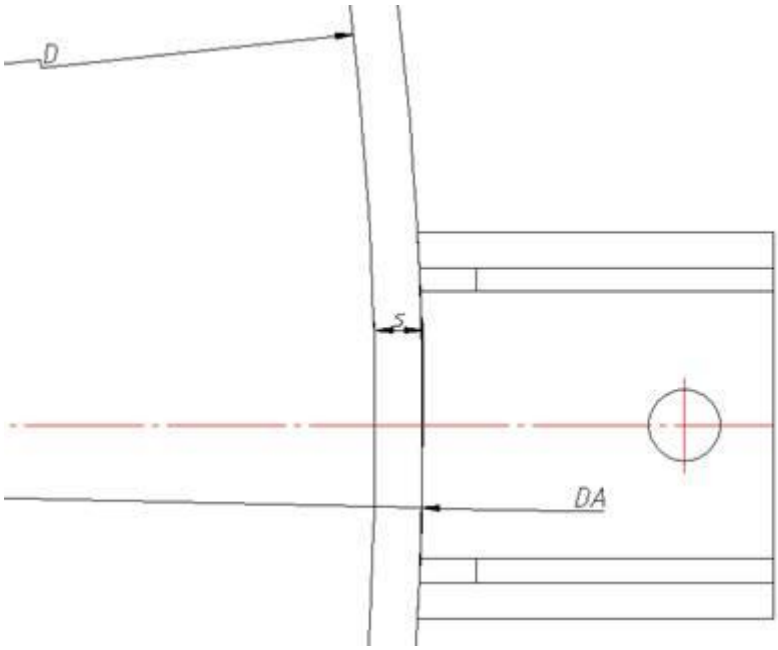
Образмеренные виды вставляются на чертеж с текущим размерным стилем и масштабом символов.

Детали в структуре БД. Зависимости

Итак, детали умеют присоединяться друг к другу. Каким образом это происходит? Путем установки параметрических и геометрических зависимостей. Параметрическая зависимость - это выражение, устанавливающее равенство параметра одного объекта параметру или арифметическому выражению от параметров другого объекта. Например: диаметр резьбы болта должен быть равен диаметру резьбы гайки, к которой он присоединяется.



Т.е. параметр «диаметр резьбы» (dr) гайки приравнивается параметру «диаметр резьбы» (dr) болта (dr=dr).



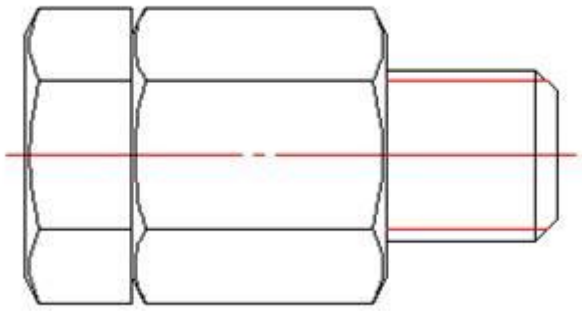
При установке опоры на обечайку устанавливается зависимость на арифметическое выражение параметров, т.е. диаметр опоры приравнивается сумме двух толщин обечайки и внутреннего диаметра обечайки. DA=D+2*s.

При установке зависимостей различают родительский и дочерний объекты, таким образом, зависимости могут быть однонаправленными и двунаправленными. Однонаправленные зависимости действуют только на дочерний объект (изменение родительского объекта вызывает изменение дочернего, но изменение дочернего не влияет на родительский объект), двунаправленные же зависимости действуют на оба объекта, т.е. изменения в одном объекте вызывают изменения в другом объекте и наоборот.

Геометрические зависимости между объектами реализуются путем перемещения и переориентации объектов. Новое положение объекта вычисляется на основании новых положений плоскостей объекта.

Различают следующие виды геометрических зависимостей:

Вставка (INSERT) - совмещение точек вставки и векторов плоскостей. Функция имеет параметр – расстояние вставки (offset) – это расстояние между точками плоскостей деталей вдоль оси абсцисс.



Совмещение по плоскости (MATE) - коллинеарность векторов направления при равенстве относительной абсциссы деталей. Имеет также параметр - расстояние по оси абсцисс.

Совмещение по оси (AXIS) - совмещение векторов направления при не фиксированной относительной координате по оси абсцисс. Параметр зависимости - расстояние по оси ординат.

Совмещение по направлению (угловая зависимость) - коллинеарность векторов направлений при нефиксированных относительных координатах.

Геометрические зависимости могут быть сонаправленными и противонаправленными в зависимости от ориентации локального вектора абсцисс.

Поскольку привязывать все объекты ко всем объектам нет смысла, то необходимо вводить условия на автоматическую установку зависимостей. Для корректного определения того, какая деталь, к какой должна коннектиться, вводится классификация объектов. Классификация строится на основании числового класса (используется в инструментах программы) а также открытых параметров объекта. В частности, это переменные strTheName, strTheType, strTheSubType - Имя, фамилия и отчество объекта. При присоединении болта к гайке производится проверка по имени-фамилии-отчеству и если они совпадают, то зависимость устанавливается. Т.о. детали крепления соединяются только с деталями крепления, а не с арматурой трубопроводов, хотя и те, и другие имеют параметр «диаметр резьбы». Кроме этого, каждый объект имеет уникальный идентификатор, по которому именно он выбирается при вставке из браузера.

Контур подавления и порядок следования

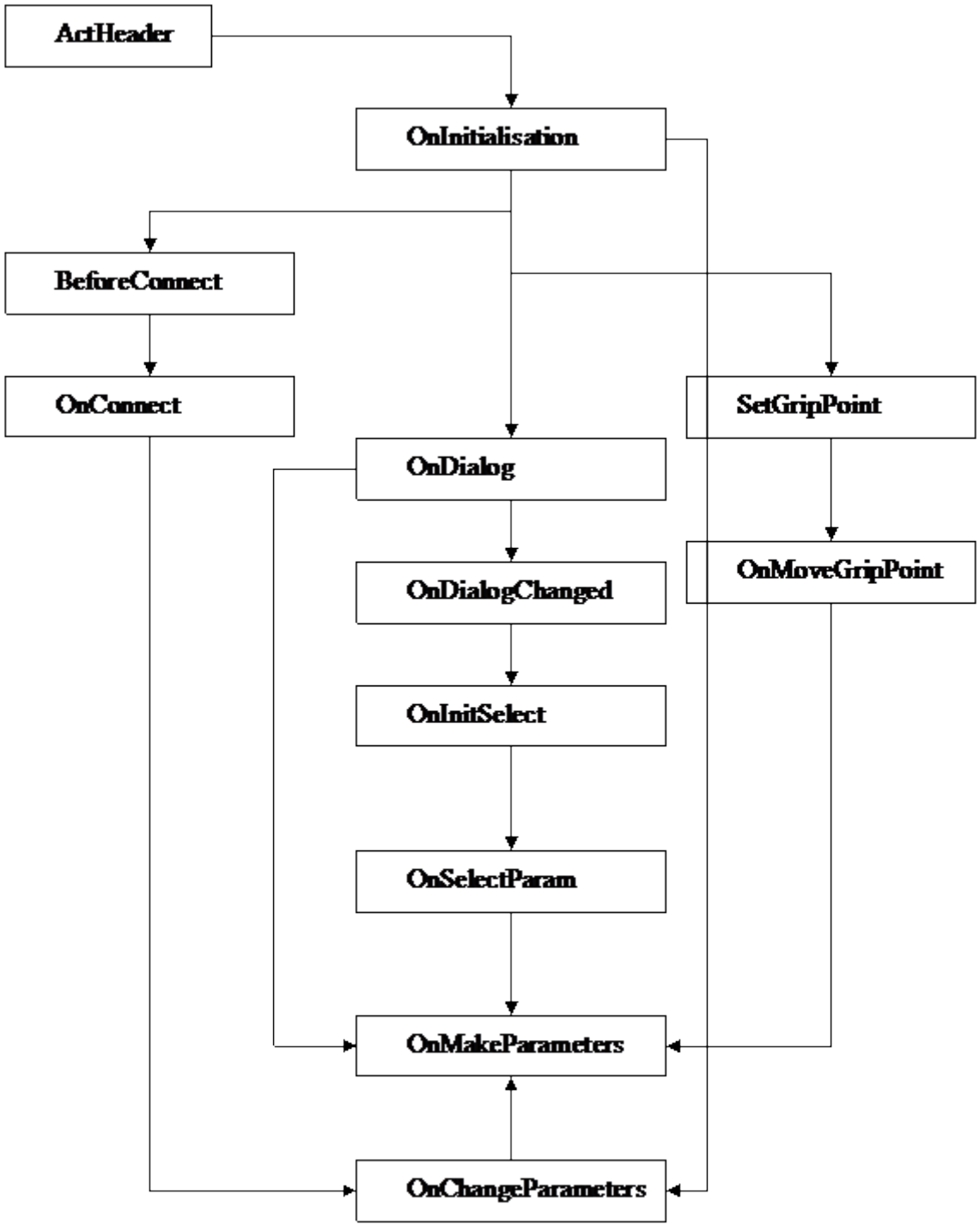
Графика стандартных базы данных приложения имеет контур подавления - это замкнутая область, скрывающая графические примитивы и другие детали. (Например, контур гайки перекрывает линии болта).

Порядок перекрытия определяется переменной rZOrder - Порядок следования. Детали с большим значением rZOrder перекрывают своим контуром подавления детали с меньшим значением rZOrder. Но эта

особенность реализуется только в пределах интервалов, кратных 2000. Объект базы данных приложения может иметь контур подавления, очерченный его внешними границами, иметь назначенный отдельно контур подавления, либо вовсе не иметь никакого.

Событийная модель объекта

Блок-схема последовательности выполнения функций скрипта:



ActHeader - это описание параметров объекта. Сюда включаются описание открытых, закрытых параметров и задание зарезервированных переменных. Сюда же включается описание подобъектов.

При каждом обращении к объекту выполняется функция OnInitialization. Поэтому в ней необходимо вводить дополнительную проверку на однократность установки параметров объекта с помощью переменной seted.

Функция OnDialog вызывается после указания точки вставки и направления или в контекстном меню при динамическом выборе параметров. В ней описывается вид диалога вставки или вызывается форма объекта.

OnDialogChanged позволяет ввести интерактивность в вид диалога вставки объекта. Она выполняется каждый раз при изменении параметров в диалоге. В теле этой функции, возможно скрывать определенные контролы для обеспечения корректного выбора параметров объекта.

OnInitSelect выполняется перед началом динамического выбора параметров. В ней, как правило задается строка-подсказка (например, текущее значение длины и т.п.)

OnSelectParam выполняется при динамическом выборе параметров. В ней определяется каким конкретно образом какие параметры следует выбирать. Часто содержит проверки по текущему виду и номеру запроса на выбор параметров.

Функция OnMakeParameters, как правило, заключительная. В ней производятся все окончательные вычисления параметров, присваиваются классификаторы, задаются плоскости. Поскольку эта функция выполняется практически после всех событий объекта, сюда часто включают ограничения на параметры и условную выборку исполнений.

Функция SetGripPoint устанавливает ручки для объекта. В ней задается их количество и положение.

В функцию OnMoveGripPoint приходит номер редактируемой ручки NMovingGrip - здесь происходит обработка относительного положения ручек и изменение параметров объекта в соответствии с ним.

BeforeConnect выполняется перед установкой зависимостей. В ней сбрасываются предыдущие зависимости и задается строка подсказки при выборе объекта для присоединения.

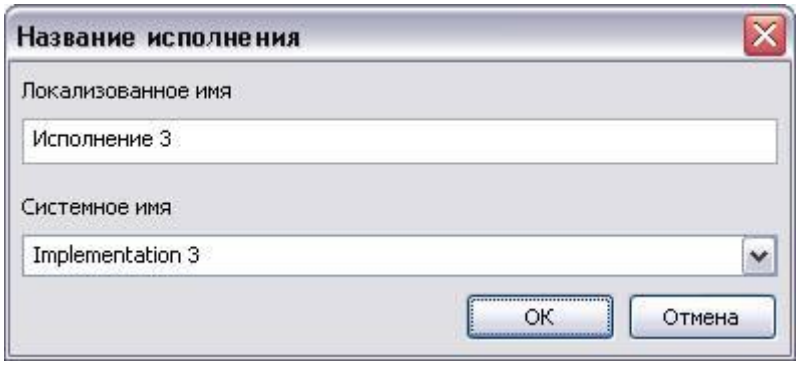
Функция OnConnect выполняется при присоединении одного объекта к другому. Содержит проверки по классификаторам объекта и операторы установки зависимостей в относительно результатов этой проверки.

Результатом OnConnect является новое значение параметров присоединенного объекта, приходящее в виде объекта new. Функция OnChangeParameters выполняется при изменении параметров объекта от другого объекта или вручную через панель параметров. (то есть извне объекта). Эта функция определяет характер реакции объекта на изменение его параметров.

Механизм локализации

В приложении реализован следующий механизм локализации:

Обращение к исполнениям объектов производится по их системному имени, которое может быть изменено в мастере объектов в контекстном меню исполнения.



Обращение к исполнению в скрипте:

```
     if (strDesignName == "Implementation 1") {
          strPartName = @BOLT_M+dr+step+" #
x"+L+@___GOST_7798_70;
     };
```

При необходимости использования в скрипте строковых значений в локальной кодировке, их следует заменять ресурсами.

Ресурс - это ссылка на строку таблицы, в которой содержится перечень строковых значений в зависимости от текущего языка приложения.

RID	Label	English	Russian	German	Polish	Czech
8	ROD_LENGTH	Rod length	Длина стержня	Länge	Długość trzpienia	Délka dříku
9	THREAD_DIAMET...	Thread diameter	Диаметр резьбы	Gewindedurchmesser	Średnica gwintu	Průměr závi
10	WORK_PLANE_1	Work Plane 1	Рабочая плоскость 1	Arbeitsebene 1	Płaszczyzna konstru...	Pracovní ro
11	SMALL_THREAD_...	Small thread step	Мелкий шаг резьбы	Feingewinde	Mały skok gwintu	Krátký závit
12	BOLT_OK	Bolt OK	Болт Ok	Bolzen OK	Śruba OK	Šroub OK
13	BOLT_HEAD_WID...	Bolt Head Width	Размер под ключ	Schlüsselweite	Szerokość łba śruby...	Šířka hlavy
14	THREAD_LENGTH	Thread Length	Длина резьбы	Gewindelänge	Segment gwintowany	Délka závitu
15	MASS	Mass	Масса	Masse	Masa	Hmotnost
16	SELECT_LENGTH	Select length	Выберите длину	Länge auswählen	Wybierz długość	Vyberte dél
17	BOLT_M	Bolt M	Болт M	Bolzen M	Śruba M	Šroub M
18	___GOST_7798_70	# GOST 7798-70	# ГОСТ 7798-70	# GOST 7798-70	# GOST 7798-70	# GOST 7
19	BOLT_3_M	Bolt 3 M	Болт 3 M	Bolzen 3 M	Śruba 3 M	Šroub 3 M
20	BOLT_4_M	Bolt 4 M	Болт 4 M	Bolzen 4 M	Śruba 4 M	Šroub 4 M
21		Thread diameter	Диаметр резьбы	Gewindedurchmesser	Średnica gwintu	Průměr závi
22		Detail length	Длина детали	Detaillänge	Długość	Délka
23		Thread Length	Длина резьбы	Gewindelänge	Segment gwintowany	Délka závitu

Для того, чтобы использовать в скрипте локальный ресурс, нужно обращаться к его имени (Label) с префиксом «@».

```
     Public(
          L, @ROD_LENGTH,
```

```
        dr, @THREAD_DIAMETER,
        WP1, @WORK_PLANE_1,
        swTr, @SMALL_THREAD_STEP,
        rScrewOk, @BOLT_OK,
        s, @BOLT_HEAD_WIDTH,
        b, @THREAD_LENGTH,
        massa, @MASS
    );
```

При выполнении скрипта под приложением в соответствующей локали вместо ссылок @<Имя ресурса> будут выбраны необходимые значения:

```
    Public(
        L, "Длина стержня",
        dr, "Диаметр резьбы",
        WP1, "Рабочая плоскость 1",
        swTr, "Мелкий шаг резьбы",
        rScrewOk, "Болт Ok",
        s, "Размер под ключ",
        b, "Длина резьбы",
        massa, "Масса"
    );
```

Перечень требований. Оформление графики и написание скрипта объектов базы данных стандартных элементов с помощью MechWizard

Настоящие требования должны применяться для всех вновь создаваемых и редактируемых объектов базы.

Свойства объекта

Название объекта должно содержать название стандарта и его номер (например, «ГОСТ 7798-70», «ISO 657/16»).

Комментарий к названию должен содержать название объекта в соответствие со стандартом (например, «Болты с шестигранной головкой и направляющим подголовком класса точности А»).

Строку спецификации следует выполнять в скрипте путем задания переменной strPartName.

Все поля комментариев табличных параметров, входящих в разделы DBFLD и DBINF функции UniDialog должны быть заполнены русскими описаниями. Рекомендуется заполнять русскими описаниями все поля комментариев таблицы.

Рекомендуется группировать поля таблицы таким образом, чтобы определяющие параметры были сгруппированы возможно левее.

По умолчанию принимается название исполнений госта «Implementation1» (без пробела), если иное не установлено стандартом.

Для российских гостов в свойствах устанавливается стандарт ГОСТ (даже если деталь создана по ОСТу, ТУ или АТК).

Параметры

При задании названий параметров следует руководствоваться:

1. Сходными параметрами присутствующих в базе объектов (например, dr – диаметр резьбы, p – шаг резьбы, L-длина профиля).
2. Названиями параметров в стандарте (например, D1 – наружный диаметр фланца, Dy – диаметр проходной условный, b – длина резьбовой части). В этом случае назначение параметра должно определяться либо комментарием в таблице, либо описанием открытого (public) параметра, либо задаваться комментарием.
3. Правилами названия переменных языка С.

