UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA

Departamento de Informática de Sistemas y Computadores

Universitat Politècnica de València

Desarrollo de un sistema software de etiquetado para aplicaciones GD&T (Geometric Dimensioning and Tolerancing)

Trabajo Fin de Máster

Máster Universitario en Automática e Informática Industrial

Autor: Juan Vañó Cerdá

Tutor: Juan Carlos Pérez Cortés

Cotutor: José Luis Guardiola García

2016-2017

Resumen

El proyecto se orienta al desarrollo de un módulo software para la anotación de tolerancias y dimensionado geométrico (GD&T) en planos técnicos 2D y 3D. Las principales aportaciones de este software son:

- Permite añadir la información de GD&T al propio diseño, vinculándose así las especificaciones de diseño, fabricación y calidad.
- Implementa el estándar ISO16792 tanto para planos 2D como para piezas 3D.
- Incorpora una interfaz gráfica homogénea e integrada con las herramientas de diseño técnico y 3D.
- No existe precedente desarrollado como software libre.

La implementación del software se ha realizado como un módulo de la aplicación de software libre de modelado paramétrico FreeCAD (http://freecadweb.org). Se trata de un proyecto multiplataforma puesto que el desarrollo del módulo está hecho con Python y FreeCAD tiene compilaciones para múltiples Sistemas Operativos.

Palabras clave: GD&T, software libre, FreeCAD, multiplataforma, Python, tolerancia.

Abstract

The project is aimed at the development of a labeling software module for Geometric Dimensioning and Tolerancing (GD&T) in 2D and 3D technical drawings. The main contributions of this software are:

- Allows the GD&T information to be added to the design itself, thus linking design, manufacturing and quality specifications.
- Implements the ISO16792 standard for both 2D and 3D parts.
- Incorporates a homogeneous graphical interface and integrated with the technical design tools and 3D.
- There is no precedent developed as free software.

The implementation of the software is done as a module of the parametric modeling free software application FreeCAD (http://freecadweb.org). This is a multiplatform project since the development of the module is done with Python and FreeCAD has compilations for multiple Operating Systems.

Keywords: GD&T, geometric dimensioning, geometric tolerancing, open software, FreeCAD, multiplatform, Python.

Índice de contenidos

Índice de ilustraciones

Índice de tablas

1 Introducción

1.1 Objetivos

El objetivo principal de este proyecto consiste en desarrollar un módulo software que permita la anotación de tolerancias y dimensionado geométrico (GD&T, por sus siglas en inglés) en planos técnicos 2D y 3D.

Dicho módulo deberá cumplir las siguientes especificaciones:

- Permitir añadir la información de GD&T al propio diseño, vinculándose así las especificaciones de diseño, fabricación y calidad.
- Cumplir con el estándar ISO16792 tanto para planos 2D como para piezas 3D.
- Implementar una interfaz homogénea e integrada con las herramientas de diseño técnico y 3D.
- Desarrollar como proyecto de software libre.

1.2 Antecedentes

Si nos ponemos a pensar en aplicaciones de modelado 3D posiblemente la primera aplicación que nos venga a la cabeza sea la de la compañía Autodesk, AutoCAD. No obstante, dicha aplicación no es de las más importantes a la hora de resolver el etiquetado GD&T. Sí es cierto que poseen un etiquetado en 2D, pero para un etiquetado de piezas 3D deberíamos de irnos a utilizar su otra aplicación Inventor y utilizar un plugin de terceros. Por ello para analizar los antecedentes nos vamos a centrar en el análisis de la aplicación IDA-STEP y GD&T Advisor.

IDA-STEP es una aplicación de la empresa LKSoft que permite realizar el etiquetado de una pieza directamente sobre el modelo 3D de una manera intuitiva en donde los datos que se van añadiendo se almacenan directamente en el formato step para una fácil exportación y que de este modo no se pierda información como podía pasar anteriormente donde la información de GD&T que se reflejaba sobre los modelos 2D debía de ser traducida por una persona de un lugar a otro pudiendo ocasionar perdidas de información.

Esta aplicación, cuya interfaz se puede observar en la Ilustración 1, se ayuda de una serie de planos de anotación que debe definir el usuario y sobre los cuales se anota en 2D toda la información de GD&T referente a los elementos seleccionados.

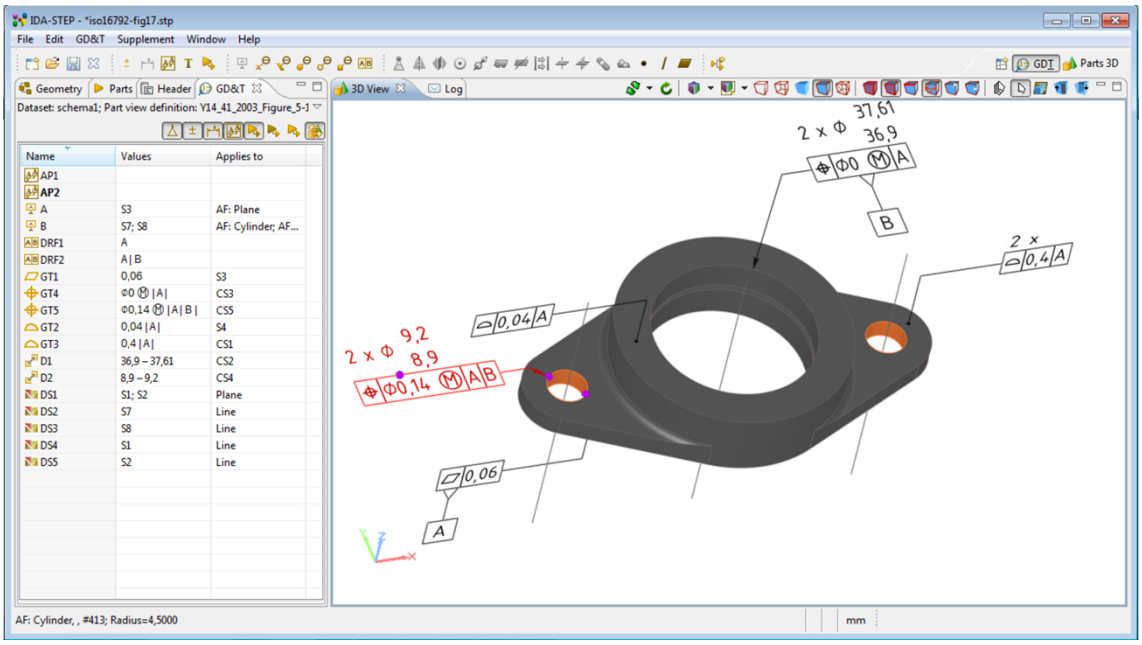

Ilustración 1: Interfaz IDA-STEP

GD&T Advisor es una aplicación de la empresa Sigmetrix que intenta llevar a otro nivel el etiquetado GD&T. Esta aplicación además de resolver el etiquetado sobre piezas 3D, se basta de algoritmos matemáticos propios para determinar que tolerancias se le deben aplicar a cada parte de la pieza simplificando los parámetros a elegir por el usuario, evitando así las discusiones sobre la manera correcta de etiquetar y con ello dicen reducir en cuatro veces el tiempo de etiquetado medio de una pieza.

Además, como se puede observar en la parte inferior izquierda de la Ilustración 2, la propia aplicación pinta la propia figura con un sistema de colores que determina si las diferentes partes de la pieza están completamente anotadas o por el contrario se precisa de alguna tolerancia más por definir, recomendando siempre el sistema las tolerancias que considera que son necesarias y ofreciendo una ayuda al usuario por si le fuera útil.

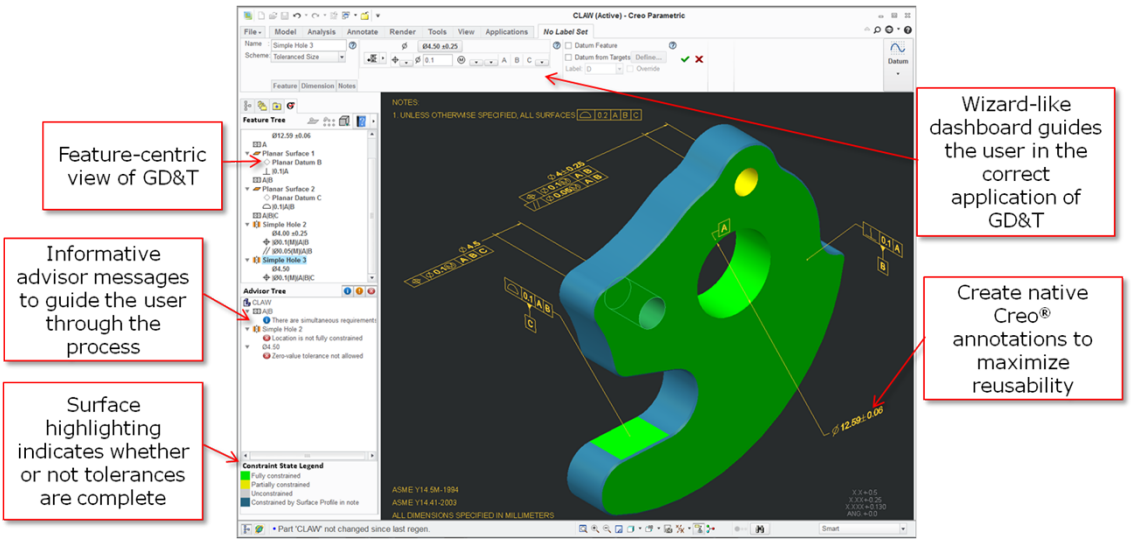

Ilustración 2: Interfaz GD&T Advisor

1.3 Estado del arte

Las aplicaciones mostradas en el apartado anterior se muestran en un estado avanzado de desarrollo. No obstante, al ser de software privativo, no se tiene acceso a su código y por tanto no se puede empezar a trabajar sobre ellas.

Por lo que respecta al software libre, se ha estado buscando información al respecto y lo único que se ha encontrado es una pequeña referencia en la web de FreeCAD (https://www.freecadweb.org/wiki/Drawing_Module#General_Dimensioning_and_Tolerancing) en donde se muestra un link a un proyecto de GitHub pero dónde se nos advierte que en cualquier momento puede romperse dicho enlace. De hecho, la primera vez que se intentó acceder a él ya estaba roto.

Por todo ello, este trabajo se va a considerar como un proyecto nuevo empezado desde cero con el cual se desea contribuir al desarrollo de una parte por ahora inexistente en el software libre y que es de gran utilidad en el mundo de la industria.

1.4 Estructura del trabajo

Este trabajo se encuentra estructurado en los siguientes capítulos:

1. En el primer capítulo de la memoria se pretende introducir el marco en el que se desarrolla el proyecto a la vez que se da una idea general del problema a resolver.

2. Aquí se presentan los conceptos teóricos necesarios para entender que es GD&T y se expone la aplicación FreeCAD para la cual se desarrollará un módulo que resuelva el etiquetado GD&T.

3. El tercer capítulo trata de una manera extensa todo el proceso seguido para el desarrollo del módulo GD&T: la creación de su interfaz, la estructuración de los datos que nos son de interés, el proceso de representación de los mismos en un plano 2D sobre el modelo 3D y la creación de preferencias en la aplicación que mejoren la experiencia de usuario.

4. Este capítulo pretende exponer las conclusiones obtenidas a lo largo de la realización del trabajo, así cómo exponer trabajos futuros que sirvan para mejorar y ampliar las prestaciones de nuestro proyecto.

5. Aquí se muestra la bibliografía utilizada para la consecución de este trabajo.

6. En este capítulo se adjuntarán diversos anexos correspondientes al código desarrollado para la consecución del proyecto.

2 Introducción teórica

2.1 Dimensionamiento geométrico y tolerancia (GD&T)

2.1.1 Descripción general

El dimensionamiento geométrico y tolerancia (GD&T) es el lenguaje usado en dibujos de ingeniería mecánica para definir y comunicar tolerancias de fabricación. Este sistema usa una serie de símbolos que son usados para comunicar de una manera eficiente los requerimientos mecánicos asociados a una pieza o elemento indicándonos su variación permitida o tolerancia.

La principal finalidad que se persigue con el etiquetado GD&T es el poder llevar a cabo un proceso de control de calidad que compruebe que se cumplan las tolerancias de fabricación establecidas mediante el etiquetado GD&T de nuestra pieza. Este proceso en sus orígenes se ha realizado mediante palpadores, pero la tendencia actual es utilizar la Visión Artificial para realizar un escaneado 3D de la pieza que estamos analizando y comprobar que los datos extraídos cumplan las tolerancias establecidas en el modelo etiquetado con GD&T. De modo que si no se pasase el test se pudiera rechazar o modificar la pieza.

GD&T es, y ha sido, utilizado de manera satisfactoria a lo largo de los últimos años en diferentes sectores como la automoción, aeroespacial, electrónica, diseño y las industrias manufactureras. En el mundo moderno actual utilizar una comunicación precisa y efectiva es esencial para llegar a obtener un producto final que se ajuste a la perfección a nuestro diseño. Por ello, se debe utilizar un estándar que no lleve a posibles malinterpretaciones de las representaciones. En este proyecto se va a utilizar el estándar ISO19792 el cual es un derivado del ASME Y14.41-2003 y que además cuenta con el permiso de la ASME (Asociación Americana de Ingenieros Mecánicos).