Правила определения переменных:

- Идентификатором называется последовательность символов, представляющая собой имя переменной, функции или оператора.
- Идентификатор может состоять из букв латинского алфавита, цифр и символа подчеркивания.
- Идентификатор должен начинаться с буквы или символа подчеркивания.
- Идентификаторы, состоящие из одинаковых последовательностей букв, но в разном регистре (строчные/прописные буквы), являются разными идентификаторами.
- Максимальная длина идентификатора ограничена 50 символами.

Рекомендации к названию переменных:

- Использовать префиксы типа переменной (r – вещественный, i – целый, str – строковый, b – логический, vec – вектор, pnt - точка)
- Использовать описательные имена переменных (например, rLength вместо L, bSection вместо sh)
- Использовать прописные и сточные буквы в длинных именах переменных (например, rWP2offset, bUseFlatSeal)

Описание параметра в качестве открытого (public) должно быть обусловлено:

1. Необходимостью задания коннекта (connect) объекта

2. Необходимостью задания типоразмера объекта (параметры являются определяющими для объекта)

3. Необходимостью извлечения данного параметра для дальнейшего использования (универсальным маркером, либо включение в поля таблицы и т.п.)

Во всех остальных случаях включение параметра в раздел public не допускается.

Параметрам strTheName, strTheType и strTheSubType допускается присваивать только значения на английском языке. Значения этих

переменных необходимо подбирать таким образом, чтобы обеспечивалась максимальная унификация процедур коннекта.

Все параметры типа Public должны иметь описание на русском языке (включая описания рабочих плоскостей). Описания параметров должны начинаться со значащего существительного (например, «Диаметр наружный», «Ширина паза втулки»).

Не рекомендуется выполнять описания параметров в разделе Public с размерностями. Описания параметров в таблице и описаниях поля VFLD могут иметь размерность. В этом случае она указывается в единицах СИ после запятой (например, «Давление условное, МПа», «Диаметр внешний, мм»).

Все названия параметров разделов VFLD и BFLD должны быть заполнены на русском языке. Не допускается использование описаний переменных, одинаковых с именем переменной (т.е. не допускается использовать L,”L”).

Если для вида установлен предварительный просмотр с обозначением размеров детали, то допускается использовать комментарии произвольных параметров в виде VFLD L, “L, Длина”.

Рекомендуется задавать описание для всех параметров, используемых в скрипте (в виде комментариев).

Если в стандарте указана масса детали, то она должна быть включена в параметры public и раздел UniDialog DBINF.

Масса детали задается параметром massa и указывается в килограммах на одну деталь.

Если в таблице указана масса по исполнениям, то этим параметрам присваиваются имена massa1, massa2 и т.д.

Для обеспечения корректной локализации скриптов все строковые значения должны оформляться в виде ссылок на ресурсы, кроме строк обозначений названия исполнений, классификаторов strTheName, strTheType, strTheSubType, имен форм и т.п.

Названия ресурсов должны быть заданы прописными английскими буквами без пробелов. Например:

```
@DIAMETER_OF_THREAD
@WORKING_PLANE_1
```

Системные имена исполнений должны содержать только английские буквы.

Скрипт

Комментарии рекомендуется создавать на английском языке.

Не рекомендуется располагать несколько операторов в одной строке.

При создании скрипта следует придерживаться следующего порядка расположения функций:

1. ActHeader()
2. OnInitialisation()
3. (Пользовательские функции)
4. OnMakeParameters()
5. SetGripPoint()
6. OnMoveGripPoint()
7. OnDialog()
8. OnDialogChanged()
9. OnChangeParams()
10. OnInitSelect()
11. OnSelectParam()
12. OnMenu()
13. BeforeConnect()
14. OnConnect()

Операторы задания переменных strPartName и strPartDescription должны располагаться в конце процедуры OnMakeParameters.

В первых строчках скрипта должны быть указаны версия скрипта и описание госта. Далее могут следовать комментарии общего плана по объекту. Например:

```
    SVersion = 2;
        ObjectDescription = "ГОСТ 19425-74";
        // Slope of flange should be taken
        //between 4 and 10%.
        //In particular 8%
```

Для оформления отступа составных операторов рекомендуется использовать табуляцию и не рекомендуется использовать пробелы.

Оформление составных операторов должно соответствовать следующим требованиям:

- Открывающая скобка находится в той же строке, что и оператор
- Вложенные операторы располагаются на одном уровне с отступом на один символ табуляции
- Закрывающая скобка располагается на том же уровне, что и составной оператор.

Пример:

```
function OnConnect {
        if (rPart == 0) {
                if (obj.strTheSubType == "vaBottom") {
                        setWorkId  (0, obj.objectID);
                        Handled = OBJ_HANDLED;
                };
                if (obj.strTheSubType == "vaShell") {
                        setWorkId  (1, obj.objectID);
                        Handled = OBJ_HANDLED;
```

```
                    };
          };
         …
};
```

Оформление функции UniDialog должно соответствовать следующим требованиям:

- Функция оформляется как составной оператор
- Каждое ключевое слово начинается со следующей строки
- Каждая строковая опция многовариантных разделов располагается с новой строки с отступом

Пример:

```
UniDialog(
    DBFLD, seria, h, b,
    DBINF, num, mas1,
    VFLD,
        L,"Длина двутавра, мм",
        strRefDataHeader,"Справочные данные:",
        A, "A, см^2",
        Ix, "Ix, см^4",
        ix, "ix, см",
        Wx, "Wx, см^3",
        Sx, "Sx, см^3",
        Iy, "Iy, см^4",
        Wy, "Wy, см^3",
        iy, "iy, см",
    BFLD, bHid, "Отображать невидимые линии",
    TVIDS,lViewType,"AnyWBK",
    VIEW,"Vids"
);
```

Рекомендуется разбивать сложные арифметические выражения на несколько строк, либо использовать несколько промежуточных переменных.

Рекомендуется с помощью описательных комментариев группировать строчки кода, образующие логически завершенную последовательность (например, блок операторов задания рабочих плоскостей, блок операторов расчета параметров объекта и т.п.).

Оформление двухмерной графики

Чертеж должен обеспечивать распознавание параметрического двухмерного вида элемента базы при возможно большей читаемости и содержать все необходимые для элемента виды (включая графику, из которой создается предпросмотр).

В дальнейшем понятия «одиночный параметрический элемент базы», «деталь» и «ГОСТ» (строчными буквами, чтобы не путать с государственными стандартами) принимаются равнозначными.

Чертеж детали или деталей должен быть выполнен в одном файле. Название файла обязательно должно включать все номера стандартов, на основании которых исполняется деталь, и, при необходимости, описательную часть (например, «ГОСТ 7798-70 Болты с шестигранной головкой», «Детали крепления для фланцевых соединений ОСТ 26-2039, 26-2040, 26-2041, 26-2042»).

Общие требования

По умолчанию масштаб оформления принимается 1:1. Масштаб отрисовки графики выбирается произвольный, но обеспечивающий уверенное чтение чертежа. Рекомендуется строить графику на основании реальных размеров таблицы детали.

Все построения выполняются в пространстве модели. Для создания элементов используются настройки по умолчанию.

Слои

По умолчанию принимается толщина линий для всех слоев и объектов - default

Для черчения должны использоваться слои:

Имя слоя	**Параметр - тип линий объектов**	**Цвет**	**Объекты**
ОСНОВНЫЕ	0 или не отмечается	White (белый)	Объекты, которые должны иметь данный тип линий. Точка вставки
ТОНКИЕ	2	Red (красный)	Объекты, которые должны иметь данный тип линий штриховка
ШТРИХОВЫЕ	4	Green (зелёный)	Объекты, которые должны иметь данный тип линий
ОСЕВЫЕ	назначается отдельной командой	Red (красный)	Объекты, которые должны иметь данный тип линий
РАБОЧИЕ	Флажок "рабочий объект"	Yelow (жёлтый)	Объекты, которые должны иметь данный тип линий
РАЗМЕРЫ	Не отмечается	Blue (синий)	Размеры и текстовые формулы Названия ГОСТ-ов, исполнений и видов, и другие пояснения

Параметр «осевая линия» должен устанавливаться для графических объектов после их перемещения на соответствующий слой.

Размещение графики

Отдельные детали, а также отдельные исполнения одного ГОСТа на чертеже должны четко разделяться от других ГОСТов и исполнений (зонально либо с помощью прямоугольных рамок). При наличии более одного ГОСТа, либо более одного исполнения номер ГОСТа либо название исполнения должны быть написаны над группой видов данного исполнения ГОСТа.

Виды на чертеже должны находиться в проекционной связи (рис.1).

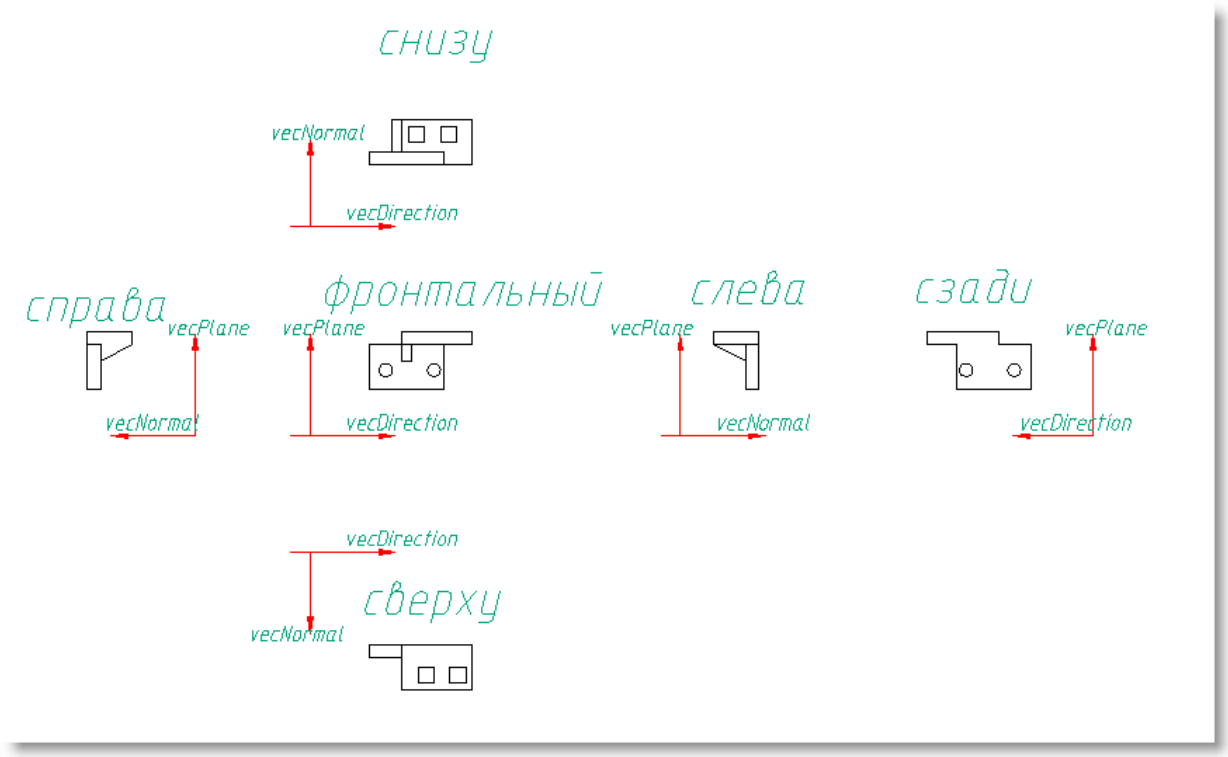


Рисунок 1. Размещение видов на чертеже.

Виды, находящиеся вне проекционной связи должны иметь четкое название для определения отношения.

Рекомендуется группировать виды с разрезами, упрощенные виды, и виды с размерами рядом с соответствующими проекционными видами.

Если один или несколько видов относятся более чем к одному ГОСТу \ исполнению, то он(и) должны иметь четкое название для определения отношения (например: «Вид слева (все исполнения)», «Фронтальный (ГОСТ 7798,7805, 7806)»). Графика вида детали с размерами должна иметь название.

Деталь на видах должна быть ориентирована таким образом, чтобы направление детали, соответствующее vecDirection скрипта, совпадало с осью oX (строго вправо для фронтального вида).

Нанесение размеров

При нанесении размеров следует руководствоваться:

1. Правильным распознаванием;
2. Читаемостью чертежей;
3. Требованиями оформления чертежей по ЕСКД.

Не допускается размещать размеры внутри заштрихованной области (не допускается реализовывать обтекание штриховкой размерных чисел).

Нанесение штриховки

Штриховка должна выполняться типом «User defined» («Из линий»), Угол наклона 45 градусов. Шаг штриховки должен задаваться параметрически в зависимости от ширины самой узкой штрихуемой части (s):

Например:

1. Для тонкостенных объектов s/2;

2. Для массивных объектов s/8;

3. Промежуточный шаг штриховки s/4.

Допускается указывать шаг штриховки в абсолютных единицах кратный 10 (0.1;1;10;100)

Допускаются отступления от этих требований при необходимости штриховки областей в разные стороны.

Параметры

На размеры с помощью инструмента «установить параметр» должны быть установлены соответствующие параметры. Снятие с размера галки «рабочий объект» допускается только для графики образмеренного вида.

Следует избегать использования текстовых формул в поле чертежа (предпочтительно б*о*льшую часть вычислений выносить в скрипт).

Текстовые формулы допускается наносить на чертеж в том случае, когда необходимо вычислить параметры, относящиеся к данному виду данного исполнения, если определено, что их использование в скрипте не будет необходимым.

Использование кириллических букв в текстовых полях, кроме названий не допускается.

При названии параметров следует руководствоваться (в порядке очередности):

1. Аналогичными названиями, уже существующими в базе
2. Названиями параметров в ГОСТе
3. Осмысленными названиями параметров, соответствующими синтаксису языка программирования С.

По возможности следует избегать сложных арифметических выражений, устанавливаемых в параметры размера (предпочтительнее использование переменных типа Protected).

Прочие требования

Название видов должно наноситься однострочным текстом высотой 10мм. Текстовые формулы должны иметь высоту текста 5мм. Названия видов всегда размещаются над видом, а текстовые формулы под видом.

Блокам, входящим в массив, необходимо давать осмысленные названия на английском языке, либо транслите (например, otv_d, hole, chamfer и т.д.). Точка вставки блоков, из которых создается осевой массив должна находиться в центре осевого массива.

Допускается включать в поле чертежа растровое изображение детали, скопированное из ГОСТа, если оно не затрудняет чтения чертежа и не пересекается с другими объектами чертежа.

Синтаксис языка скриптов

Скрипт - это программный код на языке, схожем с СИ, который определяет природу объекта БД программного продукта- параметры, плоскости, реакцию объекта на воздействия. Автоматическую установку зависимостей и т.д. Скрипт состоит из ряда блоков, выполнение которых определяется событиями извне объекта.

Идентификаторы

Идентификатором называется последовательность символов, представляющая собой имя переменной, функции или оператора. Правила описания идентификаторов в MechWizard аналогичны правилам в СИ:

- Идентификатор может состоять из букв латинского алфавита, цифр и символа подчеркивания.
- Идентификатор должен начинаться с буквы или символа подчеркивания.
- Идентификаторы, состоящие из одинаковых последовательностей букв, но в разном регистре (строчные/прописные буквы), являются разными идентификаторами.
- Максимальная длина идентификатора ограничена 50 символами.

Рекомендации к названию имен переменных приведены в соответствующем документе.

Ключевые слова

Ключевые слова - это зарезервированные идентификаторы, которые наделены определенным смыслом. Их можно использовать только в соответствии со значением известным интерпретатору макроязыка. Ключевые слова не могут выступать в качестве идентификатора.

Список ключевых слов перечислен в приложении.

Типы данных

В макроязыке существуют пять основных типов данных:

1. Действительное число с плавающей запятой (аналог double в СИ). Диапозон -1.7e308 .. 1.7e308.
2. Строка.
3. Трехмерная точка (Point)
4. Трехмерный вектор (Vector)
5. Плоскость, определяющаяся базовой точкой и вектором нормали (Plane).

В скрипте принято соглашение к названиям переменных действительного типа добавлять префикс r, к строковым переменным добавлять префикс str или s, к переменным точечного типа добавлять префикс pnt или p, к переменным векторного типа добавлять префикс vec или v, а плоскости обозначать WP (WorkPlane).

Инициализация переменных осуществляется без предварительного описания, как в Бейсике. Чтобы объявить переменную, нужно просто присвоить ей некоторое значение. Если переменная не описана в заголовочной функции ActHeader в секциях Public или Protected, то она будет считаться локальной и будет сохранять свое значение от момента первого определения до конца действия, в котором она описана.

```
        StrName = "ObjectA";
         rPI =  3.1415926;
         rA = 2.0;
```

Для инициализации точки и вектора используются специальные функции-конструкторы Point(), Point(x,y,z) и Vector(), Vector(x,y,z) соответственно.

```
        pnt2=Point();//ручка длины
         pnt1= Point(1.0, 10.0, 10.0); //Это точка с конкретными
координатами
         vec1 = Vector(); // Вектор единичной длины, совпадающий
с осью OX (1,0,0)
         vec1 = Vector(1.0, 1.0, 0); //Вектор с конкретными
координатами
```

К отдельной координате точки или вектора можно обратиться с помощью квалификатора:

```
        pnt1:x = 12.5;//Извлечение абсциссы из точки
         vec1:y = 25.6;//Извлечение ординаты из точки
```

Для инициализации плоскости используется функция-конструктор Plane(pnt, vecNormal).

```
         WP1 = Plane( pntBase, vecNormal );
         //pnt- Это базовая точка для плоскости WP1
         //vecNormal- Это вектор нормали для плоскостиWP1
```

Например,

```
         WP1 = Plane( pntOrigin, vecDirection);
```

Индексирование переменных

Если объявить ряд переменных любого типа с именами, отличающимися только цифрой в конце, то к этим переменным можно обращаться по индексу, как к элементам массива.

```
        rParam1 = 1.0;
         rParam2 = 2.0;
         rParam3 = 4.0;
         ...
         rParam[2] = 45.7;
         rParam[3] = 9.1223;
```

Комментарии

Комментарии в скрипте начинаются с двойной косой линии и заканчиваются концом строки.

```
        //Расставляем плоскости в порядке против часовой стрелки
         vecNormal=getLocalNormal(vecDirection,vecPlane);
         WP1 = Plane( pntOrigin, vecDirection );
         WP2 = Plane( pntOrigin-vecPlane*(0), -vecPlane );
         //Правые боковые
         WP3 = Plane( pntOrigin+vecNormal*(b/2), vecNormal );
         WP4 = Plane( pntOrigin+vecNormal*(b/2)+vecPlane*h,
vecNormal );
         WP5 = Plane( pntOrigin+vecPlane*(h), vecPlane );
```

Операторы

Все операторы макроязыка могут быть разделены на следующие категории:

- арифметические операторы
- логические операторы
- условный оператор
- оператор цикла
- другие операторы (пустой оператор, оператор присваивания, оператор доступа к внутренним членам)

Все операторы заканчиваются пустым оператором - точкой с запятой ";".