Algunas ventajas de usar GD&T son:

- Proporciona una manera clara de representar los datums de referencia para luego poder hacer referencia a los mismos de manera sencilla. Un datum de referencia es una parte del modelo a la cual se la identifica con un símbolo para posteriormente referenciarla en una tolerancia.

- Reduce de manera drástica la necesidad de realizar notas en los dibujos para describir requisitos complejos de geometría en una pieza o componente mediante el uso de una simbología estándar que define de forma precisa y rápida los requisitos de diseño, fabricación e inspección.

- Cuando se usan de manera correcta algunos modificadores como el de MMC (condición de máximo material) se facilita y se simplifica el diseño ahorrando costes derivados de accesorios de fabricación y plantillas.

2.1.2 Representación gráfica

Como se ha dicho en el apartado anterior, GD&T es un lenguaje de símbolos utilizado para comunicar y describir los requerimientos mecánicos asociados a una pieza o elemento indicándonos su variación permitida o tolerancia. A continuación, se van a exponer los diferentes símbolos que se utilizan.

En la Ilustración 3 se puede observar cómo se representaría una referencia datum que serviría para identificar alguna parte de la pieza que se está etiquetando para así luego poder hacer referencia a ella en las tolerancias geométricas.

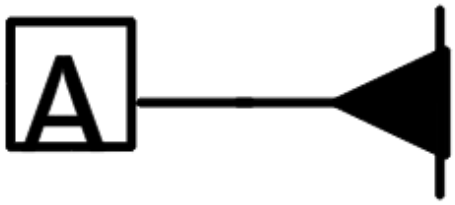

Ilustración 3: Símbolo de referencia datum

En la Tabla 1 se muestran todos los símbolos establecidos en el estándar respecto a las tolerancias geométricas y además se han agrupado por su función (individual o relacionado, dependiendo si se precisa o no de una referencia datum sobre la que se aplica la tolerancia) y por su tipo (forma, perfil, orientación, posición y oscilación).

Tabla 1: Símbolos de tolerancias geométricas

Tipo de función	Tipo de tolerancia	Característica geométrica	Símbolo
Individual (sin referencia a datum)	Forma	Rectitud	
		Planicidad	
		Circularidad	
		Cilindricidad	
Individual o Relacionado	Perfil	Perfil de una línea	
		Perfil de una superficie	

Relacionado (se requiere referencia a datum)	Orientación	Perpendicularidad	
		Angularidad	
		Paralelismo	
	Posición	Simetría	
		Posición	
		Concentricidad	
	Oscilación	Oscilación circular	
		Oscilación total	

En la Tabla 2 se muestran todos los modificadores que define el estándar, aunque cabe destacar, que los más utilizados son el de condición de mínimo material y condición de máximo material. Siendo el resto de modificadores poco empleados pero que en ocasiones resuelven algunos casos que anteriormente se debían especificar mediante notas sobre el dibujo que podían no ser interpretadas correctamente.

Tabla 2: Símbolos de modificadores

Símbolo	Modificador
Ⓕ	Estado libre
Ⓛ	Condición de mínimo material (LMC)
Ⓜ	Condición de máximo material (MMC)
Ⓟ	Zona de tolerancia proyectada
Ⓢ	Indiferencia dimensional (RFS)
Ⓣ	Plano tangente
Ⓤ	Unilateral

Finalmente, en la Ilustración 4, se puede observar lo que se conocería como marco de control. Aquí es donde se colocan de manera ordenada todos los símbolos y datos para definir correctamente una tolerancia. En primer lugar, iría el símbolo correspondiente a la característica geométrica. En segundo lugar, se muestra el valor de la tolerancia, precedido del símbolo de diámetro si es que la tolerancia se refiere a un elemento circular, y seguido del modificador si es que se utiliza. Por último, irían las referencias datum sobre las que se aplica la tolerancia.

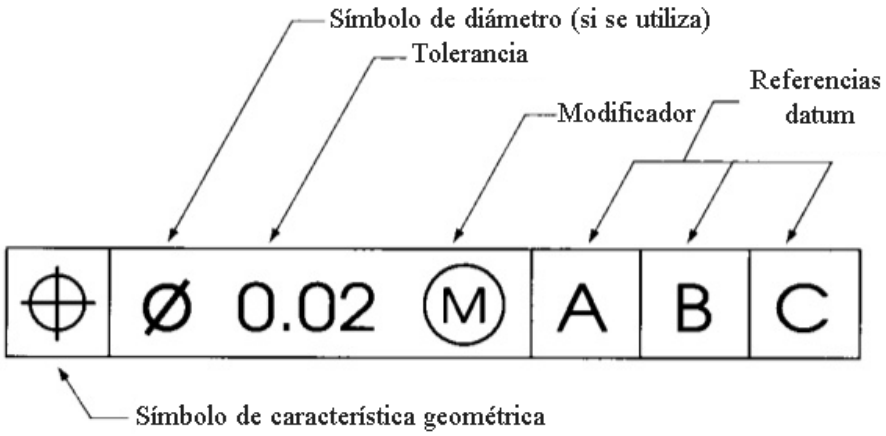

Ilustración 4: Marco de control

Analizando el ejemplo de la Ilustración 4 nos transmite la información de que el cilindro al que apunta dicho marco de control tendría una tolerancia de posición respecto a los elementos definidos previamente como A, B y C de 0.02 como máximo permitido.

2.2 FreeCAD

2.2.1 Descripción general

FreeCAD es una aplicación que es capaz de realizar diversas operaciones de modelado 3D paramétrico, orientada al diseño de productos e ingeniería mecánica, pero que posee un gran abanico de posibilidades para todo el mundo de la ingeniería. El ser una aplicación de modelado paramétrico implica que se puede modificar fácilmente el diseño elaborado modificando los parámetros del propio modelo.

Se trata de un programa de software libre bajo la licencia LGPLv2+, altamente personalizable, scriptable y extensible. Algunas de sus principales características son:

- Posee un completo núcleo de geometría basado en Open CASCADE Technology que permite complejas operaciones 3D y crear todo tipo de formas.

- Todos los objetos de FreeCAD son paramétricos de forma nativa. Esto implica que la forma de los objetos al ser basada en propiedades, al modificarlas también se verá recalculada.

- Su estructura es modular de modo que permite plugins (módulos) que agregan funcionalidades a la aplicación principal. Estos módulos pueden ser programados en C++ y Python.

- Posee una gran cantidad de formatos para exportar/importar como STEP, IGES, OBJ, STL, DXF, SVG, STL, DAE, IFC, OFF, NASTRAN, VRML y FCSTD, siendo este último el formato nativo de FreeCAD.

- Se puede ejecutar como una aplicación de línea de comandos para ahorrar memoria. En esta ejecución no disponemos de una interfaz gráfica pero sí contamos con todas sus herramientas de geometría.

- La propia aplicación dispone de una consola de Python en la cual se pueden lanzar comandos o incluso scripts enteros.

- Su interfaz gráfica está basada en el entorno Qt y cuenta con un visualizador 3D basado en Open Inventor, que aporta un rápido renderizado de escenas 3D y representación gráfica de escenas.

- Es multiplataforma, estando disponible para Windows, Linux/Unix y Mac OSX. Funcionando y comportándose de la misma manera en todas las plataformas.

En la Ilustración 5 podemos observar la interfaz para Mac OSX. En la primera barra de herramientas se encuentran los diferentes comandos básicos como nuevo, abrir, guardar… y en la parte central se observa cómo se encuentra seleccionado el módulo Part. En la segunda barra de herramientas están los elementos para cambiar de vistas. En la tercera se muestran las diferentes acciones que se pueden hacer con el módulo seleccionado. En la parte central tendríamos a la derecha la representación de nuestra pieza y en la izquierda una vista combinada donde se muestra la lista de los objetos que tenemos y los datos paramétricos de nuestra pieza seleccionada. Finalmente, en la parte inferior de la aplicación disponemos de la consola de Python y una vista de los mensajes que vaya notificando la aplicación.

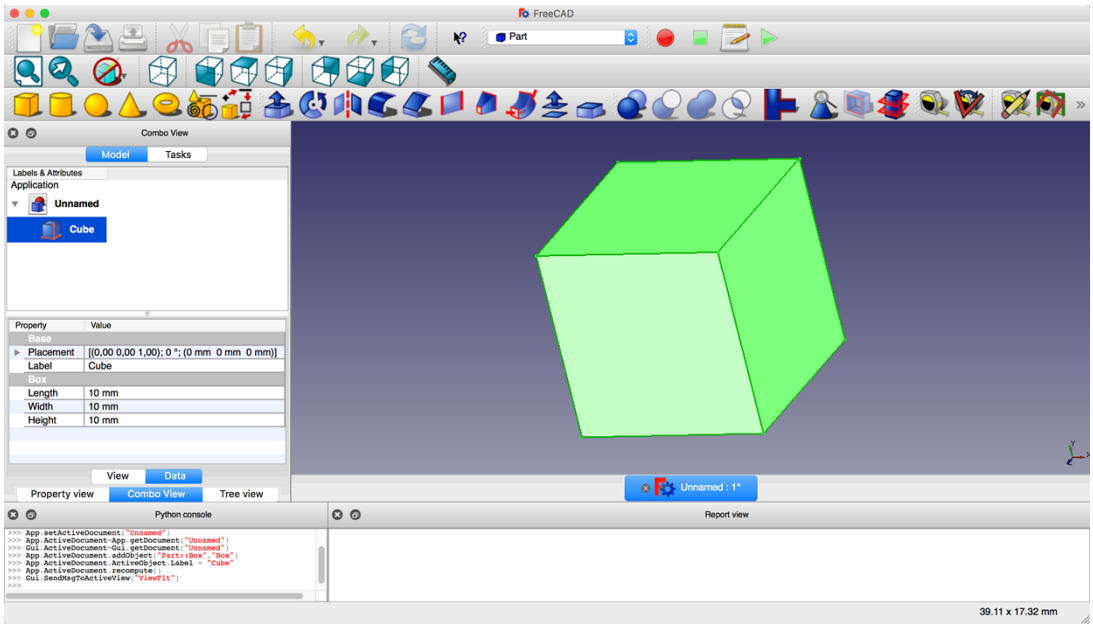

Ilustración 5: Interfaz FreeCAD

2.2.2 Módulos

Los módulos en FreeCAD son una parte muy importante ya que son los que le aportan una gran cantidad de funcionalidades a la misma. Además, estos módulos son programables por cualquier usuario que desee crear uno nuevo o incluso modificar uno existente amoldándolo a sus necesidades o gustos.

Se denomina módulo a cualquier extensión de FreeCAD y lo normal es que estas extensiones vengan acompañadas de su propio entorno otorgándole barras de herramientas y menús para llevar a cabo funcionalidades específicas. Estos módulos pueden estar programados en C++, Python, o en una mezcla de ambos. No obstante, el archivo init de todos los módulos debe de estar en Python para que pueda ser ejecutado por el núcleo del programa.

Para crear un nuevo módulo debemos crear un nuevo directorio dentro de la carpeta Mod que se encuentra alojada en la carpeta de instalación de FreeCAD. En este nuevo directorio se deberá encontrar nuestro fichero InitGui.py junto al resto de ficheros que desarrollemos en nuestro proyecto.

En ese archivo de inicialización se tiene que definir nuestro entorno con como mínimo las siguientes características: el icono que se va a utilizar para dicho módulo, el nombre que va a tener, la información que va a salir cuando pasas el ratón por encima, las acciones que se van a ejecutar cuando se inicia el FreeCAD, las que se van a ejecutar cuando el usuario cambie a tu entorno y las que se van a ejecutar cuando se salga del mismo.

Posteriormente se crearía otro archivo donde se definirían las herramientas de nuestro módulo, conocidas como comandos en FreeCAD. Aquí también habría que

definir el icono de nuestro comando, su nombre y la acción que deseemos que se realice tras pulsar en su botón creado en la barra de herramientas. Además, también se podrá definir en qué condiciones este comando estará activo y demás funcionalidades.

En la Ilustración 6 se puede observar una lista con diferentes módulos para FreeCAD. En esta lista se puede observar nuestro módulo GD&T y otros muy interesantes como por ejemplo el de Robot en el que se pueden simular diferentes robots industriales y el movimiento de sus articulaciones. También hay módulos de arquitectura, simulación de barcos, creación de figuras, mallas, ensamblajes…

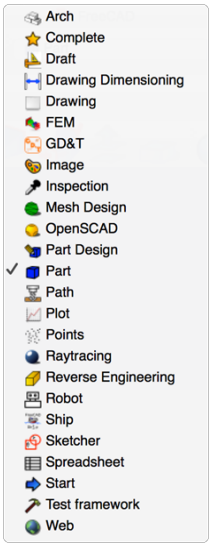

Ilustración 6: Módulos FreeCAD

2.2.3 Instalación

FreeCAD se puede descargar en su versión estable 0.16.67 desde su propia web (https://www.freecadweb.org/wiki/Download) de forma gratuita para Windows, Linux y Mac OS X. Además, para Linux se puede descargar desde el terminal insertando el comando "sudo apt-get install freecad", y para Mac OS X usando el gestor de paquetes homebrew desde el terminal "brew install freecad".

También podemos descargar la versión para desarrolladores 0.17 desde el repositorio de FreeCAD en GitHub (https://github.com/FreeCAD/FreeCAD) y recompilarla. Cabe destacar que esta versión está en constante cambio debido al gran número de desarrolladores que participan simultáneamente.

Una vez descargado e instalado, sobre la carpeta Mod, podremos hacer cualquier cambió de programación en el módulo que deseemos y este se aplicará al reiniciar la aplicación. Del mismo modo, si creamos un nuevo módulo, solo debemos de introducir el módulo entero en esa misma carpeta y la propia aplicación al iniciar detectará el fichero InitGui.py y lo cargará en la aplicación sin necesidad de volver a recompilar el código o volver a instalar la aplicación.

3 Módulo GD&T para FreeCAD

3.1 Introducción

El módulo GD&T para FreeCAD se ha desarrollado íntegramente en Python y se trata de una extensión que nos permite etiquetar las tolerancias geométricas de nuestros modelos o piezas con el lenguaje de símbolos establecido en GD&T cumpliendo el estándar ISO19792.

Este etiquetado se ha querido hacer directamente sobre la figura 3D ya que así se observan de una manera más clara las tolerancias respecto a los planos 2D donde se etiquetaban las diferentes vistas del modelo. Para poder realizar este etiquetado sobre la figura 3D en esta aplicación nos hemos ayudado de unos planos auxiliares, definidos por el usuario, sobre los que se mostrarán los datos del etiquetado. A estos planos los llamaremos planos de anotación.

Sobre una misma cara de la pieza que estemos etiquetando hay dos atributos que le podremos asignar: referencia datum y tolerancia geométrica. No obstante, como hay diferentes casos donde se necesita a la vez aplicarle una referencia datum y una o varias tolerancias geométricas, en esta aplicación se ha agrupado todo en anotaciones para que a la hora de dibujarlo lo dibuje todo en el mismo lugar de nuestro escenario.

Así mismo, las tolerancias geométricas también pueden tener asignadas varias referencias a datum. Estas referencias datum sobre las que se referencia la tolerancia se han definido como un sistema datum, el cual definirá el usuario como un conjunto de referencias datum, y este sistema será el que elegirá el usuario a la hora de definir la tolerancia geométrica.

3.2 Interfaz gráfica

Como ya se ha dicho anteriormente, el etiquetado GD&T trata de comunicar de una manera clara, mediante símbolos, las tolerancias geométricas de una pieza. Por ello, lo primero que se ha hecho en cuanto a la interfaz de este módulo ha sido el desarrollo de dichos símbolos.

En la Tabla 3 podemos observar los iconos desarrollados para representar las diferentes características geométricas y en la Tabla 4 los desarrollados para representar los modificadores. Todos estos iconos han sido creados mediante la aplicación de software libre inkscape y son de una tonalidad naranja la cual será la base de nuestro módulo a la hora de representar nuestras anotaciones y los diferentes comandos de la barra de herramientas para darle una estructura homogénea a nuestro módulo.

Tabla 3: Iconos de características geométricas en el módulo GD&T

Icono	Característica geométrica
	Rectitud
	Planicidad
	Circularidad
	Cilindricidad
	Perfil de una línea
	Perfil de una superficie
	Perpendicularidad
	Angularidad
	Paralelismo
	Simetría
	Posición
	Concentricidad
	Oscilación circular
	Oscilación total

Tabla 4: Iconos de modificadores en el módulo GD&T

Icono	Modificador
	Estado libre
	Condición de mínimo material (LMC)
	Condición de máximo material (MMC)
	Zona de tolerancia proyectada
	Indiferencia dimensional (RFS)

	Plano tangente
	Unilateral

En la Tabla 5 se observan el resto de iconos que se han desarrollado para nuestra aplicación. El primero se trata del icono que va a identificar nuestro módulo. Los cinco siguientes son los iconos utilizados para representan los diferentes comandos disponibles en nuestra barra de herramientas: añadir plano de anotación, añadir referencia datum, añadir sistema datum, añadir tolerancia geométrica e inventario de los elementos de GD&T. De estos, los tres primeros también se utilizan como identificadores de los objetos creados, y en el caso de las tolerancias geométricas se usará el propio icono de la característica geométrica que esté definiendo el usuario. Por último, tendríamos el símbolo que define el diámetro de una pieza y el que define una anotación.

Tabla 5: Iconos generales en el módulo GD&T

Icono	Identificador
	GD&T
	Plano de anotación
	Referencia datum
	Sistema datum
	Tolerancia geométrica
	Inventario
	Diámetro
	Anotación

En la Ilustración 7 se puede observar la interfaz que tiene el módulo GD&T. En la barra de herramientas se pueden observar los diferentes comandos disponibles, a la parte izquierda los diferentes objetos creados para definir nuestras anotaciones y a la derecha el propio etiquetado.

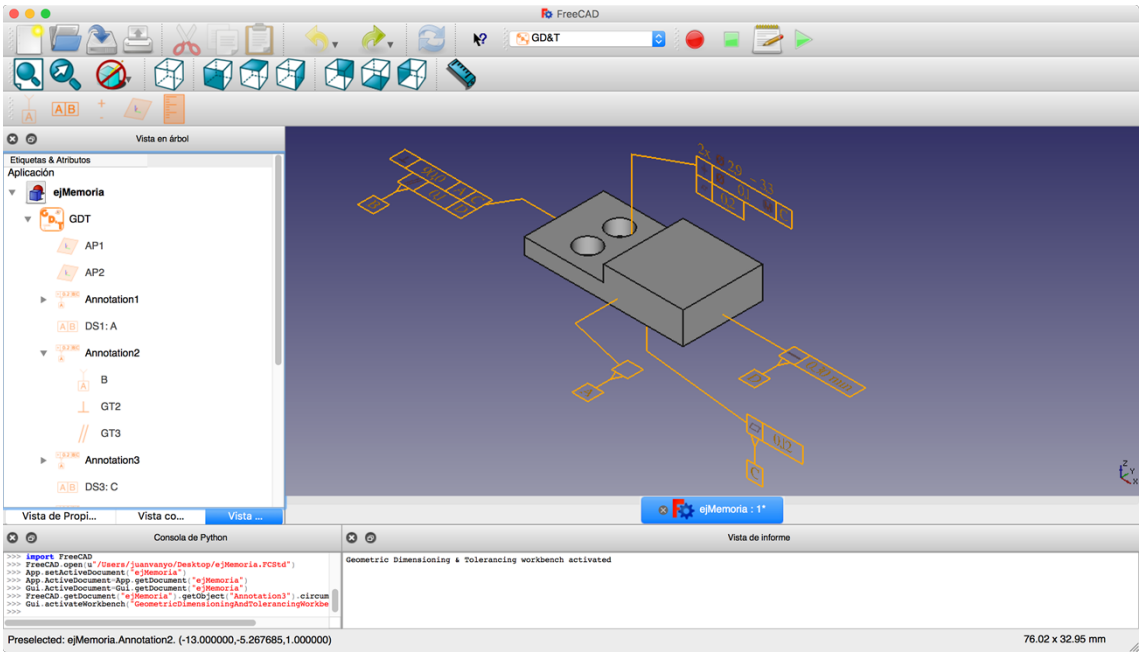

Ilustración 7: Interfaz general del módulo GD&T

Otra parte importante de la interfaz de esta aplicación son los widgets que se muestran al activar un comando de la barra de herramientas. Estos widgets han sido desarrollados con Qt y para llevarlo a cabo de una forma más sencilla y clara se han seguido los siguientes pasos:

En primer lugar, se han creado unas pequeñas clases donde se definen todos los pequeños conjuntos que se van a utilizar en nuestro módulo. Por ejemplo, se han creado pequeñas clases donde se define la creación de una etiqueta y un campo de texto, otra clase con una etiqueta y una comboBox para seleccionar alguna preferencia, y así con todos los elementos que nos hagan falta. Todas estas clases varían dependiendo de los parámetros que le pasemos y cuentan con el método "generateWidget" que será el encargado de devolver un *layout* horizontal con los elementos de la clase.

En segundo lugar, tenemos una clase más genérica a la cual se le pasa una lista con las diferentes clases que tiene que llamar para construir su widget. Esta clase se encargará de llamar a todos los métodos "generateWidget" de las clases que haya en la lista mencionada anteriormente y los colocará en un *layout* vertical. Además, se le añadirá un botón de cerrar y uno de aceptar.

Finalmente hay que llamar a este último método para que se cree el widget correspondiente al comando activado. Para ello, para cada comando, se ha creado un nuevo fichero (con las características que se explicaron en el capítulo 2.2.2) y aquí es donde se añaden los diferentes elementos que se desean crear en una lista con sus parámetros específicos y llamamos al método anterior para que cree el widget entero con los elementos de la lista.

También cabe destacar que todos los textos del módulo se han desarrollado en inglés ya que podríamos considerarlo como un idioma estándar. A continuación, se va a detallar cómo quedarían todos los comandos de nuestro módulo y sus características.

3.2.1 Add Datum Feature

Add datum feature, o añadir referencia datum en español, es el primer comando que se muestra en la interfaz de nuestro módulo. Como se ha comentado anteriormente, a los comandos se les puede añadir una condición de activación la cual deberá de cumplirse para poder activar el mismo. En este caso, para poder añadir una referencia datum se deberán de cumplir las siguientes premisas: deberá de existir como mínimo un plano de anotación definido por el usuario, deberá de estar seleccionada como mínimo una cara para etiquetar y además esta cara, o conjunto de caras, no deberá de tener previamente asignada una referencia datum.

Una vez seleccionado el comando de añadir referencia datum, se nos desplegará un widget en la parte izquierda de nuestra aplicación. Este widget nos permitirá definir diversos parámetros que poseerá la referencia datum que creemos.

En primer lugar, tendremos el nombre que le asignaremos a nuestra referencia datum. Este nombre lo calcula automáticamente nuestro módulo mediante un pequeño diccionario que va desde DF1 hasta DF99, pudiéndolo modificar el usuario.

En segundo lugar, tenemos el valor de la referencia datum. Este valor también se calcularía automáticamente mediante otro diccionario que va desde A hasta Z, pudiéndolo cambiar también el usuario.

A continuación, se nos pide que seleccionemos un plano de anotación, el cual se previsualizará en nuestro escenario mediante una rejilla (véase la Ilustración 8). Esta rejilla o cuadricula es manejada por el módulo Draft de FreCAD.

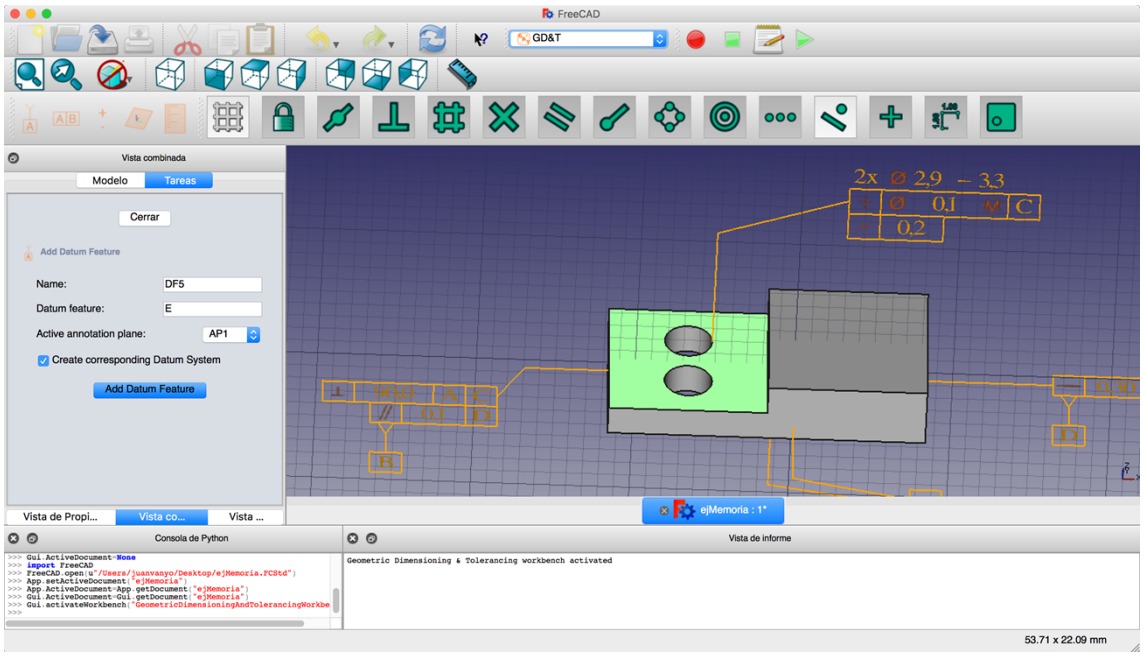

Ilustración 8: Widget Add Datum Feature

Finalmente, el widget también nos da la posibilidad de marcar una casilla con la cual indicamos si queremos que automáticamente se cree un sistema datum que contenga únicamente esa referencia datum para luego poder usarlo en una tolerancia geométrica.

Una vez pulsamos sobre "*Add Datum Feature*", nos aparecerá un nuevo widget, también del módulo Part, con el que podremos seleccionar un punto sobre la cuadrícula de nuestro plano de anotación o insertarlo manualmente (véase la Ilustración 9).