Пустой оператор

Пустой оператор состоит только из точки с запятой. При выполнении этого оператора ничего не происходит. Этот оператор является разделителем для других операторов, функций и строк в скрипте.

Составной оператор

Составной оператор представляет собой несколько операторов и объявлений, заключенных в фигурные скобки {}. Выполнение составного оператора заключается в последовательном выполнении составляющих его операторов. В конце составного оператора точка с запятой не ставится.

Оператор присваивания

Оператор присваивания обозначается с помощью знака равенства**.**

Выражение *A = B;* означает, что переменной **A** присваивается значение выражения **B**, причем тип переменной *A* установится типом выражения *B*.

Оператор доступа к внутренним членам сложных типов данных

Для доступа к внутренним членам сложных типов данных используется специальный оператор-квалификатор «:». Так, например, его можно использовать для доступа к отдельным координатам переменных типа Точка и Вектор.

Арифметические и логические операторы

Синтаксис арифметических и логических операторов в макроязыке скрипта аналогичен синтаксису в СИ.

Арифметические		**Логические**	
A + B	Сложение	A == B	A равно B
A - B	Вычитание	A != B	A неравно B
A * B	Умножение	A > B	A больше B
A / D	Деление	A < B	A меньше B
		A && B	A и B
		A || B	A или B

Для аргументов типа Точка результатом операции вычитания будет вектор. А умножение вектора на число даст новый вектор, равный скалярному произведению вектора на число.

```
vecDirection = pntEnd - pntStart;
 vecScaled = vecDirection * rScale;
```

Условный оператор

Формат оператора:

```
if ( <выражение> )
        оператор_IF1;
 else
        оператор_ELSE1;
        //<выражение>- это логическое условие на выполнение
оператора
```

В качестве оператора_IF1 и/или _ELSE1 может выступать составной оператор, т.е. группа операторов, заключенных в фигурные скобки.

Выполнение оператора if начинается с вычисления выражения.

Далее выполнение осуществляется по следующей схеме:

- если выражение истинно (т.е. отлично от 0), то выполняется оператор _IF1.
- если выражение ложно (т.е. равно 0), то выполняется оператор _ELSE1
- если выражение ложно и отсутствует оператор _ELSE1, то выполняется следующий за if оператор

Блок операторов после else может отсутствовать вместе с самим словом else. В этом случае условный оператор выглядит следующим образом:

```
if ( <выражение> )
        оператор_IF1;
```

В качестве выражения может выступать любой логический или арифметический оператор или последовательность таких операторов разделенных круглыми скобками.

```
if (A>1)
        оператор_IF1;
 if ((A>1)&&(B<5)&&(C==25))
        оператор_IF1;
```

Допускается использование вложенных условных операторов, т.е. условный оператор может быть включен в конструкцию *if* или конструкцию *else* другого условного оператора.

```
if (A > 1){
        if (B < 5){
                if (C == 25)
                        оператор_IF1;
        }
 };
```

Оператор цикла

Формат оператора:

```
while (выражение)
        оператор_BODY;
```

Требования к выражению такие же, как в условном операторе. Оператор _BODY называется телом цикла.

Выполнение оператора цикла начинается с вычисления выражения. Далее выполнение осуществляется по следующей схеме:

1. Если выражение истинно (отлично от 0), то выполняется тело цикла.
2. Если выражение ложно, то выполняется следующий за while оператор.
3. Цикл повторяется с пункта 1.

Функции

Функция - это совокупность операторов и вызовов других функций. Каждая функция имеет имя. Объявление функции начинается с ключевого слова function. Тело функции заключено в фигурные скобки. В данной версии скриптов функции не имеют параметров.

Формат функции:

```
function Function_Name{
    оператор_1;
    ...
    оператор_N;
 }
```

Допускается вызывать из одной функции другие, но вызываемые из funcA функции должны быть объявлены выше в теле скрипта.

Пример:

```
function user_Function_1{
    MessageBox("Это моя первая функция");
 }
 function OnInitialization{
    user_Function_1(); // Правильный вызов. Вызываемая функция
объявлена выше!
    user_Function_2(); // Неправильный вызов. Вызываемая функция
объявлена ниже!
 }
 function user_Function_2{
    MessageBox("Это моя вторая функция");
 }
```

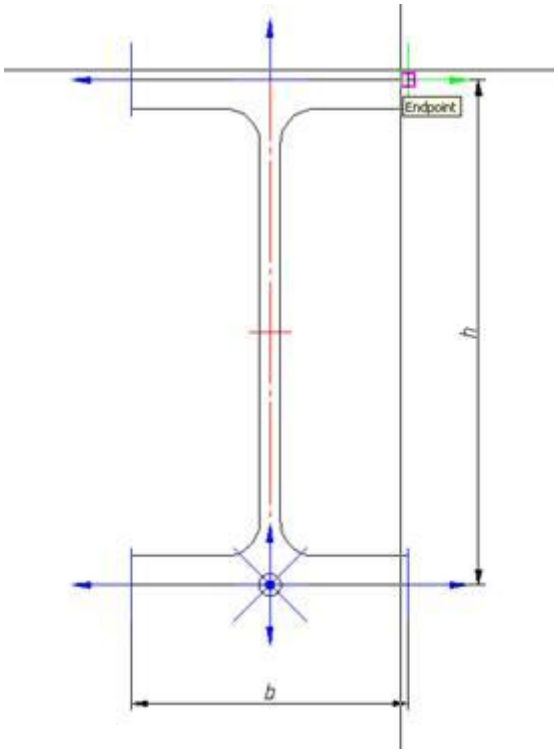
Интерпретатор имеет ряд встроенных функций, перечень которых приведен в приложении 2.

Задание плоскостей объектов

Задание абстрактной геометрии объекта (точек, векторов, плоскостей) может быть в координатной форме (практически не используется) или относительно локальной системы координат. Положение точек определяется путем задания операторов сложения точки и плоскости. В примере положение базовой точки плоскости задается как

```
pntWP4=pntOrigin+vecDirection*(b/2)+vecPlane*h
```

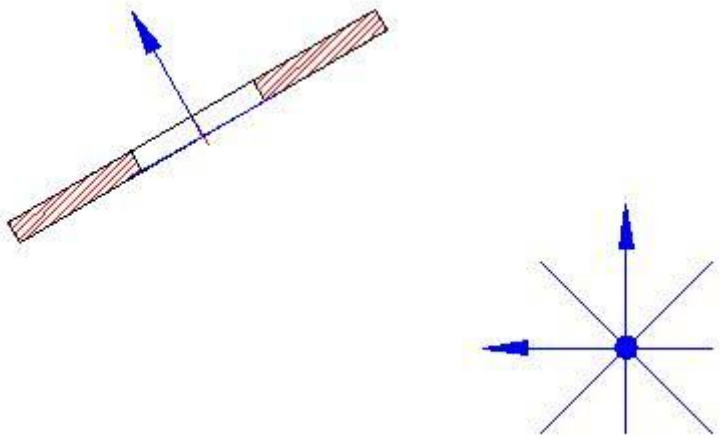
Вектор нормали этой плоскости коллинеарен вектору вставки vecDirection



Следовательно, сама подсвечиваемая плоскость WP4 будет задаваться как

```
WP4=Plane(pntWP4,vecDirection);
```

Если вектор нормали плоскости не ортогонален системе координат, то он зачастую определяется с помощью оператора RotateBy или GetLocalNormal.



Например, для накладки конического днища, вектор нормали плоскости WP2 определяется как

```
        vecNormal=getLocalNormal(vecDirection,vecPlane);
         vecFitting = rotateBy(vecDirection,90-ang/2,vecNormal);
         //ang - это угол днища в плане
```

Кроме выше обозначенных, используются операторы нахождения вектора нормали и базовой точки плоскости:

```
        pntWP1 = Point(WP1);
         vecWP1 = Vector(WP1);
```

Для нахождения расстояния между точкой и плоскостью используется разность точки и плоскости

```
        rDistance = WP1-pntOrigin;
```

Длину вектора (расстояние между точками) определяют оператором vecLen:

```
 rLength = vecLen(pntWP1 - pntOrigin);
```

Ортогональный вектор находится оператором getPerp:

```
vecPlane = getPerp(vecDirection);
```

Работа с таблицей

Для работы с таблицей объекта используются два оператора LoadInCache и SelectInCache.

Оператор LoadInCache загружает указываемые параметры в кэш работы с таблицей. Данный оператор в новых версиях необходимо вызывать принудительно только при выборке с вычисляемым выражением от параметров (..., "+", ...).

Оператор SelectInCache производит выборку из таблицы в соответствии с указанными критериями (см синтаксис в приложении)

Выборка по «kFirst» выбирает одну первую строку. Выборка по «kFilter» выбирает диапазон записей из таблицы, соответствующих указанному фильтру.

Варианты выборки по конкретным параметрам «~» - приближенное равенство параметра в записи выбираемому значению. «=» - точное равенство, «+» - выборка по фильтру. Выборка по фильтру означет, что если логическое условие фильтра выполняется, то запись выбирается из таблицы (подвергается дальнейшим проверкам по условиям).

Например, функция

```
SelectInCache("kFirst", "dr", "~", rdr, "L", "+", "L>=rMinLen&&
L<=rMaxLen&& (L-b<=rTrLen)");
```

выбирает из таблицы первую запись, где сочетание параметров dr,L,b таково, что dr≈rdr, rMinLen<=L<=rMaxLen, (L-b)<=rTrLen.

Для функции SelectInCache важен порядок перечисления условий выборки. Это означает, что в первую очередь из таблицы будут выбираться записи, соответствующие первому условию, далее - из них выбираются записи соответствующие второму условию и т.д.

Если в детали присутствует несколько таблиц, то, при совпадении наименований параметров необходимо включить галку «Использовать имя таблицы в качестве префикса параметров» в контекстном меню соответствующей таблицы.

В этом случае обращение к параметру в скрипте будет производиться следующим образом:

```
 <Имя таблицы>.<Имя параметра>
```

Например:

```
If (Table0.dr==Table1.Thread1) {
    ...
 };
```

Для обеспечения выборки из не первой таблицы необходимо модифицировать оператор SelectInCache.

Например, если необходимо выбрать из таблицы Table1 параметр dr, равный 10, то первым условием в операторе выбора из таблицы должен следовать индекс таблицы (начинается с 0):

```
 SelectInCache(1, «kFirst», «dr», «~», 10);
```

Для таблицы с имененм Table0 имя таблицы можно опускать. Т.е. справедливо и то, и другое выражение:

```
SelectInCache ("kFirst", "dr", "~", 10, "L", "~", 50);
 SelectInCache ("0","kFirst", "dr", "~", 10, "L", "~", 50);
```

Диалог объекта. Оформление функции UniDialog

Вид диалога вставки объекта определяется заданием ключей функции UniDialog.

См. соответствующий раздел приложения.

Здесь следует просто сопоставить, что

DBFLD - это определяющие табличные параметры.

DBINF - справочные табличные параметры.

VFLD - это произвольные параметры и справочные данные.

VIDS - перечень исполнений объекта (возможно опускать, если имеется только одно исполнение.

TVIDS - различные виды одного исполнения (возможные ключи - по видам, All - те, которые присутствуют, AnyWBK - противоположные виды генерируются зеркальным отображением (для симметричных объектов).

VIEW - превью видов.

Ключ OnDglBeforeSelParam в ActHeader определяет, будет ли вызываться диалог перед динамическим выбором параметров.

Подключение пользовательской формы

Если для объекта задана форма, то для того, чтобы она вызывалась, необходимо в функции OnDialog Вместо функции UniDialog вызывать конкретную форму.

Например, если в объекте задана форма с именем frmMain, то следует написать следующее:

```
functionOnDialog{
     ShowForm("frmMain");
 }
```

Подобным образом по условиям можно вызывать несколько различных форм объекта, обращаясь к ним по имени с помощью оператора ShowForm.

Установка зависимостей

Операторы установки зависимостей реализуют геометрические и параметрические зависимости.

Параметрические зависимости

```
SetParamConstraint(< param >,obj,EXPR,< expression to
constrain>,bBidirect);
```

<param> - название параметра объекта, к которому устанавливается зависимость.

Obj - к какому объекту устанавливается.

EXPR - пока только зависимость на выражение

<expression to constrain> - арифметическая комбинация параметров, на которую ставится зависимость (должна быть в кавычках).

bBidirect - двунаправленная или однонаправленная зависимость.

Геометрические зависимости

```
SetGeomConstraint(rTYPE, rDIRECTION, obj, < WPname >, <
WPtoConnect >,offset,bBidirect);
 SetGeomConstraint(ANGE,CODIRECT,< obj >,< wpSourcePlane >,<
wpObjPlane >,< rAngle >,< bBiderect >);
```

Типы зависимостей rTYPE = INSERT (вставка) MATE (совмещение по плоскости) AXIS (совмещение по оси) DIRECTION (он же ANGLE) (угловая ориентация).

Направление rDIRECTION = CODIRECT (сонаправлено) CONTRDIRECT (противонаправлено).

<WPname> - имя плоскости, которой присоединяются.

<WPtoConnect> - имя плоскости, к которой присоединяются.

Offset - численное или параметрическое выражение, определяющее:

Для INSERT - осевое расстояние между плоскостями.

MATE - нормальное расстояние между плоскостями.

AXIS - относительное смещение осей в плане.

DIRECTION - угол относительной ориентации плоскостей.

Для зависимостей DIRECTION (ANGLE) есть определенные условия:

Синтаксис:

```
SetGeomConstraint(ANGE,CODIRECT,< obj >,< wpSourcePlane >,< wpObjPlane >,< rAngle >,< bBiderect >);
```

ANGLE - ключевое слово угловой зависимости.

CODIRECT - всегда для угловой зависимости. Полагается из-за минимальности угла поворота.

<obj> - к какому объекту ставить.

<wpSourcePlane> - плоскость исходного объекта, к которой ставится зависимость.

<wpObjPlane> - плоскость того объекта, к которому присоединяемся.

<rAngle> - числовое значение угла зависимости в градусах.

<bBidirect>=(TRUE,FALSE) - однонаправленная или двунаправленная зависимость. По умолчанию FALSE.

Зависимости типа ANGLE следует вводить после зависимостей типа INSERT, MATE, AXIS.

При наличии зависимости типа INSERT, MATE или AXIS поворот осуществляется относительно осей плоскостей данной зависимости.

При отсутствии зависимостей типа INSERT, MATE или AXIS поворот осуществляется вокруг нормали к плоскости, в которой лежат оба вектора в направлении минимального угла поворота.

Если рассматриваемые векторы коллинеарны, поворот осуществляется относительно оси OX.

Пример:

```
SetGeomConstraint(ANGLE,CODIRECT, obj,WPnormal,obj.WPnormal,45);
 SetGeomConstraint(ANGLE,CODIRECT, obj,OYPlane,obj.WP2,15,TRUE);
```

Примеры различных типов зависимостей можно найти в приложении.

Функция ShowValue

Использование окна отладчика позволяет выявить ошибки в скрипте и определить значения параметров во время выполнения скрипта. Синтаксис см. в приложении.

Приложение 1. Список ключевых слов и зарезервированных переменных

SVersion=2;	Первая строка скрипта, определяющая версию интерпретатора
strTheType	Тип объекта
strTheName	Имя объекта
strTheSubType	Подтип объекта
pntOrigin	Точка вставки
vecDirection	Вектор направления вставки
VecPlane	Ортогональный вектор к vecDirection
Scl	Масштаб объекта
rZOrder	Порядок следования (используется для перекрытия)
strDesignName	Системное имя исполнения
rXcoord	Относительная абсцисса курсора при динамическом выборе параметров
rYcoord	Относительная ордината курсора при динамическом выборе параметров
VecCoord	Переменная доступная в OnSelectParam VecCoord:X - относительная координата по X VecCoord:Y - относительная координата по Y
pntGrip#	(#-число) Точка ручки с номером #
rPart	Порядковый номер запроса.

rKbd	Флаг ввода с клавиатуры. Равен 1 если значение введено с клавиатуры.
strPartName	Название детали в спецификации (раздел наименование)
strPartDescription	Описание детали в спецификации (раздел обозначение)
strPromt	Строка подсказки (при динамическом выборе или при выборе детали для коннекта)
NSelect	Количество запросов объектов при вставке.
NPart	Количество выбираемых параметров, если 0 или не определено тогда просто вызывается диалог.
BreakCur	Завершить текущий выбор динамического параметра
BreakAll	Завершить весь цикл выбора динамических параметров
NoVectorSelect	Не указывает вектор вставки (используется при коннекте)
Handled	Результат обработки объекта
NGrip	Количество ручек
NMovingGrip	Номер редактируемой ручки
lSimpleView	Переменная определяет тип вида, в UniDialog создается и устанавливается автоматически. Если == 1 - упрощенный вид. Если == 2 - вид с разрезом
scaleDisable	Если этот параметр установлен в 1 в UniDialog запрещено изменение Scale
ShowWhenSelPnt	Если установлен этот флажок, объект, будет отрисовываться во время вставки, если не указано, то по умолчанию включен
ContourOnLine	Генерация контура не из скрипта

	отрисовки, а на лету после отрисовки
IsAHole	Объект является отверстием (врезаемый объект)
OnDlgBeforeSelectParam	Вызов диалога перед вызовом функции OnSelectParam.
IsACalculator	Задается для расчетов равным 1
No3dViews	Отключает автоматическое управление видами
AutoShift	Автоматическое псевдоудержание кнопки Shift.
UseOnlyZOrder	Использовать для пересечения только rZOrder, не использовать координату по Z
fUseCommonScale	Поддерживать внешнее управление масштабом
MINPOSSIBLEVALUE	Минимально допустимое значение для приложения
NotStdBody	Если ==1, то деталь не помещается в каталог компонентов
SpecPartition	Раздел спецификации, в который попадает деталь
UnknownVal=nknownValue=-0,12345	Значение, присваиваемое переменной по умолчанию. Если переменная не определена, то ей присваивается это значение.