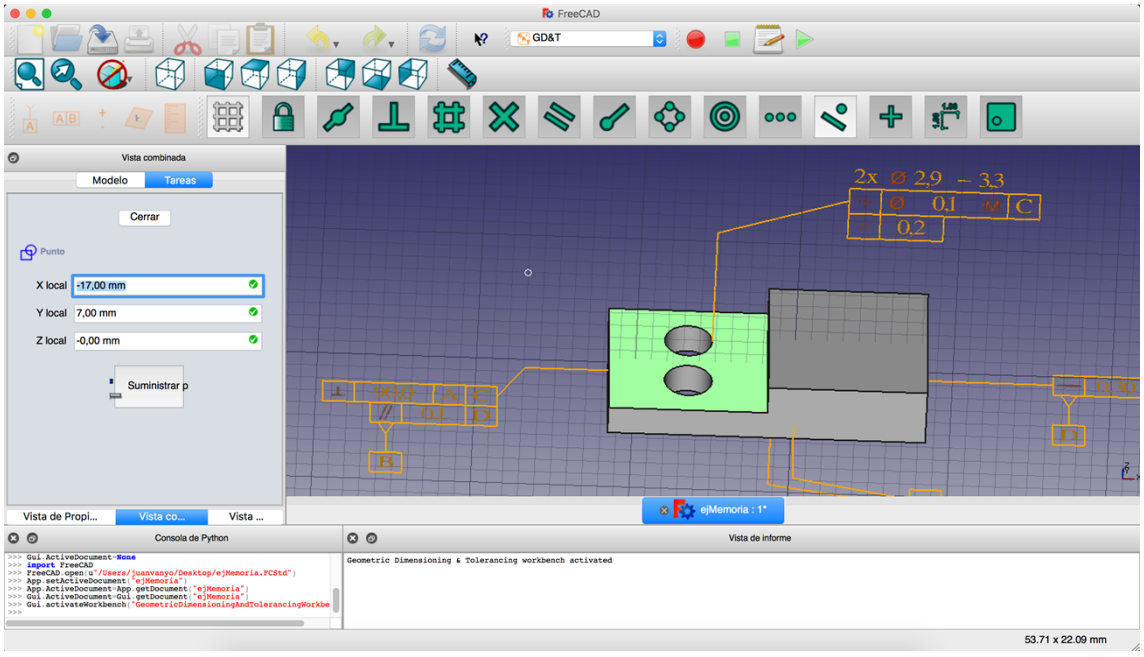

Ilustración 9: Selección del punto de anotación

Cuando se haya seleccionado el punto de anotación, el plano de anotación se esconderá y se pintará sobre ese punto nuestra referencia datum como se muestra en la Ilustración 10.

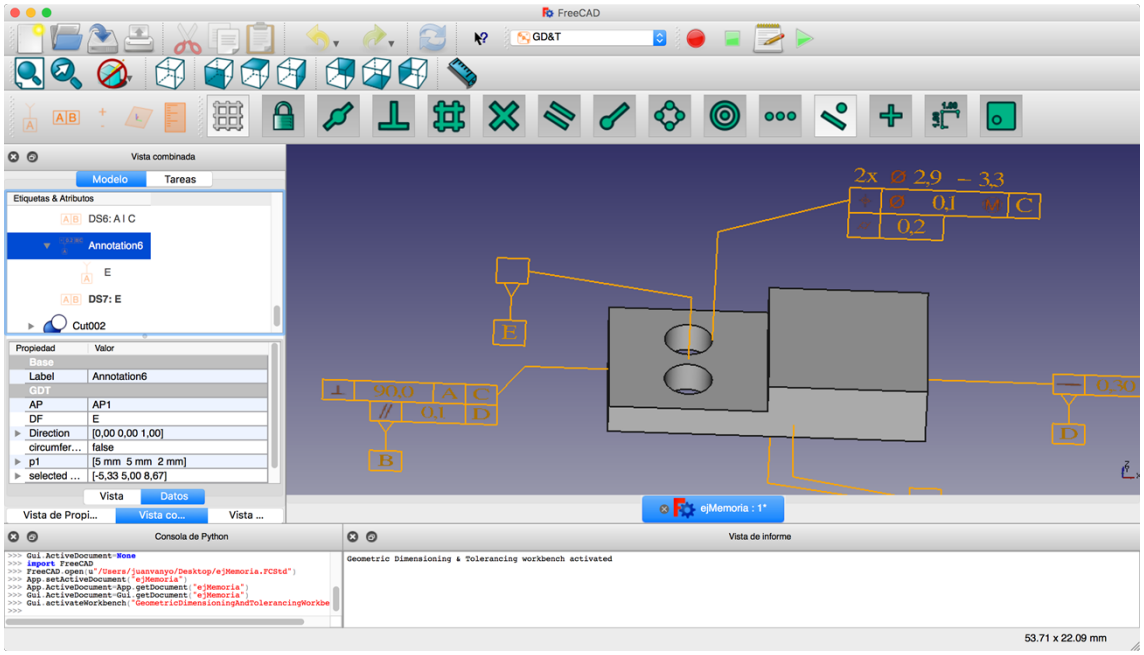

Ilustración 10: Resultado de añadir una referencia datum

Luego, también cabe la posibilidad de que se desee anotar una referencia datum sobre una cara que tenga definida una anotación mediante alguna tolerancia geométrica (véase la primera imagen de la Ilustración 11). En este último caso lo que se hará es añadir a esa anotación la información pertinente de la referencia datum que vamos a crear. Por tanto, ya no nos hará falta ni escoger sobre que plano de anotación se va a pintar ni sobre qué punto, ya que dichos parámetros ya se encontrarán definidos en la anotación. Obteniendo como resultado la Ilustración 12.

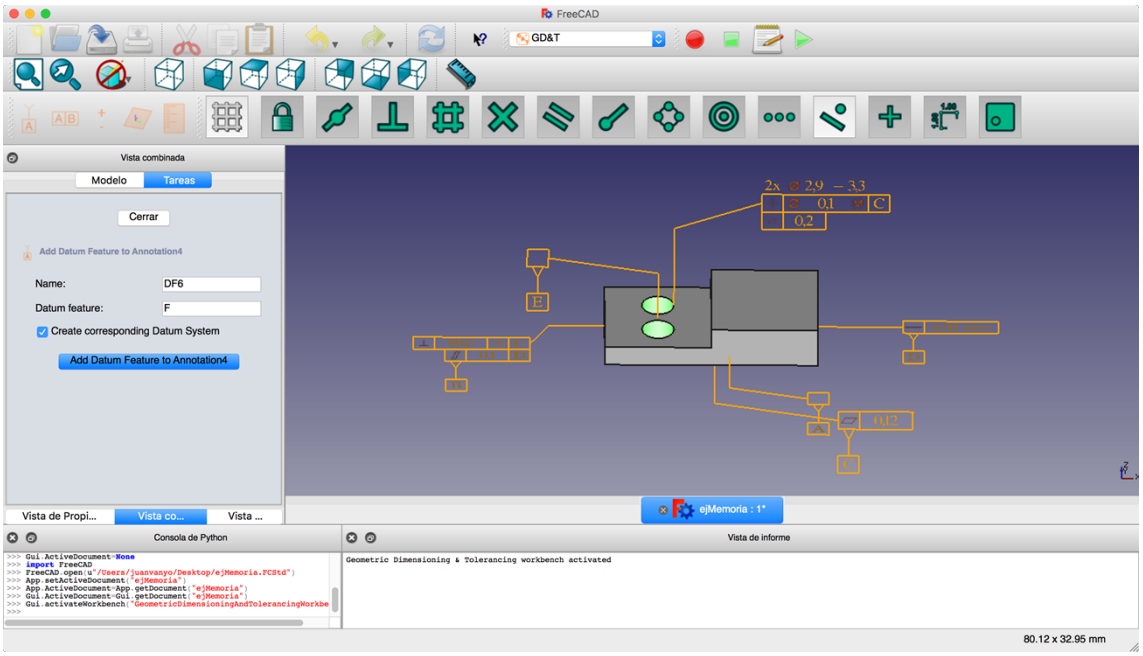

Ilustración 11: Widget Add Datum Feature en una anotación existente

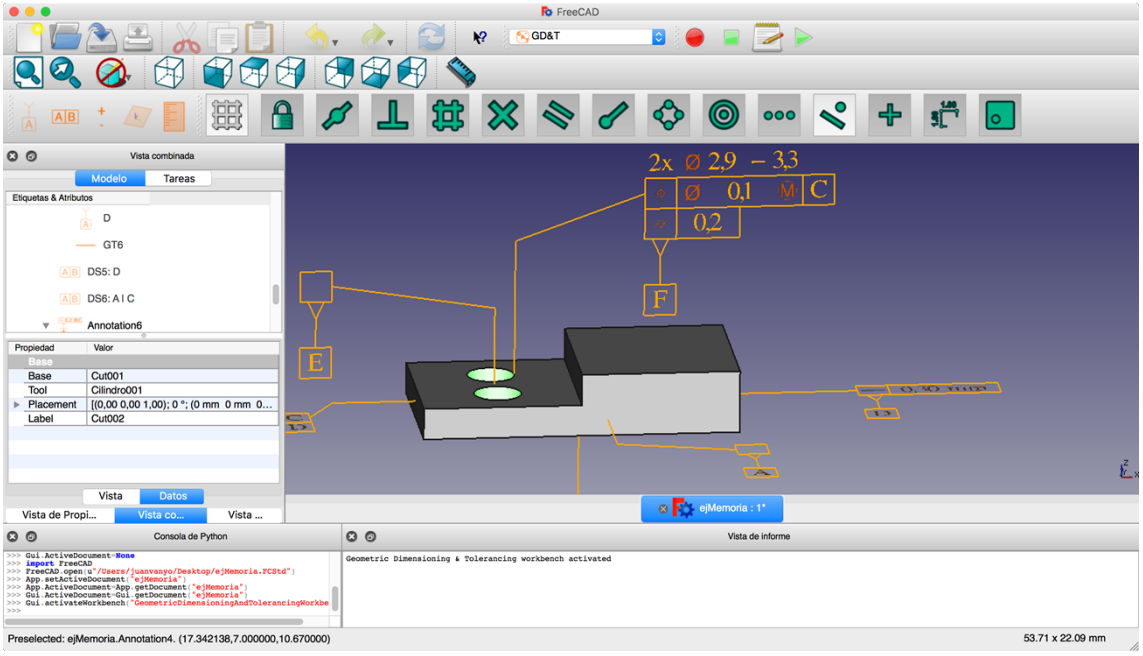

Ilustración 12: Resultado de añadir una referencia datum en una anotación existente

3.2.2 Add Datum System

Add datum system, o añadir sistema datum en español, es el segundo comando que se muestra en la interfaz de nuestro módulo y su condición de activación la cual deberá de cumplirse para poder activar el mismo es únicamente que previamente exista al menos una referencia datum.

En la Ilustración 13 se puede observar el widget creado para añadir un sistema datum. En este widget, al igual que en el anterior, existe un campo para asignarle un nombre al objeto que vamos a crear. Este se nos rellenará automáticamente

mediante un diccionario que va desde DS1 a DS99 pudiéndose modificar la etiqueta por el usuario.

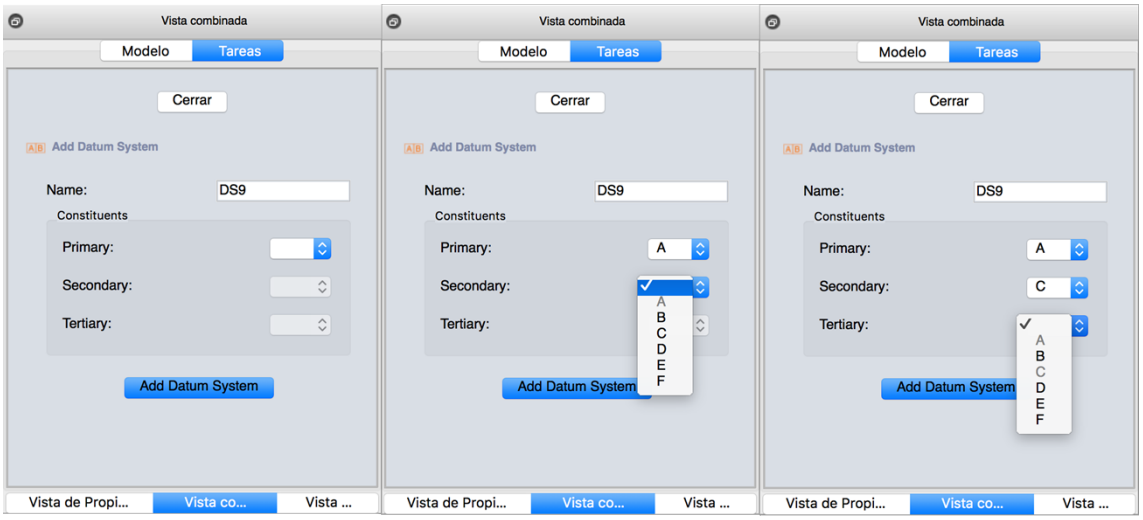

Ilustración 13: Widget Add Datum System

El siguiente parámetro que se nos pide es que seleccionemos los parámetros datum que van a formar nuestro sistema. En estos *comoBox* se nos mostrará como opciones seleccionables todas las referencias datum que hayamos creado previamente. Además, esta sección se ha programado de modo que no puedas seleccionar una referencia datum secundaria si primero no existe una primaria y que no podamos seleccionar una referencia datum terciaria si no existe una secundaria. Del mismo modo, en la lista de ítems seleccionable se deshabilitarán los que ya se hayan seleccionado previamente. También, por ejemplo, si se cambia y se deja en blanco la referencia datum primaria; la secundaria y terciaria pasarán a estar en blanco también. Todo ello para mejorar la experiencia de usuario.

Finalmente destacar que el nombre que se le asignará a estos sistemas será una combinación entre el campo de texto rellenado y las referencias datum seleccionadas siguiendo la siguiente estructura: *Name* + ": " + *Primary* + " | " + *Secondary* + " | " + *Tertiary*. Ejemplo: "DS9: A | C"

3.2.3 Add Geometric Tolerance

Add geometric tolerance, o añadir tolerancia geométrica en español, es el tercer comando que se muestra en la barra de herramientas de nuestro módulo y su condición de activación la cual deberá de cumplirse para poder activar el mismo es que previamente exista al menos un plano de anotación y que el elemento seleccionado sea una o más caras de la pieza, a la cual se le asignará dicha tolerancia.

En la Ilustración 14 se puede observar el widget creado para añadir una tolerancia geométrica y algunos campos de selección desplegados con sus elementos disponibles. En este widget, al igual que en los anteriores, existe un campo (editable por el usuario) para asignarle un nombre al objeto que vamos a

crear. Este se nos rellenará automáticamente mediante un diccionario que va desde GT1 a GT99.

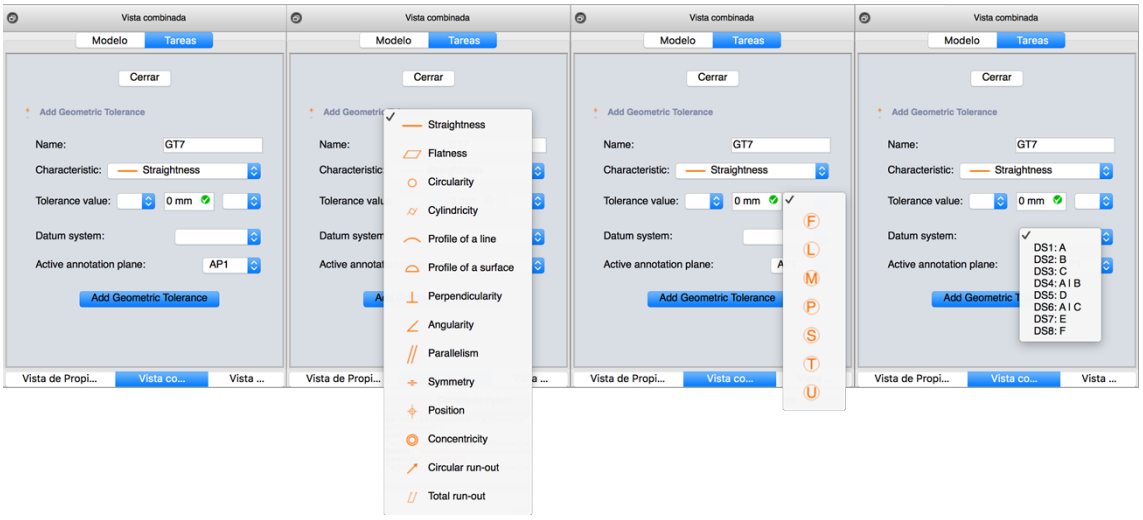

Ilustración 14: Widget Add Geometric Tolerance

El siguiente parámetro que aparece se trata de la selección de la característica geométrica a la que se va a referir la tolerancia que estamos definiendo. El siguiente parámetro sería la definición del valor que tendrá dicha tolerancia. Para ello tenemos varios campos:

- En primer lugar, tenemos un selector para decir si la tolerancia se va a aplicar a un elemento circular o no (se hablará detalladamente de esto más adelante)

- En segundo lugar, se especifica el valor numérico de la tolerancia. Este campo se trata de un campo especial el cual comprueba que el valor introducido sea válido y por defecto pilla las unidades que el propio usuario tiene establecidas en la aplicación FreeCAD.

- Finalmente se seleccionará si se desea utilizar algún modificador y que modificador en sí. Además, al pasar el cursor sobre este campo, aparecerá el nombre del modificar seleccionado para aportarle una ayuda al usuario.

A continuación, si se precisa, se seleccionará el sistema datum sobre el que se referirá la tolerancia entre una lista con todos los sistemas datum creados previamente por el usuario.

Finalmente se seleccionará el plano de anotación sobre el cual se va a pintar la tolerancia en nuestro escenario y posteriormente, al pulsar sobre "*Add Geometric Tolerance*", se nos pedirá que seleccionemos un punto sobre el que se dibujará nuestra anotación. Además, del mismo modo que ocurría con las referencias datum, si se va a añadir la tolerancia geométrica a una cara que ya tiene asignada alguna anotación, no se nos pedirá dicho campo ni el punto de anotación.

Como ya hemos dicho, se puede indicar si la tolerancia geométrica que estamos definiendo se va a aplicar a un elemento circular o no. En caso afirmativo, nuestro widget se verá modificado automáticamente añadiendo algunos campos como se observa en la Ilustración 15.

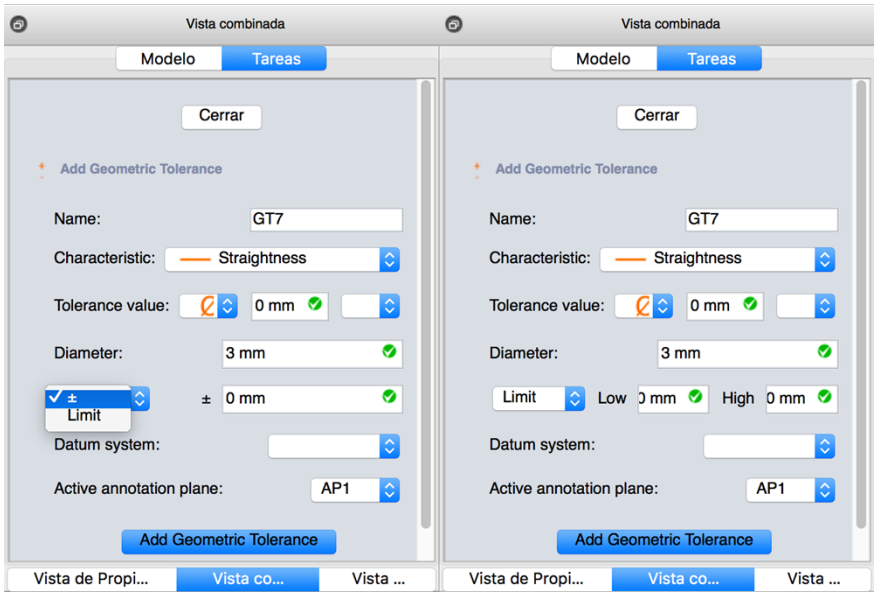

Ilustración 15: Widget Add Geometric Tolerance sobre formas circulares

En este último caso, nuestro módulo intenta extraer de los datos de nuestro modelo el valor del diámetro del elemento que estamos etiquetando y lo asocia al campo reservado para el diámetro. Además, te permite editarlo por si se diera el caso de que no obtuviera el valor correcto.

En esta sección también se definiría si el propio diámetro de la pieza puede tener alguna tolerancia en su dimensión, pudiéndose seleccionar entre una tolerancia de ± o de límite inferior y límite superior, modificando nuevamente el widget dependiendo de la tolerancia seleccionada.

Esta tolerancia del diámetro sería una tolerancia independiente de GD&T pero que nos hace falta definirla para poder definir las tolerancias geométricas que estén relacionadas con elementos circulares como serían las tolerancias de circularidad o cilindricidad. De este modo se utilizará el símbolo del diámetro, como está establecido en el estándar, para identificar dichos casos.

3.2.4 Add Annotation Plane

Add annotation plane, o añadir plano de anotación en español, es el cuarto comando que se muestra en la barra de herramientas de nuestro módulo y su condición de activación la cual deberá de cumplirse para poder activar el mismo es que se haya seleccionado una cara de la figura sobre la que vayamos a trabajar.

En la Ilustración 16 se observa el widget creado para añadir una tolerancia geométrica. Este widget, al igual que en los anteriores, existe un campo (editable por

el usuario) para asignarle un nombre al objeto que vamos a crear. Este se nos rellenará automáticamente mediante un diccionario que va desde AP1 a AP99. Además, contamos con un campo de offset, el cual definirá exactamente donde se encontrará nuestro plano de anotación y cuando este valor sea modificado, también lo hará la cuadrícula auxiliar que se dibuja para representar el plano de anotación.

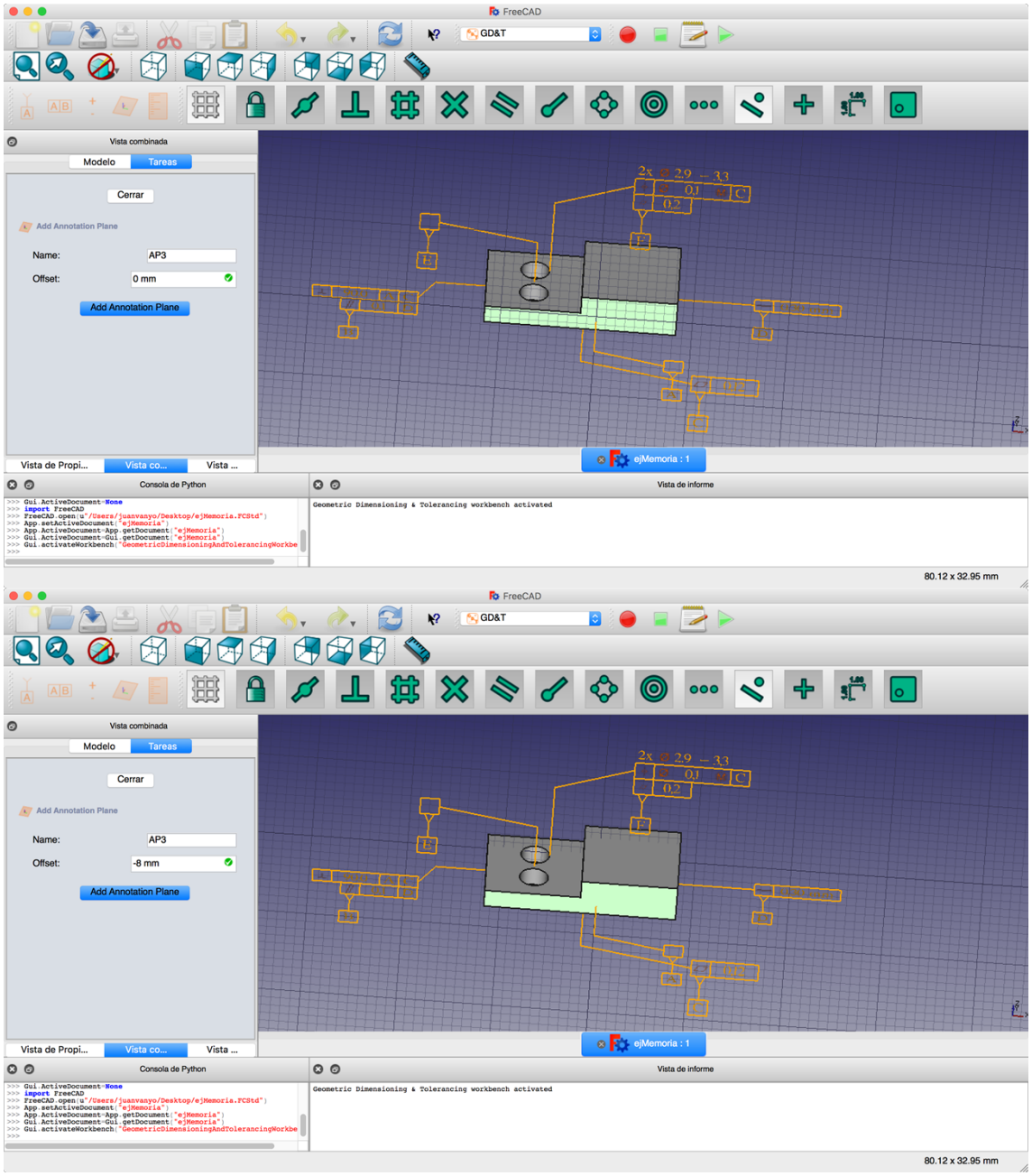

Ilustración 16: Widget Add Annotation Plane

3.2.5 Inventory of the elements of GD&T

Inventory of the elements of GD&T, o inventario de los elementos de GD&T en español, es el quinto y último comando de nuestro módulo y este no posee ninguna condición de activación. Es decir, no hace falta que se cumpla ninguna condición para que el usuario lo puede activar.

Este widget es una recopilación de todos los elementos u objetos creado con el módulo GD&T. Como se observa en la Ilustración 17, estos widgets siguen la misma estructura que los descritos anteriormente para los diferentes comandos, pero con pequeños cambios.