Список фильтров функции SelectInCache:

- "kAsk" - Спросить пользователя.
- "kFirst" - Найти первое значение в таблице, удовлетворяющее заданному условию.
- "kLast" - Найти последнее значение в таблице, удовлетворяющее заданному условию.
- "kFilter" - Выбрать все значения из таблицы, удовлетворяющие заданному условию.
- "kAskIfNeed" - Спросить пользователя, если выбор не удался.

Значения, которые может принимать переменная Handled:

- OBJ_UNHANDLED - Зависимости не обработаны.
- OBJ_HANDLED - Зависимости обработаны
- OBJ_WAITMORE - Ожидание ввода

- OBJ_WAITMORE_M
- OBJ_WARNING- Ошибка установки зависимостей. При возвращении этого кода, родительский объект будет стремиться установить те же параметры, что и у дочернего, после чего обновление дочернего объекта повторяется.
- OBJ_ERROR

Константы вида для переменной lViewType

- VFRONT - Фронтальный вид
- VLEFT - Вид сверху
- VRIGHT - Вид снизу
- VTOP - Вид слева
- VBOTTOM - Вид справа
- VBACK - Вид сзади

Типы зависимости

- INSERT - Вставка
- MATE - Совмещение по плоскости
- AXIS - Совмещение по оси
- DIRECTION - Совмещение по направлению
- ANGLE - Угловая зависимость

Подтипы зависимости

- CODIRECT - Сонаправленная зависимость
- CONTRDIRECT - противонаправленная зависимость

Коды возвращаемы диалогом или функцией MessageBox:

- IDOK
- IDCANCEL
- IDYES
- IDNO

Коды для функции MessageBox :

- MB_YESNO
- MB_OKCANCEL
- MB_OK
- MB_ICONINFORMATION
- MB_ICONQUESTION
- MB_ICONERROR
- MB_ICONWARNING

Пример:

```
 MessageBox("Test");
 if(IDYES == MessageBox("Да или нет?", MB_YESNO,
MB_ICONQUESTION)){
         ...
 }
```

Математические функции и функции работы с плоскостями

Функция Комментарий

`sin(alfa)`
`cos(alfa)`
`tg(alfa)`
	Тригонометрические функции.
	alfa - угол в градусах.
	Возвращают вещественное число

`asin(c)`
`acos(c)`
`atg(c)`
	Обратные тригонометрические функции.
	c - число
	Возвращают угол в градусах.

`DegToRad(deg)`	Функция конвертации градусов в радианы **deg** - угол в градусах
`RadToDeg(rad)`	Функция конвертации радиан в градусы **rad** - угол в радианах
`int(value)`	Функция округления вещественного числа до целого *value* - вещественное число
`abs(value)`	Функция получения модуля числа
`sqrt(value)`	Функция квадратного корня из числа
`min(a,b)` `max(a,b)`	Функции получения минимального и максимального числа из пары чисел *a* и *b*
`iff(expr, a, b)`	Функция - аналог арифметического оператора **if**. Возвращает *a*, если результат выражения *expr* >= 1, иначе *b*.
`VecLen(vec)`	Функция получения длины вектора *vec*.
`VecUnit(vec)`	Функция получения

	нормированного вектора (единичной длины) из *vec*.
`VecCodirect(vec1,vec2)`	Функция проверки сонаправленности векторов *vec1* и *vec2*. Возвращает 1 если vec1 сонаправлен vec2, иначе 0.
`anglePi(vec1, vec2)`	Функция возвращает угол между векторами *vec1* и *vec2* в интервале от 0 до PI.
`angleTwoPi(vec1,vec2)`	Функция возвращает угол между векторами *vec1* и *vec2* в интервале от 0 до 2*PI.

`Point()`
`Point(x, y, z)`
`Point(plane)`
	Функция возвращает точку. Может использовать в качестве конструктора точки. Без аргументов возвращает точку P(0,0,0). Если аргумент - плоскость, то извлекает из нее базовую точку.

`Vector()`
`Vector(x, y, z)`
`Vector(plane)`
	Функция возвращает вектор. Может использовать в качестве конструктора вектора. Без аргументов возвращает вектор V(1,0,0). Если аргумент - плоскость, то извлекает из нее нормаль.

`Plane(pntBase,vecNormal)`	Функция - конструктор плоскости. Возвращает плоскость, проходящую через точку *pntBase* и имеющую нормаль *vecNormal*.

`getPerp(vec)`	Функция возвращает вектор ортогональный вектору *vec*.
`rotateBy(vec1,rAngle,vecNormal)`	Функция выполняет поворот вектора *vec1* на угол *rAngle* (в градусах) в плоскости, имеющей нормаль *vecNormal*. Возвращает результат операции.
`getPerpendicular(pnt1,pnt2,pntFrom,pntBase)`	Функция возвращает длину перпендикуляра, построенного из точки *pntFrom* на прямую, заданную точками *pnt1* и *pnt2*. В переменной *pntBase* возвращается точка основания полученного перпендикуляра на указанной прямой.
`getMiddle(pnt1, pnt2)`	Функция возвращает точку середины отрезка, заданного точками *pnt1* и *pnt2*
`getLinesIntersect(pnt1,vec1,pnt2,vec2)` `getLinesIntersect(pnt11,pnt12,pnt21,pnt22)`	Функция возвращает точку пересечения прямых задаваемых либо парами точек [*pnt11,pnt12*] и [*pnt21,pnt22*] либо точкой и вектором: [*pnt1,vec1*] и [*pnt2,vec2*]. Если прямые не пересекаются, то в возвращаемой точке координата Z равна -1.
`getLocalNormal(vec1,vec2)`	Функция возвращает векторное произведение вектора *vec1* на вектор *vec2*.
`projectOnPlane(pnt,plane2,rDistance)`	Функция возвращает проекцию точки *pnt* на

`projectOnPlane(plane1,plane2,rDistance)`	плоскости, находящейся на расстоянии *rDistance* от плоскости *plane2*. Если в качестве первого аргумента передается плоскость, то функция вернет плоскость проходящую через проекцию точки Point(*plane1*) на плоскости, находящейся на расстоянии *rDistance* от плоскости *plane2*, и имеющую вектор нормали совпадающий с вектором нормали плоскости *plane2*.
Вычитание точки из плоскости	Пример: `d = WP - pnt;` Операция возвращает знаковое расстояние точки *pnt* от плоскости *WP*. Если точка *pnt* находится со стороны, куда указывает нормаль плоскости *WP*, то расстояние положительное, иначе - отрицательное.

Форматирование строки числами	К строковой переменной можно «добавить» число - результатом будет строка.
	
	Пример:
	
	`A = 1;`
	`B = 5;`
	`Str = "Type " + A + ". Modification " + B + ".";`

Результатом будет строка

"Type 1. Modification 5."

```
FmtText(text, number, formated_text,...,)
```

Возвращает rtf текст, если среди параметров есть хотя бы один форматированный, обычный - в противном случае.

На вход функции может подаваться переменное количество параметров, в качестве параметров могут использоваться выражения.

```
FmtSuper(Text);
 FmtSub(Text);
```

FmtSuper() - возвращает rtf текст форматированный как Superscript(верхний индекс)**FmtSub()** - соответственно, для Subscript (нижний индекс).

Text может быть строкой или числом

```
FmtDiv(Text1, Tex2, Splash)
```

FmtDiv() - возвращает дробь,

если **Splash != 0** то у дроби будет разделитель.

Пример использования:

```
a = "super";
 b= 1;
 c = 3;
 strPosition =
FmtText(
         "Начало
текста ",
         "Верхний
индекс-",
FmtSuper(a),
         "Нижний
```

```
индекс-",
FmtSub("sub"),
         " Дальше
просто текст ",
         "Дробь-
",FmtDiv(b, c, TRUE),
         "Дробь без
разделителя-
",FmtDiv(b, c, FALSE)
 );
```

`FmtDigit(rNumber, rDelimeter)`	Возвращает строку форматированным десятичным разделителем. **rNumber** - число или строка, содержащая число, если **rDelimeter == 0** то разделитель - точка, если **rDelimeter == 1** то разделитель запятая, если **rDelimeter == 2** то разделитель берется из системной локали. **Пример:**

```
a = 1.5;
 Formatted =
FmtDigit(a, 1)
```

На выходе **Formatted = "1,5"**

Функции обработки плоскостей

Функция	Комментарий
`GetNearestPlane(pnt)`	Функция возвращает имя ближайшей плоскостик точке**pnt** выбранного объекта. *pnt* - переменная типа Point. **Пример:**

```
function OnConnect{
        if(obj.strTheType ==
```

```
"StdJointParts"){
            // ...
         } else {
            strName =
GetNearestPlane(pntOrigin);
            if(strName !=
UnknownValue){

SetGeomConstraint(INSERT,
CODIRECT, obj, WP1, strName,
0);
                NoVectorSelect
= 1;
                Handled =
OBJ_HANDLED;
            }
         }
 }
```

```
//Вариант 1
findNearest(
        pnt,
        Name1,
        Name2,
         ... ,
        NameN
)
//Вариант 2
findNearest(
        pnt,
        arrName,
        arrIndexStart,
        arrSize
)
```

Возвращает индекс ближайшей точки или плоскости в зависимости от типов параметров, заданных именами *Name1*, ... *NameN* или *arrName*.

Функция поддерживает два варианта вызова:

1) в качестве аргументов указываются конкретные имена public-параметров объекта

2) в качестве аргументов указываются массив public-параметра объекта с указанием стартового индекса для поиска и количества элементов в массиве.

pnt - переменная типа Point.

Name1,NameN,arrName - имена public-параметров выбранного объекта

arrIndexStart - стартовый индекс для поиска в массиве

arrSize - размер массива

Пример:

```
 //1)
         rNearest =
findNearest(pntOrigin, obj.WP1,
```

```
obj.WP11, obj.WP21);
        // rNearest принимает
значения 0, 1 или 2
 //2)
        rNearest =
findNearest(pntOrigin, obj.WP,
1, 4);
        // rNearest принимает
значения 1, 2, 3 или 4
```

```
//Вариант1
nearestPlaneName(
        pnt,
        Name1,
        Name2,
         ... ,
        NameN
)
//Вариант2
nearestPlaneName(
        pnt,
        arrName,
        arrIndexStart,
        arrSize
)
```

Возвращает имя ближайшей точки или плоскости в зависимости от типов параметров, заданных именами *Name1*, ... *NameN* или *arrName*.

Функция поддерживает два варианта вызова:

1) в качестве аргументов указываются конкретные имена public-параметров объекта

2) в качестве аргументов указываются массив public-параметра объекта с указанием стартового индекса для поиска и количества элементов в массиве.

pnt - переменная типа Point.

Name1,NameN,arrName - имена public-параметров выбранного объекта

arrIndexStart - стартовый индекс для поиска в массиве

arrSize - размер массива

Пример:

```
 //Вариант1
        strNearest =
nearestPlaneName(pntOrigin,
obj.WP1, obj.WP11, obj.WP21);
        // strNearest
принимает значения "obj.WP1",
"obj.WP11" или "obj.WP21"
 //Вариант2
        strNearest =
nearestPlaneName(pntOrigin,
obj.WP, 1, 4);
        // strNearest
```

```
принимает значения "obj.WP1", "obj.WP2", "obj.WP3" или "obj.WP4"
```

```
 restoreBasis(vecOld, vecNormal, vecNew);
```

Функция конвертирует старый базис в соответствии с новым положением одного из векторов.

Приниматься могут следующие комбинации в СЛЕДУЮЩЕМ порядке:

(x, y, z, newX)

(y, z, x, newY)

(z, x, y, newZ) то есть vRrev - предыдущий vNext в тройке XYZ

```
 MakeVectorsFromView(lViewType, vecDirection, vecPlane);
```

Выставляет положение веторов из типа вида

Пример:

```
 MakeVectorsFromView(VTOP, vecDirection, vecPlane);
```

```
setBasePlaneForSelect(planeName, vecPlane = vecPlane);
```

Устанавливает плоскость(систему координат в которых будет работать SelectParam) т.е. от этой системы будут зависеть rXcoord, rYcoord.

Применение. При выборе несимметричных деталей у которых направление выбора зависит от зафиксированной плоскости, и поэтому расстояние может считаться не от pntOrigin а от какой то плоскости - например стандартные концы валов.

Пример:

```
setBasePlaneForSelect(WP1);
```

Используется для присоединения концов валов.

Функции для работы с таблицами

Загрузка таблицы из базы данных в Кэш

```
LoadInCache(name1, name2, ... , nameN)
```

Загружает выбранные поля из таблицы в кэш работы с таблицей.

Функция имеет переменное число аргументов.

name1, ... ,nameN - имена табличных параметров.

Пример:

```
LoadInCache(D);
LoadInCache(dr, dH);
```

Выборка из таблицы

```
SelectInCache(strFilterKey, strParamName, strOperation, Value
 ... ,
 strFilterKeyN, strParamNameN, strOperationN, ValueN)
```

Производит выборку из кэша таблицы объекта. Переменное число аргументов.

strFilterKey - Ключ фильтрации - строка, принимающая следующие значения:

"kFirst" - выбрать первое из значений, удовлетворяющих условию фильтрации

"kLast" - выбрать последнее из значений, удовлетворяющих условию фильтрации

"kFilter" - выбрать диапазон подходящих значений

"kAsk" - спросить пользователя (вызывать диалог для ручного выбора значения)

"kAskIfNeed" - спросить пользователя если не удается автоматический выбор

strParamName - Имя табличного параметра - строка, содержащая имя табличного параметра, по которому осуществляется выборка.

strOperation - Операция фильтрации - строка, принимающая следующие значения :

"=" - равно

">=" - больше или равно

"<=" - меньше или равно

"~" - приближенно равно

">" - больше

"<" - меньше

"+" - вычисление выражения

Пример:

```
 LoadInCache(dr, L);
 SelectInCache("kFirst", "dr", "~", 14, "L", ">=", 100);
 LoadInCache(dr, L, b);
 SelectInCache("kFirst", "dr", "~", rdr, "L", "r;+", "L>=rMinLen
&& L<=rMaxLen && (L-b<=rTrLen)");
```

Диалоги объектов БД

Функция UniDialog

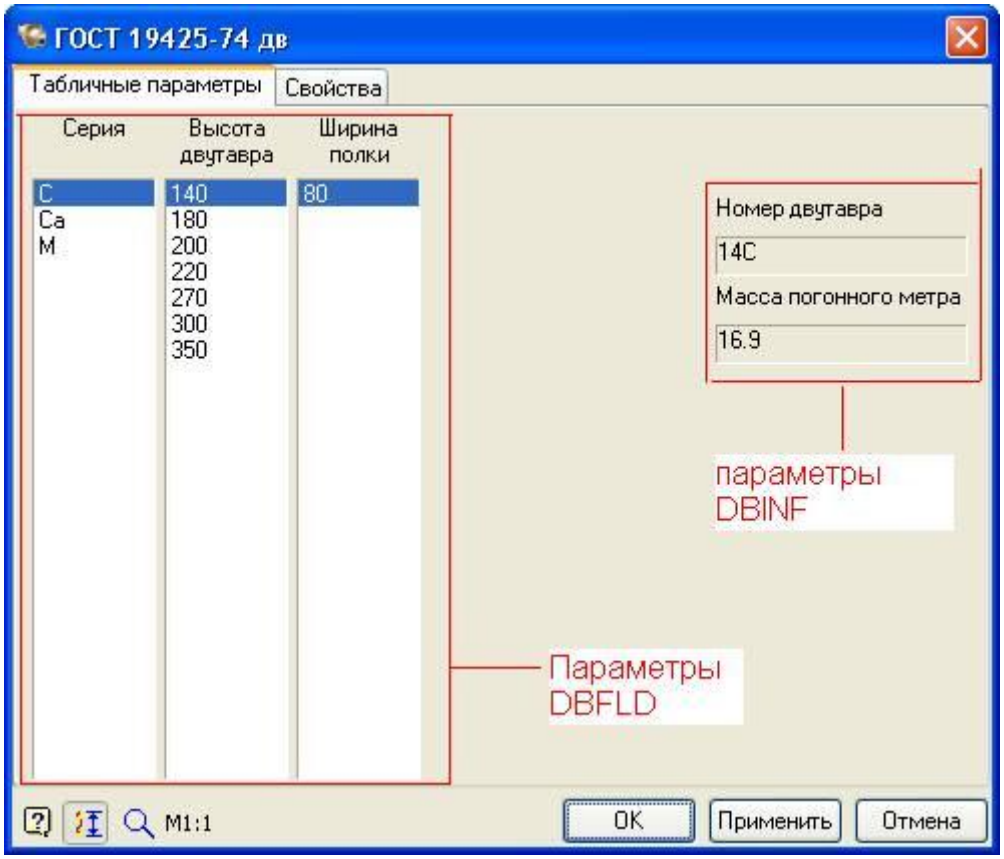
```
 UniDialog (
         [DBFLD, D, Dn, ... ,]
         [DBINF, Number,Seria,...,]
         [VFLD, rA, "A", rB, "B", ... ,]
         [BFLD, bKey1, "Ключ1", bKey2, "Ключ2",... ,]
         [RADIO, rKey1, "Вариант1", "Вариант2", ... ,]
         [VIDS, strDesignName, "Design1","Design2", ... ,]
         [TVIDS, lVidType, {"F", "T", "R", "L", "All"},]
         [VIEW, {"Vids", "Hdr", "None"}]
 )
```

Функция вызывает диалог редактирования стандартного объекта базы данных. Функция возвращает **IDOK**, если была нажата кнопка OK и **IDCANCEL**, если была нажата кнопка Cancel.