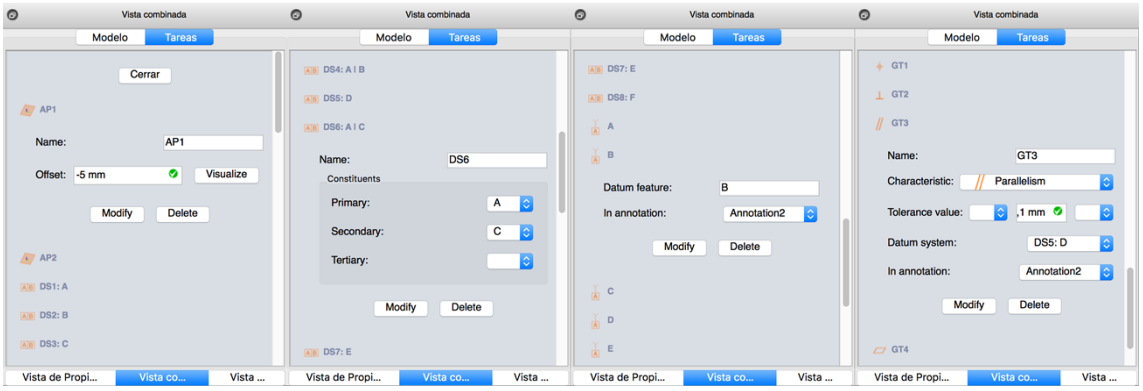

Ilustración 17: Widget Inventory of the elements of GD&T

En este inventario se ordenan todos los elementos de GD&T agrupados en el siguiente orden: planos de anotación, sistemas datum, referencias datum y por ultimo las tolerancias datum. Todos estos elementos son colapsables/expandibles mostrando así todos los posibles parámetros que se pueden modificar o si se desea, eliminar el elemento por completo.

A la hora de realizar los widgets de este inventario, se han tenido que reescribir los métodos originarios para aplicarle las modificaciones deseadas. Para ello, se ha realizado un documento nuevo para cumplir dicho propósito con una estructura similar a la que teníamos para definir el resto de widgets, pero añadiéndole una complejidad extra al tener que gestionar los widget y datos de todos los elementos de nuestro módulo a la vez y de una forma independiente de forma que se modifiquen solo los datos del elemento deseado.

Por lo que respecta a los widgets de los planos de anotación, se les ha añadido un botón con el cual al pulsarlo se muestra directamente en nuestro escenario la representación de dicho plano de anotación. Del mismo modo, si se modifica el valor del desplazamiento también se representará sobre el escenario. Además, si se modificase el valor del desplazamiento y se guardaran los cambios, todas las anotaciones que estén situadas sobre ese plano también realizarían dicho desplazamiento y por tanto no se permite la eliminación de planos de anotación sobre los que exista alguna anotación.

En cuanto a los widgets de los sistemas datum no se ha aplicado ningún cambio significativo. Únicamente se han añadido los botones de modificar y de borrar y se ha mantenido toda la gestión que se llevaba a cabo para deshabilitar los datos que ya se habían usado.

Por lo que respecte a los widgets de referencias datum y de tolerancias geométricas, además de los botones de modificar y eliminar, ambos han sufrido el mismo cambio. Se ha sustituido el selector de plano de anotación por un selector de la anotación en la que se encuentran. De este modo, el usuario puede cambiar una tolerancia o una referencia datum a otra anotación si lo considera oportuno.

También cabe destacar que en este apartado se han implementado algunos mecanismos de control que impiden que el usuario realice algunas acciones incorrectas (véase la Ilustración 18) que podrían ocasionar problemas en la estabilidad de nuestro módulo. De este modo, el propio módulo no permitiría al usuario ni eliminar planos de anotación sobre los que haya alguna anotación realizada ni mover una referencia datum a otra anotación donde ya exista otra.

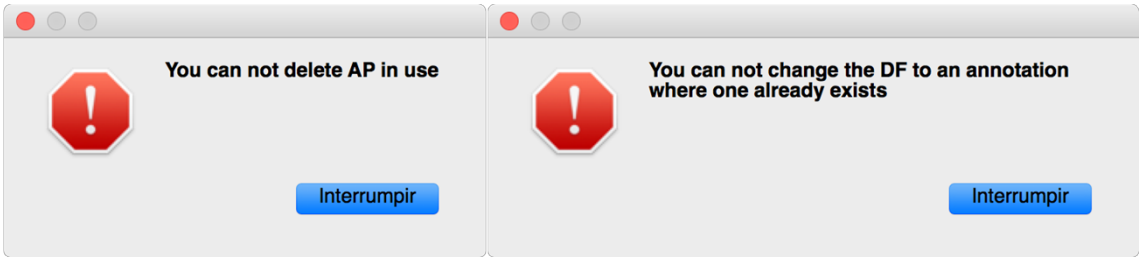

Ilustración 18: Advertencias de uso

3.3 Estructuración de los datos

En un principio, los datos resultantes de la ejecución de los comandos de nuestro módulo se almacenaban en unos buffers que se estructuraban siguiendo un protocolo propio. No obstante, conforme se fue profundizando en los conocimientos del funcionamiento de la aplicación FreeCAD, se descubrió que la propia aplicación no tenía soporte para guardar el estado de las variables tras el cierre de la aplicación.

Para solucionar este problema se investigó en la propia API de FreeCAD y se vio que se podía crea nuestro propio tipo de objetos. De modo que se optó por crear objetos para definir todos los elementos resultantes de nuestro módulo los cuales, además también serían paramétricos simplificando en gran medida el tratamiento de los datos.

A continuación, se van a exponer los dos casos anteriormente introducidos.

3.3.1 Estructuración inicial mediante buffers

En este caso se creó un buffer el cual almacenaba arrays. Donde, además, cada array representaba un elemento de nuestro módulo (plano de anotación, referencia datum, sistema datum o tolerancia geométrica) junto a sus parámetros.

De este modo se identificó a cada elemento con un identificador y dependiendo de este, tendría su propio protocolo con los parámetros que lo iban a definir. En la

Tabla 6 se puede observar la estructura que se seguía en dicha estructuración de los datos.

Tabla 6: Estructura inicial de los datos

Elemento	Identificador	Datos
Referencia datum	1	Id, Name, AnnotationPlane
Sistema datum	2	Id, Name, Primary, Secondary, Tertiary
Tolerancia geométrica	3	Id, Name, Characteristic, ToleranceValue, Modifier, DatumSystem, AnnotationPlane
Plano de anotación	4	Id, Name, Point, Direction, Offset

De este modo, todos los elementos de nuestro modulo se almacenarían en una única variable, cada elemento en una posición del array, de forma que cuando un elemento tiene en sus datos referenciado un plano de anotación, aquí se guardará la posición del array en donde se encuentra dicho plano de anotación para que se pueda acceder a la información del mismo. Del mismo modo ocurre con los datos *Primary*, *Secondary* y *Tertiary*, donde se apuntará a la posición donde se encuentren definidos sus correspondientes referencias datum. Algo similar ocurriría con *DatumSystem*, apuntando a la dirección donde se encuentra definido dicho sistema datum.

Con esta estructuración se conseguía que, si se modificaba por ejemplo una referencia datum, este cambio se viera también reflejado en todos los elementos que lo estuvieran empleando, ya que estos accedían a la posición donde se almacenaba dicho elemento.

Si se navega por el gestor de versiones del repositorio creado en GitHub para este módulo, se puede observar alguna versión con dicha estructura de los datos. Esta estructuración funcionaba correctamente, pero, no obstante, la información almacenada en dicha variable se perdía al cerrar y volver a abrir la aplicación. Por ello se crearon objetos para definir todos los elementos resultantes de nuestro módulo.

3.3.2 Estructuración en objetos de FreeCAD

FreeCAD, además de los tipos estándar de objetos, permite que el usuario cree nuevos objetos hechos en Python llamados Funcionalidades Python (Feature Python). Estos objetos se comportan exactamente como cualquier otro objeto de FreeCAD, y se guardan y restauran automáticamente al guardar o cargar archivos.

Estos objetos son guardados en archivos con la extensión FcStd de FreeCAD con el módulo de Python cPickle. Este módulo transforma los objetos en una cadena de texto y posteriormente al cargar el archivo guardado lee la cadena de texto para recrear el objeto original. De modo que, si se guarda un fichero con objetos personalizados y luego se desea abrir el fichero en otro ordenador, este, deberá de

poseer el código de Python con el que se generó esa cadena de texto para poder recrear el objeto.

Estas Funcionalidades Python se separan en las partes de App y de Gui. En la parte de App se define la geometría de nuestro objeto y en la parte de Gui el modo en que el objeto se dibujará en la pantalla.

Hay varias propiedades que se pueden utilizar para definir nuestro objeto. Las propiedades tienen que ser alguna de las que ofrece la propia aplicación FreeCAD y estás aparecerán en la ventana de vista de propiedades en nuestra aplicación pudiendo ser editadas por el usuario y por tanto son paramétricas. Además, a la hora de programarlo podemos especificar que estas propiedades sean solo de lectura o incluso que sean internas sin poder ser vistas ni editadas por el usuario.

A continuación, se va a mostrar una lista con las diferentes propiedades que se han utilizado en este módulo:

- App::PropertyBool
- App::PropertyFloat
- App::PropertyLength
- App::PropertyInteger
- App::PropertyString
- App::PropertyLink
- App::PropertyLinkSub
- App::PropertyLinkList
- App::PropertyLinkSubList
- App::PropertyVector
- App::PropertyVectorDistance
- App::PropertyColor

Por lo que respecta a las propiedades App::PropertyLinkSub y App::PropertyLinkSubList, son las utilizadas para almacenar un link a las caras con las que se va a trabajar. No obstante, en un primer lugar, dado que no había una documentación clara al respecto de cómo había que definir dichas propiedades, no se encontró una con la que se pudieran almacenar caras de una pieza y se optó por almacenar un hash asignado a la cara, pero se comprobó que, al cambiar los parámetros de dicha cara, este código hash se veía modificado de modo que no nos era de utilidad.

Más tarde se descubrió que la propiedad App::PropertyLinkSub tenía una forma determinada de declararse y que con ella se podían guardar subelementos de una pieza. Para ello, se deben de almacenar dos valores: la pieza genérica que contiene la cara que deseamos almacenar y el nombre de la cara. Con esto se consigue identificar la cara y si esta se ve modificada sigue siendo un link a la misma. App::PropertyLinkSubList se utiliza para almacenar una lista de caras.

Si bien es cierto que son objetos paramétricos, y por tanto se pueden modificar desde la vista de propiedades, se recomienda realizar las modificaciones desde el comando de inventario ya que se aportan facilidades al usuario como por ejemplo sería la elección de la característica geométrica en donde si lo modificamos mediante la vista de propiedades, habrá que escribir la característica geométrica en letra, mientras que si se hace desde el inventario se podrá seleccionar desde una lista. Del mismo modo, si se desea eliminar algún elemento, si lo hacemos directamente desde la lista de elementos, se podría producir algún error o no producirse alguna recomputación necesaria para que se refleje dicho cambio sobre el escenario. Todo esto sería gestionado desde el comando de inventarió para que se realizara con éxito sin ocasionar problemas derivados.

En adición, los objetos que creamos también disponen de diferentes métodos los cuales vamos a comentar a continuación brevemente su funcionalidad separándolos en los métodos de la parte de App y los métodos de la parte de Gui.

Métodos de la parte de App:

- __init__: aquí se añaden las propiedades personalizadas que va a poseer nuestro objeto.
- execute: aquí va el código que se desea ejecutar cuando se realice una recomputación.
- onChanged: aquí se especifica si se desea que alguna propiedad sea de solo lectura o se mantenga oculta. Por defecto son de lectura escritura.

Métodos de la parte de Gui:

- __init__: aquí se añaden las propiedades personalizadas que va a poseer nuestro objeto relacionadas con la parte visual.
- attach: aquí se configuran los elementos que se van a dibujar sobre nuestra escena.
- getDisplayModes: devuelve los modos de representación disponibles.
- setDisplayMode: aquí se definen los diferentes tipos de representación.

- onChanged: aquí se especifican las diferentes acciones a realizar cuando se haya modificado algún parámetro.

- getIcon: aquí se establece el icono que identificará el objeto.

En esta nueva estructuración de los datos en objetos, en primer lugar, se ha creado un objeto GDT el cual será la base para el resto de objetos los cuales heredarán de este y sobrescribirán sus métodos para aportarles la funcionalidad deseada. Posteriormente se han creado cinco nuevos objetos: AnnotationPlane, DatumFeature, DatumSystem, GeometricTolerance y Annotation.

También, cabe destacar, que todos estos objetos se han agrupado en un grupo (GDT) para tener ahí almacenados todos los elementos creados en nuestro módulo y así se diferencie de una manera más clara los propios objetos correspondientes al propio elemento que estamos etiquetando y las anotaciones de GD&T.

A continuación, se van a detallar los nuevos objetos creados y se va a mostrar sus propiedades con una breve explicación además de alguna característica significativa que posean y sea reseñable.

3.3.2.1 AnnotationPlane

Este objeto es el que se encarga de gestionar los planos de anotación de nuestro módulo y para ello utiliza los siguientes parámetros en la parte de App:

- (App::PropertyFloat) Offset: valor del desplazamiento utilizado para definir nuestro plano de anotación.

- (App::PropertyLinkSub) faces: se trata de un link a la cara que se ha seleccionado para realizar el plano de anotación.

- (App::PropertyVectorDistance) p1: representa la posición del centro de masas de la cara seleccionada y será el punto que definirá el plano de anotación, además de ser el punto central de la cuadrícula que representará el plano virtual.