[] - блок необязательных параметров

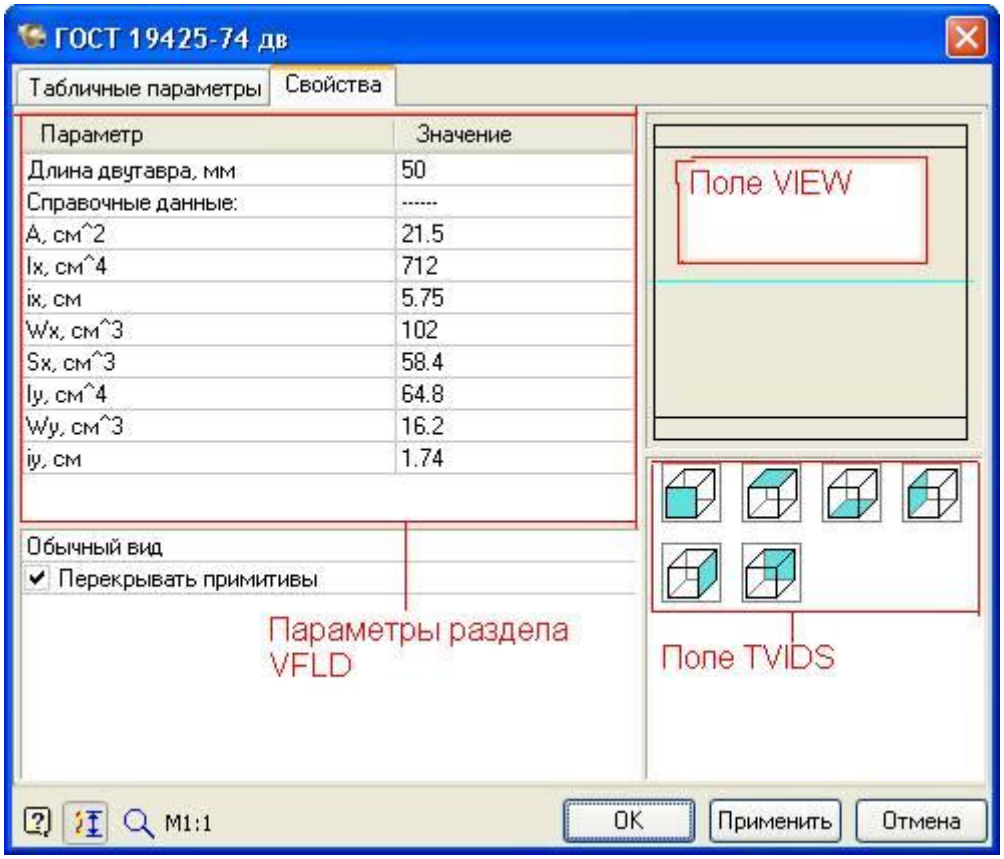
DBFLD - ключевое слово, после которого перечисляются табличные параметры

D,Dn, ... - имена табличных параметров.



VFLD- ключевое слово, после которого перечисляются параметры объекта, объявленные в секции *Public* или *Protected*.

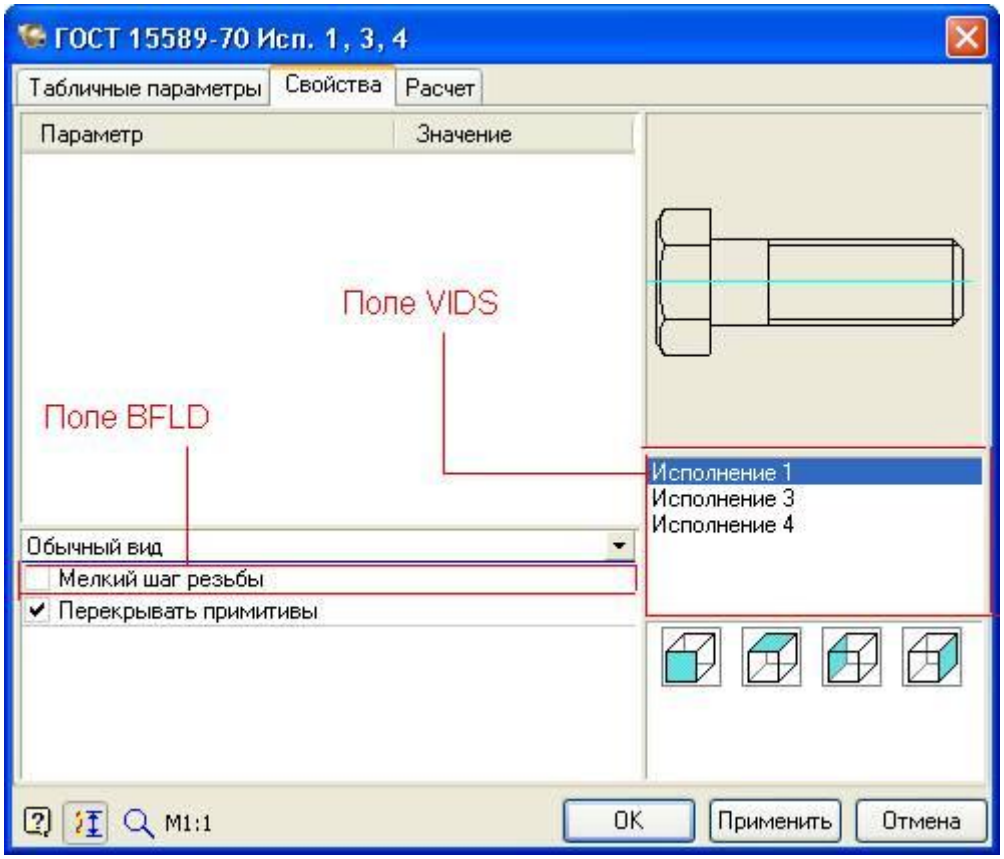
rA,rB, ... - имена параметров с комментариями.



BFLD - ключевое слово, после которого перечисляются переключатели - переменные, принимающие только значения 1 (ВКЛ) и 0 (ВЫКЛ).

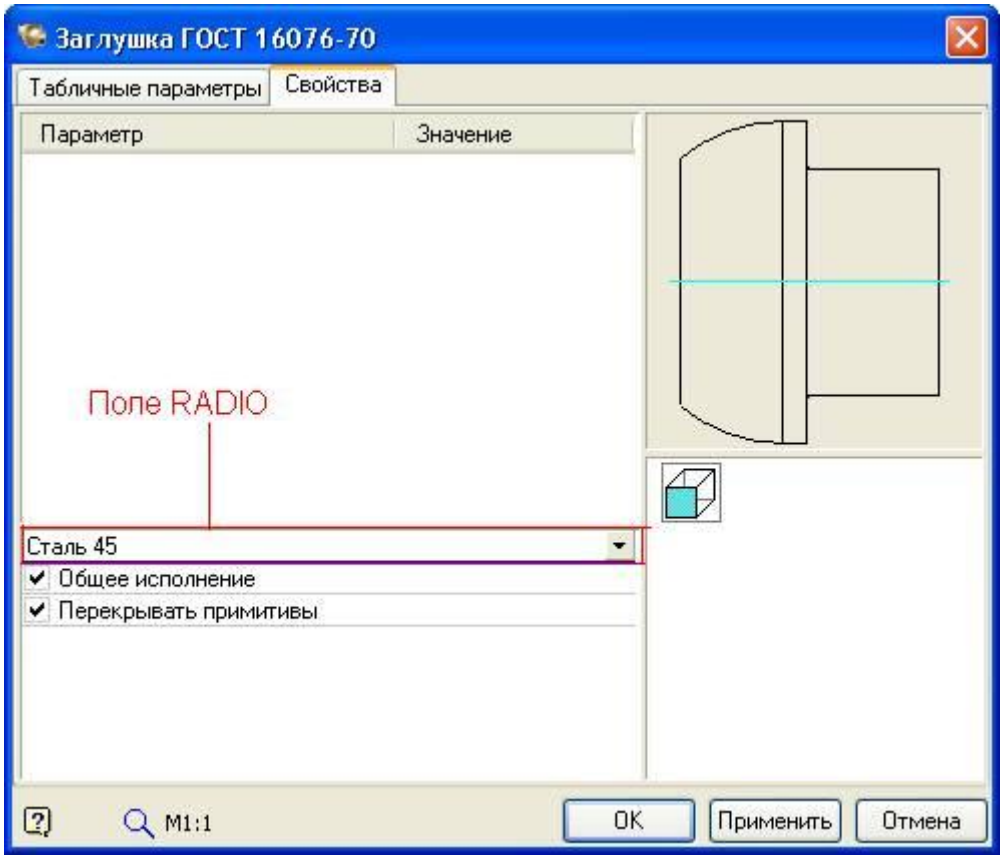
bKey1, "Ключ1",bKey2, "Ключ2" - имена переменных с названиями переключателей.

Блок параметров, начинающихся с ключевого слова *BFLD*, представляет собой объявление компонентов диалога, являющихся стандартными **CheckBox**(-ами) и имеющих два состояния: ВКЛЮЧЕНО (Checked) и ВЫКЛЮЧЕНО (Unchecked). Параметр-переменная, следующая за BFLD, в зависимости от состояния переключателя принимает значение 1 для ВКЛЮЧЕНО и 0 для ВЫКЛЮЧЕНО. В приложении такие переключатели широко используются для описания свойств стандартных объектов.



RADIO - ключевое слово, после которого перечисляются групповые переключатели - переменные, принимающие дискретные целые значения.

rKey1, "Вариант 1", "Вариант 2", ... - имена переменных групповых переключателей с названиями вариантов выбора.



Блок параметров, начинающихся с ключевого слова *RADIO*, представляет собой объявление компонентов диалога, являющихся стандартными выпадающими списками с постоянным числом неизменяемых строк. Параметр-переменная, следующая за RADIO, принимает значение индекса выбранной в ComboBox строки. Самая верхняя в строка имеет индекс 0, последняя равна числу строк минус 1. В приложении такие переключатели обычно используются для выбора материала стандартных объектов.

Для описанного ниже примера параметр bCommon принимает целочисленные значения 0 или 1, а параметр rd1 принимает значения 0, 1 или 2.

Параметры для BFLD и RADIO можно описывать и в секции Public и Protected. Перед вызовом диалога такие параметры должны быть проинициализированы (обычно в **OnInitialization**).

VIDS- ключевое слово, после которого перечисляются исполнения объекта

strDesignName, "Design1", "Design2" - имя переменной, хранящей название исполнения объекта с доступными названиями исполнений.

TVIDS- ключевое слово, после которого перечисляются виды объекта

lViewType- переменная, хранящая название вида объекта и принимающая значения только из множества {VFRONT, VRIGHT, VLEFT, VTOP, VBOTTOM,VBACK}.

"F" - обозначение вида «Фронтальный».

"R" - обозначение вида «Справа».

"L" - обозначение вида «Слева».

"T" - обозначение вида «Сверху».

"B" - обозначение вида «Снизу».

"All" - обозначение всех видов.

VIEW - ключевое слово, после которого следует тип слайдов из множества {"Vids", "Hdr", "None"}.

"Vids" - слайды исполнений объекта

"Hdr" - слайд заголовка объекта (Header)

"None" - нет слайдов

Примеры:

```
LoadInCache(dr);
UniDialog (
        DBFLD, dr,
        VFLD,
        Lthread, "Длина нарезанной части",
        Lhole, "Длина отверстия",
        d_hole, "Диаметр свободной части",
        VIDS, strDesignName, "All",
        TVIDS, lViewType,"All",
        VIEW, "Vids"
);
LoadInCache(Dn);
UniDialog (
        DBFLD, Dn,
        BFLD, bCommon, "Общего применения",
        RADIO, rd1, "Алюминиевый сплав",  "Сталь 45", "Бронза",
        VIDS, strDesignName, "All",
        TVIDS, lViewType, "All",
        VIEW, "Vids"
);
```

Пользовательские формы

```
        ShowForm("FormName");
```

Вызывает форму объекта с имененем **FormName.**

Функция MessageBox

```
MessageBox(StrMessage[,mb_Buttons = MB_OK, mb_Icons])
```

Функция вызывает стандартное всплывающее сообщение Windows.

StrMessage - строка сообщения

mb_Buttons - параметр, определяющий кнопки в MessageBox.

Может принимать одно из следующие значений:

MB_OK - только кнопка ОК

MB_OKCANCEL - кнопки ОК\Отмена

MB_YESNO - Кнопки Да\Нет

mb_Icons - параметр, определяющий иконки в MessageBox.

Может принимать одно из следующие значений:

MB_ICONWARNING - предупреждение

MB_ICONINFORMATION - информационное сообщение

MB_ICONERROR - ошибка

MB_ICONQUESTION - вопрос

В зависимости от нажатой клавиши функция возвращает следующие значения:

IDOK - была нажата кнопка ОК

IDCANCEL - была нажата кнопка Отмена

IDYES - была нажата кнопка Да

IDNO - была нажата кнопка Нет

Пример:

```
MessageBox("Ошибка при подключении!", MB_OK, MB_ICONERROR);
```

Сообщение в нотификатор

```
ShowMessage("Text"[, npIcon, objectID]);
```

Выводит нотификационное сообщение с указанным текстом **Text**

npIcon может принимать значения:

npSimple == -1, // без иконки

npUsual == 0, // "страница"

npWarning == 1, // "воскл. знак"

npCure == 2, // "крест"

npHint == 3, // "лампа"

npError == 4 //Ошибка

В случае указания objectID сообщение будет с кнопкой редактирования объекта обозначенного **objectID**

Примечание1: Сообщения с **objectID** нельзя посылать в процессе вставки или вызова диалога

Примечание2: На сообщения с ключом **npError** не действует настройка «Показывать всплывающие сообщения».

Функция ShowValue

```
ShowValue(StrMessage,Value);
```

Функция выводит значение переданного в качестве аргумента параметра (*value*), сопровождая его строкой *strMessage*, в окно специализированного отладчика InDebMon.exe. В качестве value может выступать любой параметр или переменная.

Пример:

```
ShowValue("pntOrigin", pntOrigin);
ShowValue("vecDirection", vecDirection);
ShowValue("WorkPlane WP1", WP1);
ShowValue("Dn", Dn);
ShowValue("*****************", 1);
ShowValue("OnConnect Start", 1);
```

Функции для работы с зависимостями

Функция	Комментарий
`ResetAllConstraint(idFrom = NULL)`	Сбрасывает зависимости, установленные в последнем OnConnect, когда используется вставка объекта с выбором нескольких объектов. *idFrom* - идентификатор объекта, от которого нужно сбросить зависимости. По умолчанию равен NULL, в этом случае сбрасываются все зависимости от всех объектов.
`SetParamConstraint( Parameter, idFrom, TYPE, expr, bBidirect = FALSE, idTo = objectID )`	Устанавливает параметрическую зависимость. *Parameter* - Имя параметра *idFrom* - идентификатор объекта, от которого ставится зависимость *TYPE* - Тип параметрической зависимости (пока используется только **EXPR**) *expr* - выражение, задающее параметр подключаемого объекта *bBidirect* - Флаг, TRUE - двухсторонняя

зависимость,

FALSE (по умолчанию) - односторонняя зависимость.

idTo - идентификатор объекта, к которому ставится зависимость

(по умолчанию равен **objectID** - идентификатор текущего объекта)

Пример:

```
 SetParamConstraint(rDn, obj, EXPR,
"obj.rDnE2", TRUE);
 SetParamConstraint(LTPlug, obj, EXPR,
"obj.rLT");
```

```
 SetGeomConstraint(
         TYPE,
         SUBTYPE,
         IdFrom,
         PlaneTo,
         PlaneFrom,
         expr,
         bBidirect =
FALSE,
         idTo = objectID
 );
```

Устанавливает геометрическую зависимость.

TYPE - Тип геометрической зависимости. Может принимать следующие значения:

INSERT - Вставка

MATE - Совмещение по плоскости

AXIS - Совмещение по оси

DIRECTION - Совмещение по направлению (угловая зависимость)

SUBTYPE - Подтип зависимости. Может принимать два значения:

CODIRECT - сонаправленная зависимость

CONTRDIRECT - противонаправленная зависимость

idFrom - идентификатор объекта, от которого ставится зависимость

PlaneTo - плоскость вставляемого объекта

PlaneFrom - плоскость подключаемого объекта

expr - выражение, задающее параметр зависимости (для **INSERT** - расстояние между плоскостями)

bBidirect - Флаг, TRUE - двухсторонняя зависимость,

FALSE (по умолчанию) - односторонняя зависимость.

idTo - идентификатор объекта, к которому ставится зависимость

(по умолчанию равен **objectID** - идентификатор текущего объекта)

Пример:

```
 SetGeomConstraint(INSERT, CODIRECT,
obj, WP1, obj.WP2, 0);
```

```
 IsFixedParam(Parameter,
bOnlyFixed =FALSE)
```

Функция используется в обработчике события *OnChangeParameters*.

Проверяет установлена ли зависимость на параметр и изменился ли он, если **bOnlyFixed** равно TRUE.

Вызов по умолчанию (с FALSE) рекомендуется использовать в проверке параметров объекта с постоянной геометрией (гайка), в этом случае функция возвращает 1, если параметр связан зависимостью и изменил свое значение, и 0 - в противном случае. Вызов (с TRUE) необходимо использовать для объектов с изменяющейся геометрией (труба, стакан), тогда функция вернет 1, если параметр связан зависимостью.

Пример:

```
if (IsFixedParam(WP1, TRUE)) {
    pntOrigin = Point(new.WP1);
    vecDirection = Vector(new.WP1);
}
```

Функции - обработчики событий, вызываемые ядром приложения

Функция	Комментарий
ActHeader	Описание объекта, объявление параметров, установка флагов. *Вызывается каждый раз при открытии объекта.*
OnInitialization	Инициализация объекта. *Вызывается каждый раз при обращении к объекту.*
BeforeConnect	Выполнение предварительных действий перед началом селектирования объектов во время

	вставки. *Вызывается во время вставки объекта.*
OnConnect	Установка зависимостей при подключении к другим объектам. *Вызывается при вставке объекта.*
OnChangeParameters	Установка новых значений параметров объекта другим объектом. *Вызывается при изменении связанных зависимостями параметров.*
OnDialog	Редактирование параметров объекта. *Вызывается при вызове диалога.*
OnMakeParameters	Окончательная подготовка всех параметров *Вызывается перед закрытием объекта.*
SetGripPoint	Подготовка грип-поинтов. *Вызывается при селектировании объекта.*
OnMoveGripPoint	Реактор на изменение положения grip-points. *Вызывается при сдвигегrip-point.*
OnAddObject	Установка обратных зависимостей к подключаемым объектам. *Вызывается при вставке с выбором другого объекта.*
OnDialogChanged	Интерактивная обработка вводимых значений. *Вызывается на каждое изменение параметров объекта в диалоге.*
OnInitSelect	Инициализация строки приглашения и всплывающего меню. *Вызывается перед каждым шагом цикла динамического выбора параметров.*
OnSelectParam	Обработка вводимых значений с клавиатуры и мыши. *Вызывается в теле цикла динамического выбора параметров.*
OnMenu	Обработчик меню. *Вызывается в ответ на выбор пункта меню, созданного с помощью функцииSetMenu..*

Функции для работы с идентификаторами объектов

Функция	Комментарий

`setWorkId(index, id)`	Сохраняет указанный идентификатор в специальном хранилище по указанному индексу.
	index - целое число (index >= 0).
	id - идентификатор объекта.
	Каждый объект приложения при создании получает уникальный идентификатор, подобный GUID в Windows. В скрипте доступ к идентификатору объекта Obj1 можно получить с помощью выражения Obj1.**objectID**.
	Пример:
	`id1 = getWorkId(0);`
	`id2 = getWorkId(1);`
	`If (id1 == id2) MessageBox("Попытка подключения к одному объекту дважды!", MB_OK | MB_ICONWARNING);`

`getWorkId(index);`	Возвращает сохраненный ранее идентификатор. *index* - целое число (index >= 0). **Пример:** `if (obj.strTheName == "MyEnabledObject") { setWorkId(0,obj.objectID); }`
`getObjData(WorkId,ObjName)`	Функция получает public-параметры с объекта базы данных, имеющего идентификатор *id*. В случае успешного выполнения возвращает 1, иначе 0. Доступ к полученным параметрам осуществляется с помощью выражения *ObjName*.Param1 и т.д.

Пример:

```
 getObjectConnectedTo(idOut,
objToSerch, strExpression,
ParamName);
```

Производит поиск объектов соединенных зависимостью с переменной ParamName объекта objToSerch, удовлетворяющих условию strExpression.

Условие может быть пустым.

Другие функции

```
 setGlobalParam(var,
val);
```

Устанавливает глобальную переменную **var** в значение **val**

Пример:

```
 if(rLen2 <= B/4){
         SetGeomConstraint(INSERT,
CODIRECT, obj, WP1, obj.WP2, 0);
         setGlobalParam(rSubType,
CONTRDIRECT);
 } else if(abs(rLen1) <=  B/4) {
         SetGeomConstraint(INSERT,
CODIRECT, obj, WP1, obj.WP1, 0);
         setGlobalParam(rSubType,
CODIRECT);
         } else {
                 SetGeomConstraint(INSERT,
rSubType, obj, WP1, obj.WP1, rLen1);
 };
```

```
 GridRound(b)
```

Выравнивает значения по сетке. Параметр сетки задается в параметрах или в быстрых параметрах (Ctrl+Shft+Q по умолчанию);

Пример:

```
 a = GridRound(b);
```

Т.е. если шаг сетки 5 a, b ==12.4, то функция вернет 10; Если же b >= 12.5 то функция вернет 15.

```
 rDiameter = GridRound(abs(rYcoord*2));
```

`paramByName(strName)` Возвращает значение параметра по его имени:

Пример:

```
strName = GetNearestPlane(obj);
plane = paramByName(strName);
```

`getChildrenCount()` Возвращает количество объектов-потомков

`getChildId(nChild);` Возвращает идентификатор потомка **nChild**

Объект - центровые отверстия для валов по ГОСТ 14034-74

Пример объекта

Исходная графика

Таблица

```
 SVersion = 2;//Первая строка скрипта, определяющая версию
интерпретатора
 //Описание объекта. Будет отображаться в панели свойств
 //кроме того, можно сортировать по этому полю объекты в
 //табличных данных и mcqs
 ObjectDescription = "Центровое отверстие";
 //Это функция описания объекта
 function ActHeader {
    NPart = 2; //Количество запросов на динамический выбор
параметров
    //Этот раздел содержит описания открытых параметров объекта,
    //видимых для других объектов
    Public(
        //Определяющий параметр типоразмера отверстия - его
диаметр
        d, "Диаметр отверстия",
        //Плоскость для установки зависимости на конец вала
        WP1, "Плоскость начала",
        rValType,"Тип вала",//Тип вала, к которому коннектиться
отверстие
                            //Задается для обеспечения
перекрытия на
                            //видах с разрезом и без разреза
        Lhole,"Глубина отверстия", //Это общая глубина отверстия
        Lthread,"Длина резьбы",  //Параметр длины резьбы
        //Параметры, специфичные для исполнений
        DF,"Диаметр вала для формы F",
        DH,"Диаметр вала для формы H"
    );
    //Закрытые параметры, использующиеся для построения графики
    //и вычисления ограничений других параметров
```

```
    Protected( seted,  freeLen,bBlind,rSuppressPointHeight);
    //в этом разделе перечисляются те из открытых параметров,
    //которые будут доступны извне для редактирования
    //это только объявление, поскольку конкретные действия при
    //изменении параметров извне определяются в функции
    //OnChangeParameters
    Changeable( d, WP1,DF,DH,rValType,Lhole,Lthread);
    //Показывать диалог вставки перед динамическим выбором
параметров
    OnDlgBeforeSelectParam = 1;
    //Не регенерировать контур подавления после вставки
    ContourOnLine = 0;
    //этот элемент является частью вала
    IsAValPart = 1;
    //Причем является встраиваемой частью вала
    InsidePart = 1;
    //этот элемент - отверстие ( эта переменная используется для
    //образмеривания в инвенторе)
    IsAHole = 1;
 };
 //Инициализация параметров объекта
 function OnInitialization{
    //Загружаем таблицу в кэш
    //Параметры таблицы из текущей строки всегда доступны в
скрипте
    //Поэтому загружается только один параметр d
    LoadInCache( d );
    //Переменная seted имеет по умолчанию значение UnknownValue
    //поскольку инициализируется с этим значением
    //Поэтому если мы поставим следующую проверку, то данный
блок
    //операторов будет выполнен только один раз:
    if(seted == UnknownValue) {
        seted=1;//Устанавливаем seted в 1, чтобы условие не
выполнялось
              //еще раз
        //Задаем начальные значения параметров
        rValType = 1;//Отверстие ставится на участок вала
        rZOrder=4220;//Порядок следования - начальное значение
        //Выбираем из таблицы начальные значения параметров
        SelectInCache( "kFirst", "Form","~","F H","d", "~", 10
);
        //Т.е. согласно синтаксису оператора - выбирается первая
запись
        //в таблице, где параметр Form равен "F H", и параметр
        //диаметр отверстия d приближенно равен 10
        //Устанавливаем, какое исполнение объекта будет
изначально
        //Отображаться
        strDesignName = "Форма F";
        //Задаем начальные значение произвольных параметров
        Lhole = 30; //Длина отверстия
        Lthread = Lhole-d/2; //Это минимально возможная длина
```

```
резьбы
        //Переменная, определяющая, является ли отверстие
глухим.
        //По умолчанию - глухое.
        bBlind=TRUE;
        //Эта переменная определяет значения параметра,
задающего относительную
        //высоту точки конца конуса подавления. (см по чертежу)
        rSuppressPointHeight = tg(30)*(d-1.082*p)/2;
    };
 };
 //Функция установки ручек
 function SetGripPoint{
    NGrip = 3;//Всего 3 ручки
    pntGrip0 = pntOrigin;//Первая - в точке вставки
    //Вторая ручка - ручки длины резьбовой части
    pntGrip1 = pntOrigin+vecDirection*Lthread;
    //Третья ручка - это ручка общей длины отверстия
    pntGrip2 = pntOrigin+vecDirection*Lhole;
 };
 //Эта функция определяет реакцию объекта на перемещение ручек
 function OnMoveGripPoint {
 //Сюда приходит переменная NMovingGrip, равная индексу
сдвигаемой
 //ручки нумерация в массиве начинается с нуля.
    //При редактировании первой ручки просто переместить деталь
туда
    //где будет курсор с этой ручкой. Т.е. переместить точку
вставки
    // туда, где будет первая ручка
    if (NMovingGrip == 0) {
        pntOrigin = pntGrip0;
    };
    //Это мы растягиваем длину (вторая ручка)
    //Здесь определяется длина резьбовой части в зависимости от
текущего
    //исполнения отверстия
    if (NMovingGrip == 1) {
        //Определяем расстояние растягивания, как длину вектора,
        //определяемого точками ручки1 и ручки0
        rDistance = vecLen(pntGrip1-pntGrip0);
        //Для исполнения (формы отверстия) "P"
        if (Form == "P") {
           //Длина резьбы должна быть ограничена снизу размером
           // (L-d/2). Очевидно, что параметру длины резьбы
Lthread
           //Будет присвоено значение относительного положения
курсора,
           //если оно больше (L-d/2).
           Lthread = max (rDistance,L-d/2);
        };
        //Для исполнения "F H"
```

```
        if (Form == "F H") {
           //То же самое для этого исполнения длина резьбы
должна быть
           //ограничена снизу размером (l+d/2).
           Lthread = max(rDistance,l+d/2);
        };
        //Наконец, в этом условии мы ограничиваем длину резьбы
        //сверху, поскольку длина резьбы не может быть больше
        //длины отверстия. Поэтому мы будем растягивать все
отверстие
        //в соответствии с новым диаметром резьбы.
        if(Lthread>Lhole-d/2) {
           //Т.е. при изменении длины резьбы больше длины
отверстия
           //присваиваем новую длину отверстия равной
           //длине резьбы плюс пол-диаметра
           Lhole=Lthread+d/2;
        };
    };
    //В этом блоке рассматривается перемещение третей ручки.
    //Результатом ее редактирования является увеличение длины
отверстия,
    //не затрагивающее резьбу.
    if (NMovingGrip == 2) {
        //В данном случае текущее расстояние определяется, как
        //длина вектора, образованного точками ручки0 и ручки2
        rDistance = vecLen(pntGrip2-pntGrip0);
        //Для исполнения"P"
        if (Form == "P") {
           //Длина отверстия ограничивается снизу параметром L
           Lhole = max (rDistance,L);
        };
        //Дляисполнения"F H"
        if(Form == "F H") {
           //Длина отверстия ограничивается параметрами (l+d)
           Lhole = max(rDistance,l+d);
        };
        //При изменении длины отверстия меньше длины резьбы
        //Перемещаем границу резьбы
        if(Lhole>Lthread+d/2) {
           //То есть, если длина отверстия меньше, чем
Lthread+d/2,
           //то пересчитываем новую длину резьбы, которая меньше
           //текущей длины отверстия на d/2
           Lthread = Lhole-d/2;
        };
    };
 };
 //Функция окончательного вычисления параметров
 function OnMakeParameters {
    //Имя, фамилия и отчество объекта
    strTheName = "CenterHole";//Отверстие центровое
    strTheType = "ArborParts";//Относится к деталям валов
```

```
    strTheSubType = "Simple";//Тип отверстия - простое
    //Задается положение плоскости WP1. Это просто плоскость,
    //с базовой точкой в точке вставки и вектором нормали,
    //совпадающим с вектором направления вставки
    WP1 = Plane( pntOrigin, vecDirection );
    //Реализуем перекрытие для различных случаев:
    //Если переменная rValType, определяющая тип участка вала,
на
    //который устанавливается отверстие, равна 1 (это простой
    //участок вала), то порядок перекрытия будет равен 4220
    //(это число задается на основании значения порядка
следования
    //участков вала, так, чтобы отверстие перекрывало вал своим
    //контуром подавления
    if(rValType ==1) rZOrder = 4220;
    //Для внутренних участков вала и других участков, где
rValType не равно 1
    else rZOrder = 2150;
    //Порядок следования будет другой
    //Это водятся ограничения параметров в зависимости от
исполнений
    //Для исполнения "P", точнее, для текущей записи в таблице,
    //параметры которой соответствуют форме отверстия "P".
    if (Form == "P") {
        //Ограничиваем длину отверстия параметром L
        Lhole = max (Lhole,L);
        //Ограничиваем длину резьбы параметром (L-d/2)
        Lthread = max(Lthread,L-d/2);
        //Устанавливаем переменную strDesignName в "Форма P"
        //т.е. выбираем то, двухмерное исполнение, которое
соответствует
        //текущим параметрам в таблице
        strDesignName = "Форма P";
    };
    //Тоже самое для формы "F H"
    if (Form == "F H") {
        //Ограничиваем параметры длины отверстия и длины резьбы
        Lhole = max(Lhole,l+d);
        Lthread = max(Lthread,l+d/2);
        //Это перестраховка, если все-таки пользователь выбрал
        //исполнение "P" в двухмерных видах, то, поскольку в
таблице
        //выставлен параметр Form в значение "F H".
        //В общем, если выбрано неправильное исполнение, то
устанавливаем
        //первое правильное.
        if (strDesignName=="Форма P"){
           strDesignName = "Форма F";
        };
    };
    //Ограничива��м длину резьбы сверху в зависимости от
    //общей длины отверстия
    Lthread = min(Lthread,Lhole-d/2);
```

```
    //Отверстия формы P всегда глухие
    if(Form == "P"){bBlind = TRUE;};
    //Это нужно для того, чтобы правильно отрисовывался контур
подавления
    //В графике есть контур подавления, положение одной из точек
    //которого задается параметром rSuppressPointHeight
    //Вот и относительно того, выбрал пользователь отображение
    //глухого или сквозного отверстия, устанавливаем  для
параметра
    //rSuppressPointHeight необходимое значение
    if (bBlind==TRUE) {//Для глухого отверстия
        rSuppressPointHeight = tg(30)*(d-1.082*p)/2;
    }else{//Для сквозного отверстия (т.е. не будет конуса от
сверла)
        rSuppressPointHeight = 0;
    };
    //Это просто задаем строку спецификации
    //В зависимости от исполнения добавляем необходимую букву
    //К обозначению отверстия.
    if (strDesignName=="Форма F") {strLetterForm = "F";};
    if(strDesignName=="ФормаH") {strLetterForm = "H";};
    if(strDesignName=="ФормаP") {strLetterForm = "P";};
    //Параметр strPartName - это строка спецификации отверстия.
    //Кроме того, он же выводится в хинте.
    strPartName= "Отв. центр. "+ strLetterForm+" M"+d+"
ГОСТ14034-74";
    //Форматирование строки производится автоматически, т.е.
    //при сложении строкового значения с
    //числовым число автоматически преобразуется в строку
 };
 //Эта функция определяет вид диалога вставки отверстия
 function OnDialog {
    //Загружаем параметры в кэш работы с таблицей
    LoadInCache( d );
    //Вызываем диалог с соответствующими ключами
    UniDialog(
        //Это определяющие поля таблицы отверстия
        //Форма и диаметр. Выводятся как определяющие табличные
параметры
        DBFLD, Form,d,
        //Диаметр вала для формы F и формы H
        //выводятся только в качестве справочных параметров
        DBINF,DF,DH,
        //Параметры, задаваемые произвольно (с учетом
ограничений, конечно)
        //Выводятся на закладке свойств
        VFLD,
           Lthread,"Глубина резьбы",
           Lhole, "Глубина отверстия",
        //Признак глухое или сквозное отверстие
        //Выводится на вкладке свойств в виде галочки
        BFLD,
           bBlind,"Глухое",
```

```
        //Варианты выбора исполнений. Отображаются все
исполнения
        VIDS,strDesignName,"All",
        //Отображаются все распознанные виды
        TVIDS,lViewType,"All",
        //Отображаются превью
        VIEW,"Vids"
    );
 };
 //Эта функция выполняется перед динамическим выбором параметров
 function OnInitSelect {
    //В эту функцию приходит переменная rPart - порядковый номер
    //Запроса на выбор динамических параметров. отсчет
начинается с 0
    if (rPart==0) {
        //Для первого случая указываем диаметр резьбы
        //Выводится соответствующая подсказка
        strPromt = "Укажите глубину резьбы";
    };
    if (rPart==1) {
        //Для второго случая указывается глубина отверстия(она
же-
        //общая длина).
        strPromt = "Укажите глубину отверстия";
    };
 };
 //Функция определяет поведение объекта при динамическом выборе
параметров
 function OnSelectParam {
    //Для первого цикла выбора параметров. В принципе,
ограничения
    //те же, что и для ручек.
    //Сюда приходят переменные rXcoord и rYcoord - относительная
    //абсцисса и ордината курсора
    if (rPart==0){
        //Параметр расстояния определяется
        //как модуль относительной абсциссы
        rDistance = Abs(rXcoord);
        //Ограничения по выбору параметров в зависимости от
исполнения
        if(Form == "P") {
           Lthread = max (rDistance,L-d/2);
        };
        if(Form == "F H") {
           Lthread = max(rDistance,l+d/2);
        };
        if(Lthread>Lhole-d/2) {
           Lhole=Lthread+d/2;
        };
    };
    //Для второго запроса
    if (rPart==1){
        //Параметр расстояния
```

```
        rDistance = Abs(rXcoord);
        //Ограничения.
        if(Form == "P") {
           Lhole = max (rDistance,L);
        };
        if(Form == "F H") {
           Lhole = max(rDistance,l+d);
        };
        if(Lhole<Lthread+d/2) {
           Lhole = Lthread+d/2;
        };
    };
 };


 //Эта функция выполняется при изменении параметров в диалоге
вставки
 function OnDialogChanged {
 //функция добавляет интерактивность в вид диалога.
 //В данном случае нам необходимо скрывать исполнения и галку
"Глухое"

    //Оператор ShowDesign скрывает или отображает исполнения.
    //в зависимости от значения первого параметра (1 -
показывает,
    //0 - скрывает.
    //Оператор ShowBool скрывает и отображает переключатели
(галочки)

    //Для начала, показываем все исполнения и галку
    ShowDesign(1, "Форма F");
    ShowDesign(1, "Форма H");
    ShowDesign(1,"ФормаP");
    ShowBool (1,"Глухое");

    //Если выбрана такая строчка в таблице, где d3 равно 0, то
    //Нужно скрыть исполнение "Форма H". (так задано в госте,
    //что данное исполнение не используется при таких значениях
    //параметров
    if( d3==0 )  ShowDesign(0, "Форма H");

    //Если выбрана строка, соответствующая форме отверстия "P",
то
    //нужно скрыть два других исполнения(чтоб их нельзя было
выбрать)
    //И галку "глухое", поскольку форма P всегда глухое.
    if(Form == "P") {
        ShowDesign(0,"ФормаH");
        ShowDesign(0,"ФормаF");
        ShowBool(0,"Глухое");
    };

    //Ну а для выбранных Форма "F H", нужно только скрыть
```

```
    //исполнение "Форма P"
    if (Form == "F H") {
        ShowDesign(0,"Форма P");
    };

 };

 //Функция OnChangeParameters выполняется при внешнем
воздействии
 //на объект. Это либо попытка установить зависимости, либо
прямое
 //изменение параметров объекта из панели свойств
 //Эта функция определяет реакцию объекта на такое изменение.
 function OnChangeParameters {
 //сюда приходит объект new, представляющий из себя
 //такой же объект с новым значением открытых параметров.

    //Произвольным переменным мы можем сразу присвоить новые
значения
    rValType = new.rValType;
    Lhole=new.Lhole;
    Lthread = new.Lthread;

    //А здесь последовательно задается, как выбирать строчку из
таблицы,
    //соответствующую новому значению параметров.
    //Проверяем, изменился ли параметр d
    if(new.d != d ){//т.е. если новое значение параметра не
равно старому
        //То загружаем таблицу в кэш
        LoadInCache(d);
        //И выбираем из нее первую попавшуюся запись, с
параметром
        //d, приближенно равным новому значению.
        SelectInCache("kFirst", "d", "~", new.d);
    //Дальше проверяем, не изменились ли другие параметры
    }else if (DH != new.DH) {//Изменился диаметр вала исполнения
H
        //Загружаем
        LoadInCache(DH);
        //Выбираем первую запись
        SelectInCache( "kFirst", "DH","~",new.DH );
    } else if (DF != new.DF) {//Изменился диаметр вала
исполнения F
        //Загружаем
        LoadInCache(DF);
        //Выбираем первую запись с приближенно равным значением
        SelectInCache( "kFirst", "DF","~",new.DF );
    };

    //Обработка по умолчанию завершена удачно.
    //Т.е. отправляем объекту, изменявшему параметры, что
    //необходимые значения параметров установлены.
```

```
    //Для этого присваиваем переменной Handled Значение
OBJ_HANDLED
    Handled = OBJ_HANDLED;

    //Если после выбора из таблицы диаметр вала не установился
    //в новое значение, то, возможно требуется циклическое
обновление
    //Отправляем изменяющему объекту значение переменной
    //Handled равное OBJ_WARNING
    if( d != new.d  ) {
        Handled= OBJ_WARNING;
    };

    //В этом блоке операторов пересчитывается новое положение
объекта
    //при изменении его рабочей плоскости

    //Переменная fix для проверки возможности перемещения.
    //Вообще, в данном случае ее можно было не использовать,
    //поскольку отверстие не меняет свою геометрию при изменении
    //плоскостей.
    //Этот блок генерируется скрипт-мастером, поэтому он имеет
    //Такой унифицированный вид

    fix=0;

    //Проверяем, установлена ли зависимость, и изменилась ли
    //плоскостьWP1
    if( IsFixedParam(WP1) ) {
        //Увеличиваем значение переменной fix, для последующей
проверки
        fix = fix+1;
        //Сохраняем старое значение вектора вставки.
        vecXOld=vecDirection;
        //Присваиваем новое значение вектору вставки,
совпадающее
        //с новым положением плоскости WP1
        vecDirection = Vector(new.WP1);
        //И перемещаем точку вставки отверстия в новое положение
        //плоскостиWP1
        pntOrigin = Point(new.WP1);

        //Восстанавливаем систему координат на основании
        //старых и новых значений векторов
        restoreBasis(vecXOld, vecPlane, vecDirection);
    };
    //Если вдруг отверстие с неизменяемой геометрией
    //объекты начнут растягивать в разные стороны, то
    //возвращаем ошибку
    if( fix>1 ) {
        Handled= OBJ_ERROR;
    };
 };
```

```
 //Эта функция выполняется перед каждой попыткой установить
зависимость
 function BeforeConnect {
    //В ней мы просто сбрасываем последние зависимости чтобы
    //в следующей функции установить новые
    ResetLastConstraint();
 }
 //Эта функция устанавливает зависимости. Точнее,
 //она определяет как конкретно будет вести себя объект при
 //присоединении к другому объекту
 function OnConnect {
    //Проверка по значению переменной rPart, определяющей
    //порядковый номер запроса на присоединение к детали.
    //Т.е. если зависимость устанавливается в два этапа,
    //или больше, то есть возможность установить на одну деталь
    //зависимости от двух и более деталей.
    //Но в данном конкретном случае зависимости устанавливаются
только
    //от одной детали, поэтому запрос только один, и проверка
    //чтобы rPart была равна 0
    if(rPart == 0){
        //В функцию onConnect приходит объект obj - это та
деталь,
        //к которой мы пытаемся присоединиться. Мы можем с
помощью квалификатора
        //обращаться к параметрам этой детали, чтобы проверить,
нужно ли
        //автоматически устанавливать зависимости, или нет.
        //Проверяем, является ли деталь, к которой пытаемся
присоединиться,
        //валом:
        if(obj.strTheType== "Arbor"){
           //Дальше - будем устанавливать зависимости
относительно текущего исполнения
           //отверстия. Т.е. разные исполнения реализуют по-
разному зависимости на вал.
           if (Form == "F H") {
              //Устанавливаем для отверстия тип вала такой же,
как и у
              //участка вала, к которому присоединяемся (это
необходимо для реализации
              //перекрытия
              rValType = obj.rValType;
              //Находим ближайшую плоскость к плоскости WP1
отверстия:
              strNearestPlane = nearestPlaneName(WP1, obj.WP1,
obj.WP2);
              //При этом учитываются только плоскости WP1 и WP2
объекта
              //т.е. если ближайшей являются не эти плоскости,
то strNearestPlane
              //будет неопределенна.
              //далее - проверяем тип вала. для обычных участков
```

```
(наружных):
              if(obj.rValType != 1)
                  //Устанавливаем геометрическую зависимость
                  //типа "вставка", сонаправленную. К тому
объекту, к которому
                  //присоединяемся, для плоскостей WP1 детали и
найденной
                  //ближайшей плоскости. Расстояние между
плоскостями - 0 мм.
                  //Зависимость однонаправленная
                  SetGeomConstraint(INSERT, CODIRECT, obj, WP1,
strNearestPlane, 0, FALSE);
              //Для внутренних участков валов:
              else
                  //Устанавливаем зависимость с теми же
параметрами,
                  //но противонаправленную.
                  SetGeomConstraint(INSERT, CONTRDIRECT, obj,
WP1, strNearestPlane, 0, FALSE);
              //Устанавливаем параметрические зависимости
относительно текущего исполнения
              //отверстия.
              //Для отверстия формы F необходимо приравнять
              //параметр rDiameter(диаметр вала) к диаметру DF
таблицы отверстия
              if (strDesignName == "Форма F"){
                  //таким образом, если проверка выполняется, то
                  //устанавливаем параметрическую зависимость на
выражение.
                  //приравнивая rDiameter к DF

SetParamConstraint(DF,obj,EXPR,"obj.rDiameter");
              };
              //Для формы H такая же проверка, но устанавливаем
зависимость
              //уже на диаметр DH отверстия.
              if(strDesignName == "ФормаH") {

SetParamConstraint(DH,obj,EXPR,"obj.rDiameter");
              };
              NoVectorSelect = 1;//Устанавливая переменную
NoVectorSelect в 1
                        //отключаем выбор вектора направления
для детали
                        //поскольку уже присоединились и
установили зависимости,
                        //то выбирать дополнительно его не
нужно.
              Handled = OBJ_HANDLED; //устанавливаем переменной
Handled
                  //значение OBJ_HANDLED, т.е. говорим о том,
что зависимости
                  //установлены успешно.
```

```
            };
        };
    };
 }
```