- (App::PropertyVector) Direction: es el vector dirección con el que se formará el plano de anotación junto al punto anteriormente nombrado aplicándole el desplazamiento determinado.

- (App::PropertyVectorDistance) PointWithOffset: se trata del punto anteriormente nombrado con el desplazamiento aplicado. Este parámetro se encontrará en modo solo lectura ante el usuario ya que este valor se calcula internamente ante la modificación de los anteriores.

En este objeto, al realizarse una recomputación del escenario, se reasignará el punto y dirección asignados al plano de anotación de modo que, si se ha modificado la cara asignada a dicho plano, este también se verá afectado.

En cuanto al resto de parámetros, si el usuario decide cambiar el valor del desplazamiento, el punto o la dirección que definen el plano, entonces se recomputará el valor del punto con desplazamiento de modo que se modifique nuestro plano de anotación al nuevo definido por el usuario.

Por último, en este objeto también se ha definido que al hacer doble click en el objeto sobre la lista de objetos de la aplicación, automáticamente se previsualizará nuestro plano de anotación mediante una cuadrícula auxiliar sobre nuestro escenario.

3.3.2.2 DatumFeature

Este objeto es el empleado para las referencias datum y no constaría de ningún parámetro extra más allá del nombre, el cual se encuentra en todos los objetos creados con este módulo.

A la hora de crear una referencia datum se comprueba si existe una anotación previa asignada a la cara seleccionada en ese momento y si es así se asocia dicha referencia datum a la anotación o en caso contrario crear una anotación que posea dicha referencia datum.

3.3.2.3 DatumSystem

Este objeto es el que se utiliza para los sistemas datum y consta de los siguientes parámetros en la parte de App:

- (App::PropertyLink) Primary: se trata de un link a la primera referencia datum del sistema que se va a crear.
- (App::PropertyLink) Secondary: se trata de un link a la segunda referencia datum del sistema que se va a crear.
- (App::PropertyLink) Tertiary: se trata de un link a la tercera referencia datum del sistema que se va a crear.

Este objeto la única peculiaridad que posee, es que al verse modificado alguno de sus parámetros recalcula su propio nombre con la siguiente estructura: *Name* + ": " + *Primary* + " | " + *Secondary* + " | " + *Tertiary*

3.3.2.4 GeometricTolerance

Este objeto se usa para las tolerancias geométricas y posee los siguientes parámetros en la parte de App:

- (App::PropertyString) Characteristic: característica geométrica sobre la que va a ser la tolerancia.
- (App::PropertyString) CharacteristicIcon: icono que representa la característica geométrica. Este parámetro se encontrará oculto ante el usuario y solo se utilizará para cálculos internos.
- (App::PropertyBool) Circumference: booleano que indica si la tolerancia se aplica sobre el diámetro de la pieza que se está etiquetando.
- (App::PropertyFloat) ToleranceValue: valor numérico de la tolerancia.
- (App::PropertyString) FeatureControlFrame: modificador empleado en la tolerancia.
- (App::PropertyString) FeatureControlFrameIcon: icono que representa el modificador empleado. Este parámetro se encontrará oculto ante el usuario y solo se utilizará para cálculos internos.
- (App::PropertyLink) DS: se trata de un link al sistema datum utilizado en esa tolerancia.

Este objeto tiene como característica especial que su icono será en todo momento el especificado por su parámetro *CharacteristicIcon* y que, además, igual que ocurría con las referencias datum, a la hora de crear una tolerancia geométrica se comprueba si existe ya una anotación asignada a la cara seleccionada en ese momento y si es así se asocia dicha tolerancia geométrica a la anotación o en caso contrario crear una anotación que posea dicha tolerancia geométrica.

3.3.2.5 Annotation

Este objeto se usa para las anotaciones que irán dibujadas sobre el modelo 3D y por tanto poseerá, parámetros en la parte de App para almacenar la información de los links a planos de anotación, referencias datum, tolerancias geométricas y demás información, y en la parte de Gui para almacenar información referente a los elementos que se dibujarán sobre el escenario.

Por un lado, los parámetros en la parte de App son los siguientes:

- (App::PropertyLinkSubList) faces: se trata de un link a la cara o caras sobre las que hará referencia nuestra anotación.
- (App::PropertyLink) AP: se trata de un link al plano de anotación sobre el que se dibujará nuestra anotación. De modo que, si se modifica el plano de anotación, la anotación se verá desplazada a dicha posición modificada.

- (App::PropertyLink) DF: se trata de un link a la referencia datum que identifica dicha cara o caras pertenecientes a la anotación.

- (App::PropertyLinkList) GT: se trata de una lista de links a todas las tolerancias geométricas que se le aplican a la cara o caras pertenecientes a la anotación.

- (App::PropertyVectorDistance) p1: este punto indica la posición de la cara donde apuntará la anotación representada en el escenario.

- (App::PropertyVector) Direction: indica el vector normal de nuestra anotación

- (App::PropertyVector) selectedPoint: indica el punto seleccionado por el usuario donde empezará el marco de control de nuestra anotación. De modo que, al ser paramétrico, este punto podrá ser modificado por el usuario en cualquier momento.

- (App::PropertyBool) spBool: se trata de una variable booleana empleada para saber si el usuario ya ha seleccionado un punto para colocar la anotación. De modo que, si no se ha seleccionado se le pedirá al usuario su inserción. Este parámetro se encontrará oculto ante el usuario y solo se utilizará para cálculos internos.

- (App::PropertyBool) circumferenceBool: se trata de una variable booleana que indica si la anotación se realiza sobre una circunferencia.

- (App::PropertyFloat) diameter: representa el valor del diámetro de la pieza. Este valor se calculará automáticamente a partir de los datos del modelo y solo será visible en caso de que el parámetro *circumferenceBool* sea *True*.

- (App::PropertyBool) toleranceSelectBool: se trata de un booleano que hace de selector para determinar si la tolerancia del diámetro se va a aplicar mediante una tolerancia de ± en caso de valer *True* o si se le va a aplicar una tolerancia de límite inferior y límite superior en caso de valer *False*. Y al igual que ocurría con el parámetro anterior, solo será visible en caso de que el parámetro *circumferenceBool* sea *True*.

- (App::PropertyFloat) toleranceDiameter: indica el valor de ± que se permite como tolerancia respecto al diámetro de la pieza seleccionada. Este parámetro solo será visible si el parámetro *toleranceSelectBool* vale *True*.

- (App::PropertyFloat) lowLimit: indica el límite inferior permitido para el valor del diámetro de la pieza seleccionada. Este parámetro solo será visible si el parámetro *toleranceSelectBool* vale *False*.

- (App::PropertyFloat) highLimit: indica el límite superior permitido para el valor del diámetro de la pieza seleccionada. Este parámetro solo será visible si el parámetro *toleranceSelectBool* vale *False*.

Cabe apuntar que el valor de *circumferenceBool* es calculado automáticamente por nuestro módulo de modo que si detecta que en la cara que hemos seleccionado se encuentra un borde cerrado (es decir, un borde que inicia y termina en el mismo punto) y además existen únicamente dos vértices en esa cara (el vértice que definiría el círculo superior y el inferior, véase la Ilustración 19) entonces lo interpretará como una forma cilíndrica. Además, como medida de seguridad, se ha dejado esta variable editable por el usuario por si se diera el caso de que el módulo lo detectara erróneamente.

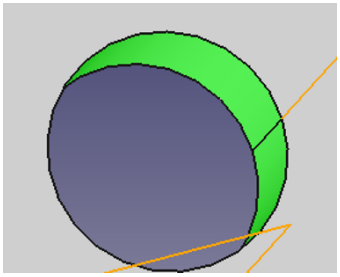

Ilustración 19: Cara con forma cilíndrica

También cabe añadir, que el punto inicial p1 donde apuntará la anotación, dependerá del parámetro anterior. De modo que, si no se trata de una forma cilíndrica, este punto será el correspondiente al centro de masas de la cara seleccionada y en su defecto, se corresponderá al vértice de la parte superior del cilindro.

Todos estos objetos, además poseen un método *make* que será el encargado de crear el propio objeto. Cabe destacar, que en este caso se accede al método *make* a través de los métodos *make* creados para los objetos *DatumFeature* y *GeometricTolerance*. De modo que, al acceder al método *make* de las anotaciones, se comprueba el parámetro *spBool* y en caso de valer *False* se le pide al usuario que seleccione un punto sobre la pantalla, punto sobre el cual se empezará a dibujar el marco de nuestra anotación. En caso contrario simplemente se añade la información nueva y al realizar la recomputación se visualizarán los cambios sobre nuestro escenario.

Por otro lado, los parámetros en la parte de Gui son los siguientes:

- (App::PropertyFloat) LineWidth: indica el valor del ancho (en pixeles) de las líneas que se utilizan para representar la anotación.

- (App::PropertyColor) LineColor: indica el color de las líneas que se utilizan para representar la anotación.
- (App::PropertyFloat) LineScale: se trata de un valor de escalado con el cual se puede modificar el tamaño de las anotaciones al gusto del usuario.
- (App::PropertyLength) FontSize: indica el tamaño de letra que se va a utilizar en las anotaciones.
- (App::PropertyString) FontName: indica la fuente que se va a utilizar en las anotaciones a la hora de representar textos.
- (App::PropertyColor) FontColor: indica el color de que van a ser los textos utilizados en las anotaciones.
- (App::PropertyInteger) Decimals: indica el número de decimales que se van a mostrar en las tolerancias geométricas.
- (App::PropertyBool) ShowUnit: indica si se desea que se muestre el valor de las unidades de las tolerancias, ejemplo: mm.

Aquí también cabe resaltar que se ha definido que al hacer doble click en el objeto que identifica la anotación sobre la lista de objetos de la aplicación, se seleccionarán automáticamente la cara o caras correspondientes a la anotación para así poderlas identificar de una forma sencilla y clara por el usuario.

3.4 Representación gráfica sobre el modelo 3D

FreeCAD usa una gran cantidad de librerías, entre ellas: openCascade para la gestión y construcción de geometrías, Qt para la interfaz de usuario, y Coin3d para la representación de las geometrías y el resto del escenario.

Coin3d es una librería gratuita y de código abierto que utiliza la API de OpenInventor. Esta librería es la utilizada para representar las vistas 3D que aparecen en FreeCAD. Si bien es cierto que openCascade también podría realizar la misma funcionalidad, en los orígenes de FreeCAD se decidió por utilizar Coin3d debido a que aportaba un mayor rendimiento.

Cabe destacar que la librería Coin3d está desarrollada en C++, pero, no obstante, también existe la librería Pivy, la cual es una biblioteca de enlace de Python para Coin3d. Esto, además de permitirnos programar elementos de nuestro escenario en Python, nos permite el dialogar directamente con cualquier escenario 3D con Coin3d mediante la consola de Python de la propia aplicación FreeCAD.

A continuación, se va a detallar como se ha utilizado dicha librería para la representación de las anotaciones de GD&T sobre el escenario de la aplicación. En primer lugar, se detallará el proceso seguido para la representación de los marcos

que encapsularán las anotaciones y, en segundo lugar, se detallará el proceso seguido para representar los textos e iconos.

3.4.1 Representación de los marcos que encapsulan las anotaciones

Para la realización de los marcos que encapsulan las anotaciones se han usado diferentes puntos de nuestro escenario y luego los hemos unido formando nuestro marco. Para llevar esto a cabo se han usado los métodos de la librería coin de Pivy, *SoCoordinate3* y *SoIndexedLineSet*.

Con el primer método almacenaríamos todos los puntos en 3D que vamos a utilizar y con el segundo especificaríamos el orden en que se van a unir dichos puntos formando segmentos. Se va a mostrar a continuación un ejemplo:

```
coords = coin.SoCoordinate3()
coords.point.setValues([0,0,0],[2,0,0],[2,2,0],[0,2,0])
lines = coin.SoIndexedLineSet()
lines.coordIndex.setValues(0,6,[0,1,-1,2,3,0])
```

La última línea indica que empezando desde el índice 0, se van a insertar seis valores cuyo significado son: un segmento desde el punto 0 al 1, un paro (el valor -1 indica un paro), otro segmento del punto 2 al 3 y de este al 0. Formando así una especie de cuadrado con su parte derecha abierta.

En nuestro caso, nuestros puntos en 3D van a variar en todo momento dependiendo de los elementos que contenga la anotación que vamos a representar. Por ello, en cada recomputación, se van a recalcular dichos puntos por si hubieran sufrido alguna variación.

A la hora de calcular nuestros puntos de interés, hemos dividido nuestra anotación en tres partes: una parte sería el segmento que va desde la cara seleccionada al punto seleccionado por el usuario para colocar la anotación, la segunda parte sería la correspondiente al encapsulado de todas las tolerancias geométricas, y la tercera parte seria la encapsulación correspondiente a la referencia datum (véase la Ilustración 20). De modo que estos dos últimos solo se calcularían en caso de existir sus elementos en la anotación.