Заглушка ГОСТ 16076-70

Объект

Таблица

Графика

Скрипт:

```
SVersion = 2;
 ObjectDescription = "16076-70";
 functionActHeader{
     NPart=0;
     Public(
          //Из открытых параметров нужны только определяющий
номинальный
          //диаметр, рабочая плоскость для присоединения
          //материал, и справочная масса.
          Dn, @NOMINAL_DIAMETER,
          massa, @MASS,
          sw0,@MATERIAL,
          WP1, @WORKING_PLANE1
     );
     Protected( seted, d, D, D1, D2, D3, D4, l, l1, L ,type,
type1);
     Changeable( Dn, WP1 );
     OnDlgBeforeSelectParam = 1;
     ShowWhenSelPnt = 1;
     ContourOnLine = 0;
 };
 function OnInitialization {
     LoadInCache( Dn, d, D, D1, D2, D3, D4, l, l1, L, massa );
     if(seted == UnknownValue) {
          seted=1;
          rZOrder=100;
          //В 6й версии все названия исполнений на английском
          //Исполнение1 = Implementation 1
          strDesignName = "Implementation 1";
          rd0=1;
          sw0=1;
          SelectInCache( "kFirst", "Dn", "~", 18 );
     };
 };
 function SetGripPoint {
     NGrip = 1;//Одна ручка в точке вставки
     pntGrip0 = pntOrigin;
 };
 function OnMakeParameters {
```

```
     //Все локализуемые строки задаются ресурсными ссылками
     //которые начинаются с собаки.
     //Соответственно отображаться они будут в зависимости от
     //текущего языка приложения.
     //
     strTheName = @BLIND_GOST_16076_70;
     strTheType = @ON_INTERNAL;
     strTheSubType = @ON_INTERNAL;
     WP1 = Plane( pntOrigin+vecDirection*(0), vecDirection );
     if(rd0==0) type="022";
     if(rd0==1) type="012";
     if(sw0==1) type1="A"; else type1="";
 };
 function OnDialog {
     LoadInCache( Dn, d, D, D1, D2, D3, D4, l, l1, L, massa );
     UniDialog(
          //Вид диалога - определяющий номинальный диаметр
          //и справочная масса
          DBFLD, Dn,
          DBINF, massa,
          //Галка общего исполнения
          BFLD, sw0, @COMMON_IMPLEMENTATION,
          //Варианты материала
          RADIO, rd0, @STEEL_45, @12X18H9T__X18H9T_,
          //Эти варианты влияют на строку спецификации заглушки
          VIEW,"Vids");
 };
 function OnDialogChanged {
 };
 function OnChangeParameters {
     if( (Dn != new.Dn) ) {
          LoadInCache( Dn, d, D, D1, D2, D3, D4, l, l1, L, massa
);
          //при изменении диаметра выбираем из таблицы новое
значение
          SelectInCache( "kFirst", "Dn","~",new.Dn );
     };
     Handled = OBJ_HANDLED;
     if( (Dn != new.Dn) ) {
          Handled = OBJ_WARNING;
     };
     fix=0;
     if( IsFixedParam(WP1) ) {
          fix = fix+1;
          vecXOld=vecDirection;
          vecDirection = Vector(new.WP1);
          pntOrigin = Point(new.WP1) - vecDirection*(0);
          restoreBasis(vecXOld, vecPlane, vecDirection);
     };
     if( fix>1 ) {
          Handled = OBJ_ERROR;
     };
 };
```

```
 function BeforeConnect {
     ResetLastConstraint();
 }
 strPartName = @SPHERE_CAP_+Dn+" - "+type+type1+@_GOST_16076_70;
```

Фланцы стальные приварные встык ГОСТ 12821-80

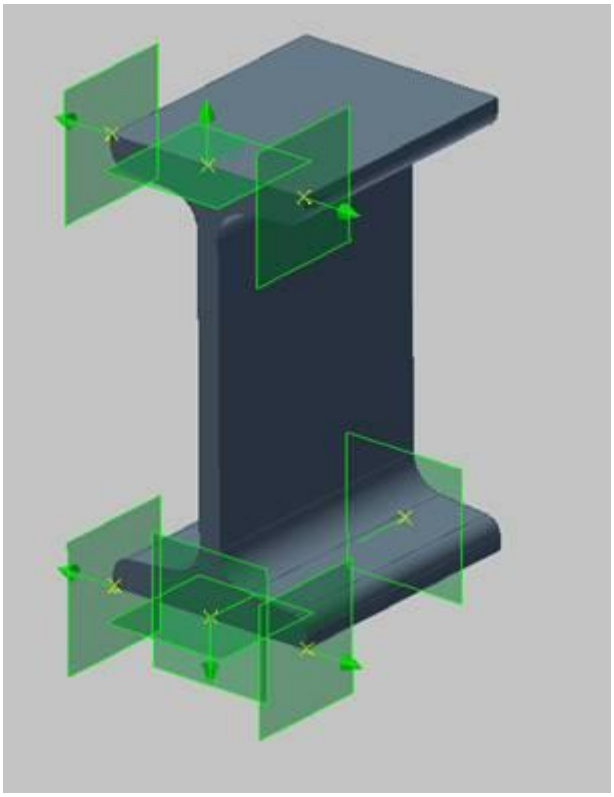
Скрипт фланца (только процедуры BeforeConnect и OnConnect)
остальное можно посмотреть в базе данных стандартных деталей.

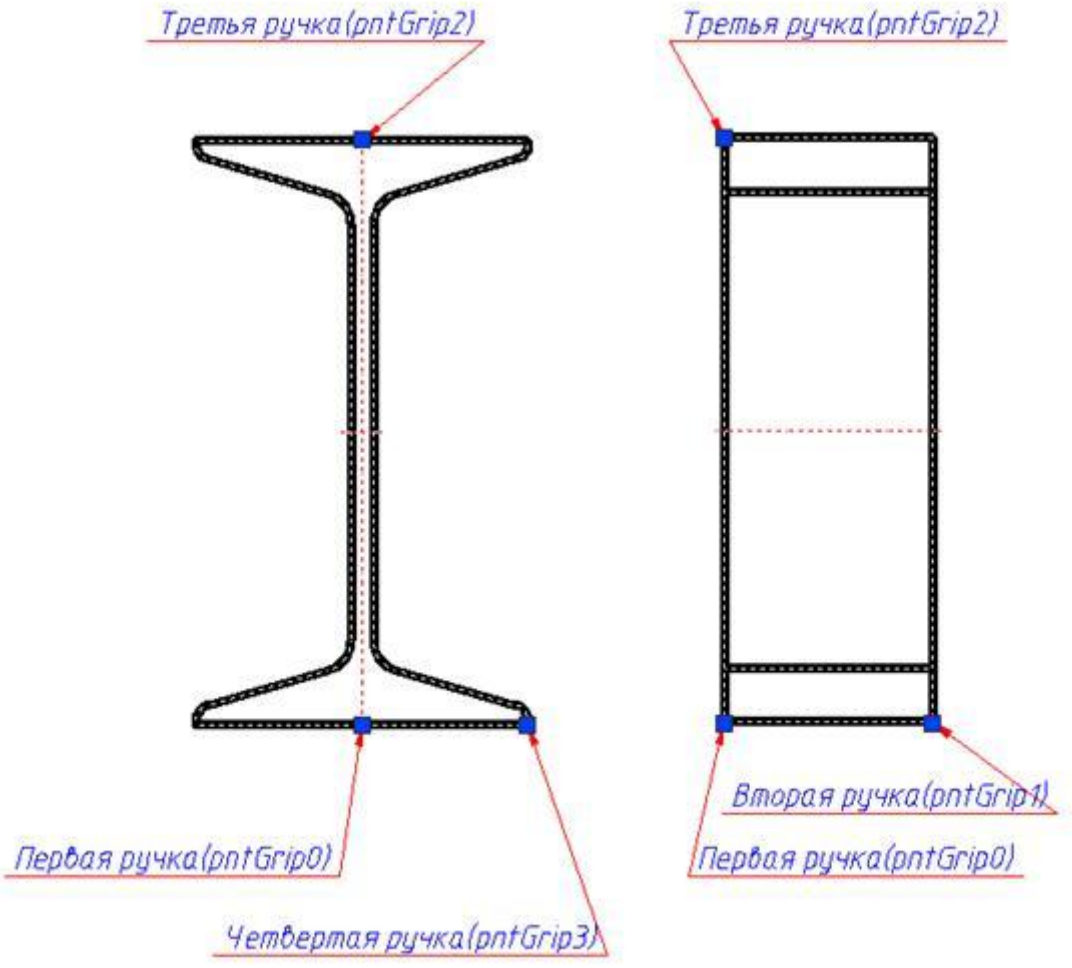
```
//Для фланца необходимо, чтобы геометрические зависимости
 //были установлены от прокладки, а параметрические -
 //от второго фланца.
 function OnConnect {
     //При установке зависимостей от нескольких объектов мы
сначала
     //сохраняем информацию об объектах,
     //а затем ее используем для установки зависимости
     //Итак, первый этап - выбор параметров
     //здесь rPart равно 0 (порядковый номер запроса начинается
     // с 0.
     //Если это первый запрос, и выбранный объект = прокладка
ГОСТ 15180,
     //то сохраняем параметры этого объекта (прокладки) под
номером 0
     if (rPart == 0 && obj.strTheName == "GOST 15180-86"){
          setWorkId(0, obj.objectID);
          Handled = OBJ_HANDLED;
     };
     //Для второго запроса - проверяем дополнительно является ли
этот
     //объект фланцем ГОСТ 12821, и, если да, то сохранить
параметры этого
     //фланца под номером 1.
     if (rPart == 1 && obj.strTheName == "GOST 12821-80"){
           setWorkId(1, obj.objectID);
          Handled = OBJ_HANDLED;
     };
     //Дальше рассматриваем сохраненные данные объектов:
     //Если что-то сохранилось в первом этапе, то есть, данные
не равны 0
     if (getWorkId(0)!=0)
     {
          //Восстанавливаем данные объекта 0 в переменную obj
          getObjData(getWorkId(0), obj);
          //Находим ближайшую к точке вставки плоскость
          strNearestPlane = GetNearestPlane(pntOrigin);
          //если плоскость была найдена,
          if (strNearestPlane != UnknownValue) {
                //То устанавливаем геометрическую зависимость
вставки.
                SetGeomConstraint(INSERT, CONTRDIRECT, obj, WP1,
strNearestPlane, 0, TRUE);
```

```
                //Не выбирается вектор направления
                NoVectorSelect = 1;
          };
     };
     //если во втором случае данные были сохранены, то:
     if (getWorkId(1)!=0)
     {
          //восстанавливаем их опять в переменную obj
          getObjData(getWorkId(1), obj);
          //Устанавливаем набор параметрических зависимостей
          //По условному давлению, условному диаметру,
          //виду с разрезом, Ряду фланца и ряду исполнений.
          SetParamConstraint(Py, obj, EXPR, "obj.Py", TRUE);
          SetParamConstraint(Dy, obj, EXPR, "obj.Dy", TRUE);
          SetParamConstraint(bSection, obj, EXPR,
"obj.bSection", TRUE);
          SetParamConstraint(R, obj, EXPR, "obj.R", TRUE);
          SetParamConstraint(isp_row, obj, EXPR, "obj.isp_row",
TRUE);
          NoVectorSelect = 1;
     };
     //Этот блок реализуется при присоединении фланца к трубе
     //для первого запроса выполняется проверка типа и подтипа
     if (rPart == 0 && obj.strTheType == "Tube" &&
obj.strTheSubType == "Round"){
          //Находится ближайшая плоскость
          strNearestPlane = GetNearestPlane(pntOrigin);
          if (strNearestPlane != UnknownValue) {
                //устанавливаются геометрические
                SetGeomConstraint(INSERT, CONTRDIRECT, obj, WP2,
strNearestPlane, 0, TRUE);

SetGeomConstraint(DIRECITON,CODIRECT,obj,WPnormal,obj.WPnormal,0
,TRUE);
                NoVectorSelect = 1;
          };
          //и параметрические зависимости
          SetParamConstraint(Dn,obj,EXPR,"obj.B",TRUE);
          Handled=OBJ_HANDLED;
     };
 };
```

Двутавры ГОСТ 19425





Исходная графика

Скрипт двутавра:

```
SVersion = 2;
 ObjectDescription = @GOST_19425_74;
 function ActHeader {
     NPart=1;
     Public(
          //Открытые параметры- определяющие геометрию: длина,
высота
          //ширина полки.
          L, @LENGTH,
          h, @I_SHAPE_HEIGHT,
          b, @FLANGE_WIDTH,
          //Справочные: масса погонного метра и общая масса
          massa, @MASS,
          mas1, @RUNNING_METER_MASS,
          //Стандартное обозначение и серия двутавра
          Designation,@STANDART_DESIGNATION,
          seria,@SERIES,
          //Рабочие плоскости
          WP1, @CROSS_PLANE,
          WP2,@BOTTOM_PLANE,
          WP3,@SIDE_BOTTOM_RIGHT_PLANE,
          WP4,@SIDE_TOP_RIGHT_PLANE,
          WP5,@TOP_PLANE,
          WP6,@SIDE_TOP_LEFT_PLANE,
          WP7,@SIDE_BOTTOM_LEFT_PLANE,
          WP8,@CROSS_PLANE
     );
     Protected( seted, massa, bHid, seria);
     Changeable( L,b,h, WP1, WP2, WP3, WP4, WP5, WP6, WP7,
WP8,Designation);
     OnDlgBeforeSelectParam = 1;
     ShowWhenSelPnt = 1;
     ContourOnLine = 0;
     NotStdBody = 1;
 };
 function OnInitialization {
     LoadInCache( seria, Designation, h, b );
     if(seted == UnknownValue) {
          seted=1;
          rZOrder=100;
          bHid = 0;
          L = 50;
          strDesignName = "Implementation 1";
          SelectInCache( "kFirst", "seria", "~", "C", "h", "~",
100, "b", "~", 10, "Designation", "~", "1" );
     };
 };
 function SetGripPoint {
```

```
     //У двутавра четыре ручки
     NGrip = 4;
     //Ручки расположены в крайних точках модели двутавра
     //В точке вставки
     pntGrip0 = pntOrigin;
     //С противоположного торца
     pntGrip1 = pntOrigin+L*vecDirection;
     //В средней точке верхней полки
     pntGrip2 = pntOrigin+h*vecPlane;
     //В крайней точке нижней полки
     vecNormal=getLocalNormal(vecDirection,vecPlane);
     pntGrip3 = pntOrigin+vecNormal*b/2;
 };
 function OnMoveGripPoint{
     //Первая ручка для переноса двутавра целиком
     if (NMovingGrip == 0) {
          pntOrigin = pntGrip0;
     };
     //Вторая ручка растягивает длину двутавра и определяет
     //вектор направления вставки двутавра
     if(NMovingGrip == 1){
          L = max(vecLen(pntGrip0 - pntGrip1),1);
          vecDirection = pntGrip1 - pntGrip0;
          LoadInCache( seria,  h, b  );
          SelectInCache( "kFirst", "seria", "~", "20", "h", "~",
h, "b", "~", b);
     };
     //Ручка высоты двутавра
     //При ее растягивании выбирается новая высота двутавра,
     //примерно равная относительному расстоянию курсора от
точки
     //вставки детали.
     if(NMovingGrip == 2){
          //Вспомогательная переменная новой высоты двутавра.
          rNew_h = vecLen(pntGrip0 - pntGrip2);
          LoadInCache( seria,  h, b  );
          SelectInCache( "kFirst", "seria", "~", "20", "h", "~",
rNew_h, "b", "~", b);
     };
     //Ручка ширины полки двутавра
     //При ее растягивании выбирается новая ширина полки
двутавра,
     //примерно равная удвоенному относительному расстоянию
     if(NMovingGrip == 3){
          //просто вспомогательная переменная нового расстояния.
          rNew_b = 2*vecLen(pntGrip0 - pntGrip3);
          LoadInCache(   b,h  );
          SelectInCache( "kFirst",  "b", "~", rNew_b,  "h", "~",
h);
     };
 };
 function OnMakeParameters {
     strTheName = "19425-74";
```

```
     strTheType = "Profile";
     strTheSubType = "I-Shape";
     massa = mas1*L/1000;
     //Находим сразу третий вектор в системе координат детали,
     //как векторное произведение
     vecNormal=getLocalNormal(vecDirection,vecPlane);
     //Расставляем плоскости в порядке против часовой стрелки
     //Плоскость по первому торцу
     WP1 = Plane( pntOrigin+vecDirection*(0), vecDirection );
     //Плоскость по второму торцу
     WP8 = Plane( pntOrigin+vecDirection*L,-vecDirection);
     //Плоскость от середины нижней полки.
     WP2 = Plane( pntOrigin-vecPlane*(0), -vecPlane );
     //Правые боковые
     WP3 = Plane( pntOrigin+vecNormal*(b/2), vecNormal );
     WP4 = Plane( pntOrigin+vecNormal*(b/2)+vecPlane*h,
vecNormal );
     //Плоскость от середины верхней полки
     WP5 = Plane( pntOrigin+vecPlane*(h), vecPlane );
     //Левые боковые
     WP6 = Plane( pntOrigin-vecNormal*(b/2)+vecPlane*h, -
vecNormal );
     WP7 = Plane( pntOrigin-vecNormal*(b/2), -vecNormal );
 };
 function OnDialog {
     //Для красоты
     strRefDataHeader = "------";
     LoadInCache( seria, Designation, h, b );
     //Вид диалога
     UniDialog(
          //Определяющие параметры серия высота и ширина полки
          DBFLD, seria, h, b,
          //Справочные параметры - обозначение и масса погонного
метра
          DBINF, Designation, mas1,
          //произвольный параметр длины двутавра
          VFLD, L,@I_SHAPE_LENGTH__MM,
          //Оформление блока справочных параметров на странице
          //произвольных свойств
          strRefDataHeader,@REFERENCE_DATA_,
          A, @A__CM2,
          Ix, @IX__CM,
          ix, @IX__CM,
          Wx, @WX__CM3,
          Sx, @SX__CM3,
          Iy, @IY__CM4,
          Wy, @WY__CM3,
          iy, @IY__CM,
          //Галка отображение невидимых линий
          BFLD, bHid, @HIDDEN_LINES,
          //Отображение видов AnyWBK означает генерацию
          //зеркальных видов, если они не переопределены
исполнениями
```

```
          TVIDS,lViewType,"AnyWBK",
          VIEW,"Vids"
     );
 };
 function OnDialogChanged {
     ShowBool (0, @HIDDEN_LINES);
     //Галочка "Невидимые линии" нужна только для тех видов,
     //у которых они есть.
     if (lViewType == VLEFT || lViewType == VRIGHT)
           {ShowBool (1, @HIDDEN_LINES)};
 };
 function OnChangeParameters {
     //Порядок изменения параметров:
     //Произвольным просто присваиваются новые значения
     L = new.L;
     //Для табличных выбирается либо по сочетанию определяющих
параметров
     if(new.b != b || new.h != h){
          LoadInCache(h, b);
          SelectInCache("kFirst", "h", "~", new.h, "b", "~",
new.b);
     //Либо по определяющему полю стандартного обозначения
     }else if (Designation!=new.Designation){
          LoadInCache(Designation);
          SelectInCache("kFirst", "Designation", "~",
new.Designation);
     };
     Handled = OBJ_HANDLED;
     fix=0;
     //Реакция на изменение положения плоскостей.
     //Состоит в пересчете нового положения точки вставки
     //и направления
     //Блоки под условиями IsFixedParam можно сгруппировать по
     //вариантам направления плоскостей.
     //Нормали плоскостей коллинеарны vecDirection
     if( IsFixedParam(WP1) ) {
          fix = fix+1;
          vecXOld=vecDirection;
          vecDirection = Vector(new.WP1);
          pntOrigin = Point(new.WP1) - vecDirection*(0);
          restoreBasis(vecXOld, vecPlane, vecDirection);
     };
     if( IsFixedParam(WP8) ) {
          fix = fix+1;
          vecXOld=vecDirection;
          vecDirection = -Vector(new.WP8);
          pntOrigin = Point(new.WP8) - vecDirection*(L);
          restoreBasis(vecXOld, vecPlane, vecDirection);
     };
     //Нормали плоскостей коллинеарны vecPlane
     if( IsFixedParam(WP2) ) {
          fix = fix+1;
          vecYOld=vecPlane;
```

```
          vecNormal=getLocalNormal(vecDirection,vecYOld);
          vecPlane = -Vector(new.WP2);
          pntOrigin = Point(new.WP2) + vecPlane*(0);
          restoreBasis(vecYOld, vecNormal, vecPlane);
          vecDirection = getLocalNormal(vecPlane,vecNormal);
     };
     if( IsFixedParam(WP5) ) {
          fix = fix+1;
          vecYOld=vecPlane;
          vecNormal=getLocalNormal(vecDirection,vecYOld);
          vecPlane = Vector(new.WP5);
          pntOrigin = Point(new.WP5) - vecPlane*(h);
          restoreBasis(vecYOld, vecNormal, vecPlane);
          vecDirection = getLocalNormal(vecPlane,vecNormal);
     };
     //Нормали плоскостей коллинеарны vecNormal
     if( IsFixedParam(WP3) ) {
          fix = fix+1;
          vecZOld=getLocalNormal(vecDirection,vecPlane);
          vecZNew = Vector(new.WP3);
          vecYNew = getLocalNormal(vecDirection, vecZNew);
          restoreBasis(vecZOld, vecPlane, vecZNew);
          pntOrigin = Point(new.WP3) - vecZNew*(b/2);
          vecDirection = getLocalNormal(vecZOld, vecYNew);
     };
     if( IsFixedParam(WP4) ) {
          fix = fix+1;
          vecZOld=getLocalNormal(vecDirection,vecPlane);
          vecZNew = Vector(new.WP4);
          vecYNew = getLocalNormal(vecDirection, vecZNew);
          restoreBasis(vecZOld, vecPlane, vecZNew);
          pntOrigin = Point(new.WP4) - vecZNew*(b/2)+vecYNew*h;
          vecDirection = getLocalNormal(vecZOld, vecYNew);
     };
     if( IsFixedParam(WP6) ) {
          fix = fix+1;
          vecZOld=getLocalNormal(vecDirection,vecPlane);
          vecZNew = -Vector(new.WP6);
          vecYNew = getLocalNormal(vecDirection, vecZNew);
          restoreBasis(vecZOld, vecPlane, vecZNew);
          pntOrigin = Point(new.WP6) + vecZNew*(b/2)+vecYNew*h;
          vecDirection = getLocalNormal(vecZOld, vecYNew);
     };
     if( IsFixedParam(WP7) ) {
          fix = fix+1;
          vecZOld=getLocalNormal(vecDirection,vecPlane);
          vecZNew = -Vector(new.WP7);
          vecYNew = getLocalNormal(vecDirection, vecZNew);
          restoreBasis(vecZOld, vecPlane, vecZNew);
          pntOrigin = Point(new.WP7) + vecZNew*(b/2);
          vecDirection = getLocalNormal(vecZOld, vecYNew);
     };
```

```
     if( fix>1 ) {
          Handled = OBJ_ERROR;
     };
 };
 function BeforeConnect {
     ResetLastConstraint();
 };
 function OnSelectParam {
     //Функция динамического выбора параметров.
     if (lViewType == VTOP || lViewType == VBOTTOM)
          //Не выбирать параметры для видов сбоку
          {BreakAll = 1;}
     else {
          //для остальных видов выбирать длину двутавра по
относительной
          //абсциссе курсора
          L = max(abs(rXcoord),1);
          };
 };
 //Формирование строки спецификации, как суммы подстрок
 strPartName = @I_PROFILE_N_ + Designation + " x " +  L +
@_MASS_+massa+ @_GOST_19425_74;
```

Панели НВ

Скрипт детали:

```
SVersion = 2;
 ObjectDescription = @MCS_STRING6;
 function ActHeader {
     NPart=1;
     Public(
          //наружные параметры- длина и ширина плиты
          L, @LENGTH,
          W, @WIDTH,
          //Расчетная нагрузка и масса плиты
          P, @MCS_STRING9,
          massa, @MASS
     );
     Protected( seted, obozn, H );
     Changeable( );
     OnDlgBeforeSelectParam = 1;
     ShowWhenSelPnt = 1;
     ContourOnLine = 1;
 };
 function OnInitialization {
     LoadInCache( obozn, L, W, H, massa,P);
     if(seted == UnknownValue) {
          seted=1;
          rZOrder=100;
          //Устанавливаем начальные значения параметров и
исполнение
          SelectInCache( "kFirst", "L", "~", 2700, "W", "~",
```

```
1190 );
          strDesignName = "Implementation 1";
     };
 };
 function SetGripPoint {
     NGrip = 2;
     //Две ручки - в точке вставки и с противоположного торца
     //панели
     pntGrip0 = pntOrigin;
     pntGrip1 = pntOrigin + L*vecDirection;
 };
 function OnMoveGripPoint{
     //При редактировании за ручки следующее поведение:
     //перемещать объект целиком при перетаскивании за
     //первую ручку. И растягивать длину и менять вектор
     //направления при растягивании второй ручки
     if(NMovingGrip == 1){
          //Промежуточные переменные для определения геометрии
          rW = W;
          rP = P;
          //Относительная длина
          rL = vecLen(pntGrip0 - pntGrip1)/rScl;
          //Новое направление вектора вставки
          vecDirection = pntGrip1 - pntGrip0;
          LoadInCache(P, W, L);
          //Выбираем новые значения параметров плиты.
          SelectInCache("kFirst", "P", "~", rP, "W", "=", rW,
"L","~",rL);
     } else {
          //При редактировании за ручку 0 переместить точку
          //вставки объекта
          pntOrigin = pntGrip0;
     };
 };
 function OnMakeParameters {
     //Классификаторы плиты
     strTheName = "Panel NV";
     strTheType = "Plita";
     strTheSubType = "Plita";
 };
 function OnDialog {//Вид диалога вставки
     LoadInCache( obozn, L, W, H, massa, P );
     UniDialog(
          //Табличные управляющие поля
          DBFLD, L, W, P,
          //Табличные справочные поля
          DBINF, obozn, H, massa,
          //Виды - фронтальный, сверху, слева
          TVIDS,lViewType,"F", "T", "L",
          VIEW,"Vids");
 };
 function OnDialogChanged {
     ShowDesign(0, "Preview");
```

```
 };
 //изменения параметров
 function OnChangeParameters {
     //Просто при изменении управляющих параметров
     //выбираем из таблицы те, которые соответствуют новым
     //значениям
     if (new.L != L || new.W != W)
     {
          LoadInCache(L, W);
          SelectInCache("kFirst", "L", "~", new.L, "W", "~",
new.W);
     }
     Handled = OBJ_HANDLED;
 };
 function OnSelectParam{
     //функция динамического выбора параметров
     rW = W;
          //для вида VTOP не выбирать параметры динамически
     if(lViewType == VTOP){
          BreakAll = 1;
     }else{
          //Для остальных видов
          //выбирается значение переменной rL - как
          //абсцисса относительного положения курсора.
          rL=abs(vecCoord:x);
          //из таблицы выбирается значения с той же шириной,
          //но с новым значением длины плиты.
          SelectInCashe("kFirst","W", "=", rW, "L", "~", rL);
     };
 };
 function BeforeConnect {
     ResetLastConstraint();
 }
 function OnConnect
 {
 //Автоматическая установка зависимостей
 //Устанавливаются только параметрические зависимости.
 //между плитами по длине и ширине.
     if(rPart == 0)
     {
          if(obj.strTheType == "Plita")
          {
                SetParamConstraint(L, obj, EXPR, "obj.L");
                SetParamConstraint(W, obj, EXPR, "obj.W");
                NoVectorSelect = 1;
                seted = 1;
          };
     };
 }
 //Название плиты формируется на основании табличного поля
 //obozn
 strPartName = obozn;
```