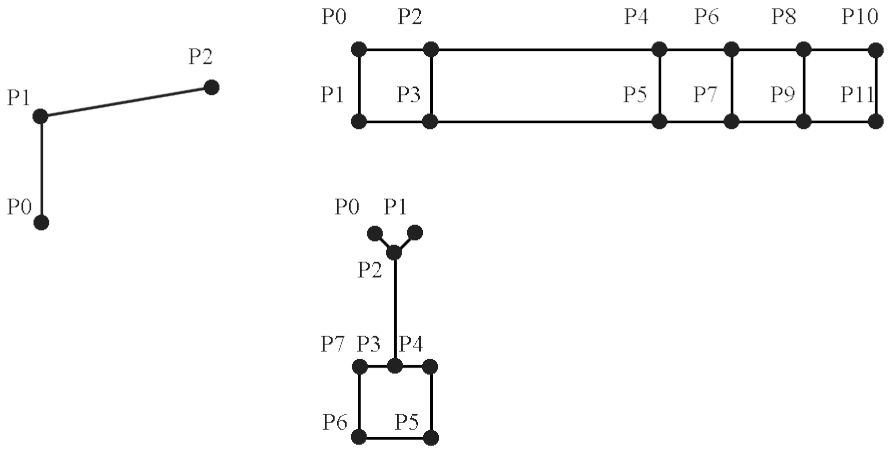

Ilustración 20: Puntos para la creación del marco

En primer lugar, si en nuestra anotación existe por lo menos una tolerancia geométrica o una referencia datum, se realizaría un primer segmento en perpendicular a la cara seleccionada que iría desde el centro de masas de la cara en caso de ser una cara normal o desde el vértice superior en caso de tratarse de una forma cilíndrica hasta tres cuartas partes de la altura del punto seleccionado por el usuario para la colocación de la anotación. Finalmente se realizará un nuevo segmento desde este último punto al punto seleccionado por el usuario para la colocación de la anotación.

Cabe destacar, que en caso de tratarse de una forma cilíndrica la que hubiera que etiquetar, el plano de anotación seleccionado para su representación se desplazaría hasta estar a la altura del vértice superior sin necesidad de tener que crear otro plano de anotación que pase justo por ese punto y que así quede una representación más clara.

En segundo lugar, irían los puntos que formarían el marco de las tolerancias geométricas. En este caso nos hemos ayudado del parámetro *LineScale* de las anotaciones para definir la distancia entre puntos. Este valor por defecto vale 1 mm y es editable por el usuario. Si nos fijamos en el marco de las tolerancias geométricas de la Ilustración 20, la distancia entre P0 y P2 sería dos veces el valor de *LineScale*. Esta distancia sería igual a la distancia entre P0 y P1, P1 y P3, y P3 y P2. Formando así, un cuadrado donde dentro se colocará el icono correspondiente a la tolerancia geométrica seleccionada por el usuario.

La distancia entre P2 y P4 sería dependiente de su contenido de modo que, si tenemos el símbolo del diámetro, equivaldría a dos veces el valor de *LineScale*; si

tenemos un modificador, equivaldría a otros dos por el valor *LineScale*; y por último le sumaríamos el tamaño del texto pertinente al valor de la tolerancia multiplicado también por el valor de *LineScale*.

Finalmente, por cada referencia datum existente en el sistema datum, colocaríamos otro cuadrado de las mismas dimensiones que el primero. Todo este proceso para el encapsulado de las tolerancias geométricas se repetiría por el número total de tolerancias geométricas existentes en dicha anotación, añadiéndose una debajo de la otra.

En tercer y último lugar, nos faltaría definir los puntos correspondientes para dibujar el marco que encapsule la referencia datum en caso de haberla. Para ello en primer lugar, comprobamos si existe algúna tolerancia geométrica y en caso de no existir se dibujaría un cuadrado igual que el primero de las tolerancias geométricas. Este nos servirá de base para apoyar nuestra referencia datum. Debajo de esto, o de la última tolerancia geométrica, iría colocada la tercera composición de la Ilustración 20.

En este último caso, los segmentos formados por P0, P1 y P2 serían dos vértices correspondientes a un triángulo equilátero de lado *LineScale*. El punto P3 se encontraría a una altura respecto de P0 de tres veces el valor de *LineScale*. Y finalmente, el cuadrado que encapsulará el valor de la referencia datum será de las mismas dimensiones que el usado para encapsular el símbolo de la tolerancia geométrica.

A la hora de utilizar estos datos para su representación, todos los puntos se almacenarán en un mismo marray de puntos, al igual que los segmentos y finalmente se representarían sobre el escenario con los métodos de Coin3d mencionados anteriormente.

Durante este proceso se ha encontrado algún problema a la hora de calcular dichos puntos ya que dependiendo del plano en que se situara la cara que se estaba etiquetando, se debía de utilizar diferentes direcciones para representar los marcos. De modo que si utilizábamos la misma dirección respecto de nuestra cara seleccionada se obtenían resultados como los de la Ilustración 21.

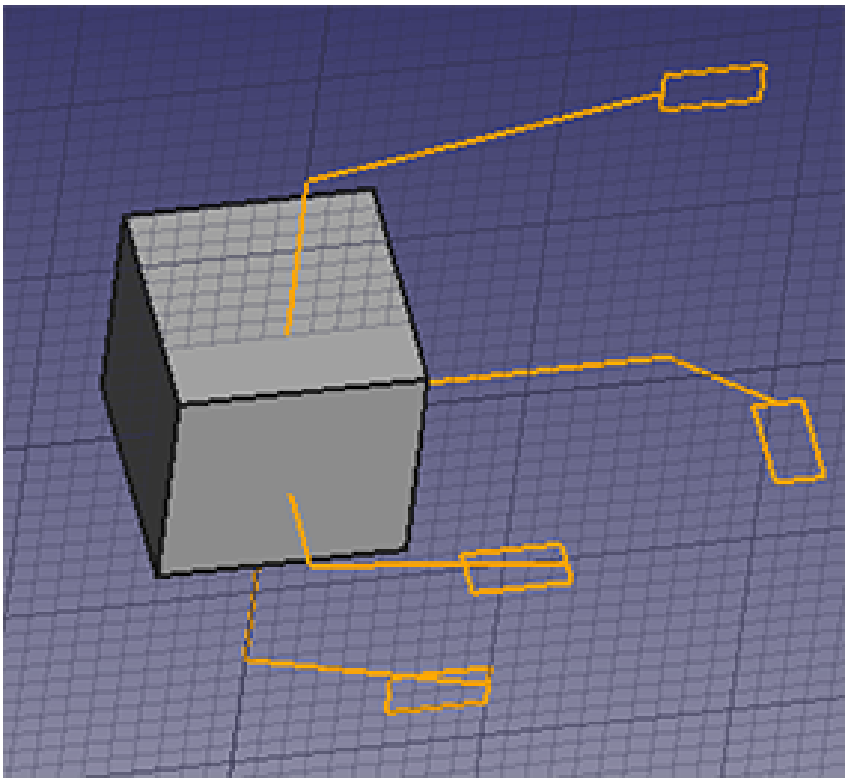

Ilustración 21: Error de representación en los marcos que encapsulan las tolerancias

Para solucionar este problema se han definido dos direcciones, *Vertical* y *Horizontal* para hacer desplazamientos de los puntos en dichas direcciones. Estas direcciones se calcularán a partir del vector dirección del plano de anotación y para ello se ha utilizado el trozo de código que se muestra a continuación y obteniendo como resultado lo expuesto en la Ilustración 22.

```
X = FreeCAD.Vector(1.0,0.0,0.0)
Y = FreeCAD.Vector(0.0,1.0,0.0)
Direction = X if abs(X.dot(obj.AP.Direction)) < 0.8 else Y
Vertical = obj.AP.Direction.cross(Direction).normalize()
Horizontal = Vertical.cross(obj.AP.Direction).normalize()
```

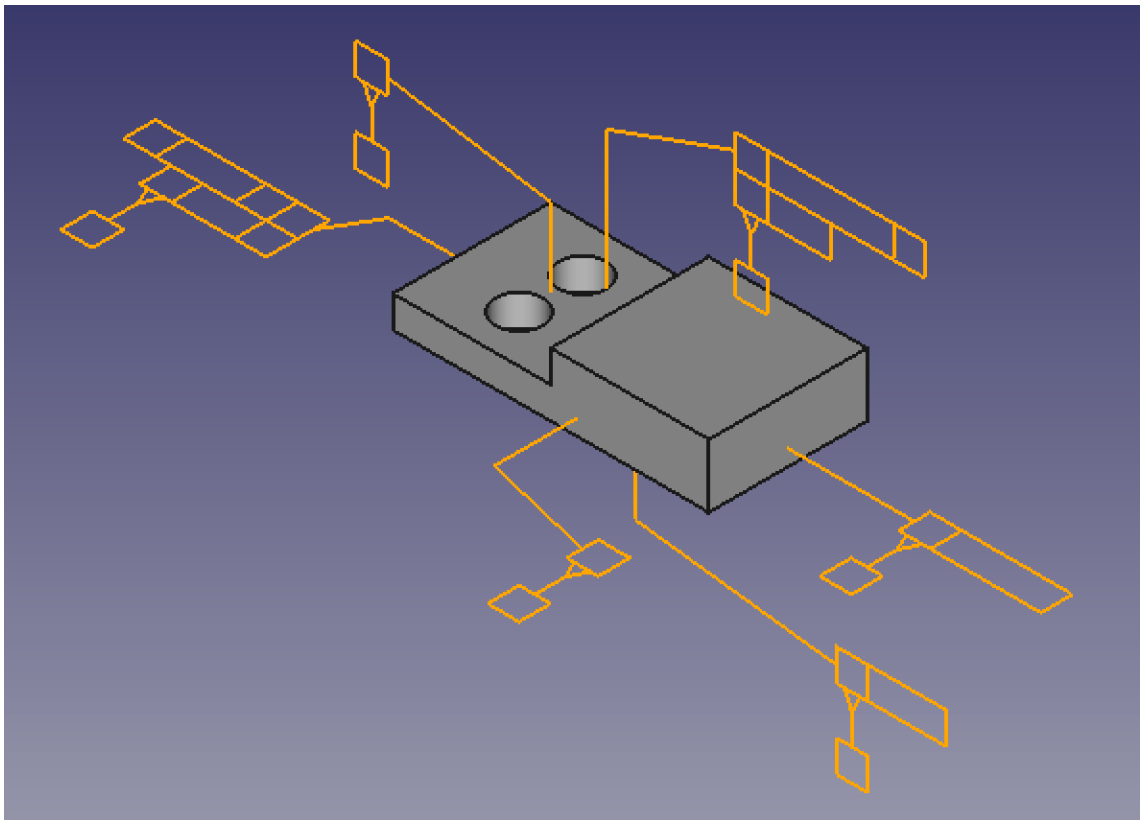

Ilustración 22: Representación de los marcos de las anotaciones

También cabe destacar que estos marcos se han asociado con los propios objetos de nuestro módulo de forma que, al pasar el ratón por encima de la representación de alguna anotación, nos detectará automáticamente de que anotación se trata y si pulsamos sobre ella, se nos seleccionará la anotación correspondiente en la lista de elementos del proyecto.

3.4.2 Representación de textos e iconos 2D en el espacio 3D

Una vez que ya tenemos los marcos que encapsulan las anotaciones, lo que nos falta es rellenarlos con la información pertinente. Para ello hemos vuelto a usar algunos métodos de la librería coin de Pivy: *SoAsciiText* para la representación de los textos, y *SoFaceSet* y *SoTexture2Transform* para la representación de los iconos. Además, también hemos usado los métodos *SoTransform* en los textos y *SoVRMLCoordinate* en los iconos, para establecer su posición en el espacio 3D.

La cantidad de los textos a utilizar dependerá del número de anotaciones que contenga una anotación. Por esta razón se intentó ir creando los elementos de coin que nos hacían falta a medida que se necesitaba un texto nuevo. Pero al realizar esta acción la textura de los textos no salía correctamente y se observaba con una especie de trama a rallas. Esto es debido a que los elementos que se van a usar de coin se deben de crear en el método *attach* del objeto que lo va a utilizar inicializándolos en la creación del objeto. Por esta razón se optó por crear en dicho método veinte

elementos de texto para poder abastecer nuestras necesidades. Lo mismo se hizo con los iconos.

Además del texto pertinente para las tolerancias geométricas y las referencias datum también hay otros textos necesarios. Uno sería un texto que aparecerá en la parte superior de la anotación en donde en caso de haber más de una cara seleccionada se mostrará el número de elementos al que pertenece dicha anotación. A continuación de este valor se mostraría, en caso de haberlo, la tolerancia correspondiente al diámetro de la cara seleccionada. Este valor, en caso de tratarse de un valor establecido por los límites inferior y superior, éstos se representarán separados por un guion y en caso de tratarse de una tolerancia de ± se representará el valor con dicho símbolo. No obstante, hay que añadir que la librería de coin3D solo soporta dicho símbolo a partir de su versión 4.0. Por ello, en nuestro código hay un fragmento donde se comprueba la versión que estamos utilizando para que en caso de estar utilizando una versión inferior, se sustituya este símbolo por los caracteres más y menos (+-).

En cuanto a los iconos, nos encontramos con que no había ningún método de la librería de coin3D que nos permitiera colocar directamente un icono sobre nuestro escenario. Por ello lo que se intentó fue realizar figuras geométricas y a la misma aplicarle una textura pasándole como imagen nuestro icono deseado. Lo primero que se intento fue crear un cubo con una profundidad de cero intentando que así se quedará una sola cara sobre la que dibujar nuestro icono. Pero esto resultó dibujar nuestro icono en la cara delantera y trasera del cubo dibujando una especie de efecto espejo como se puede apreciar en la Ilustración 23.

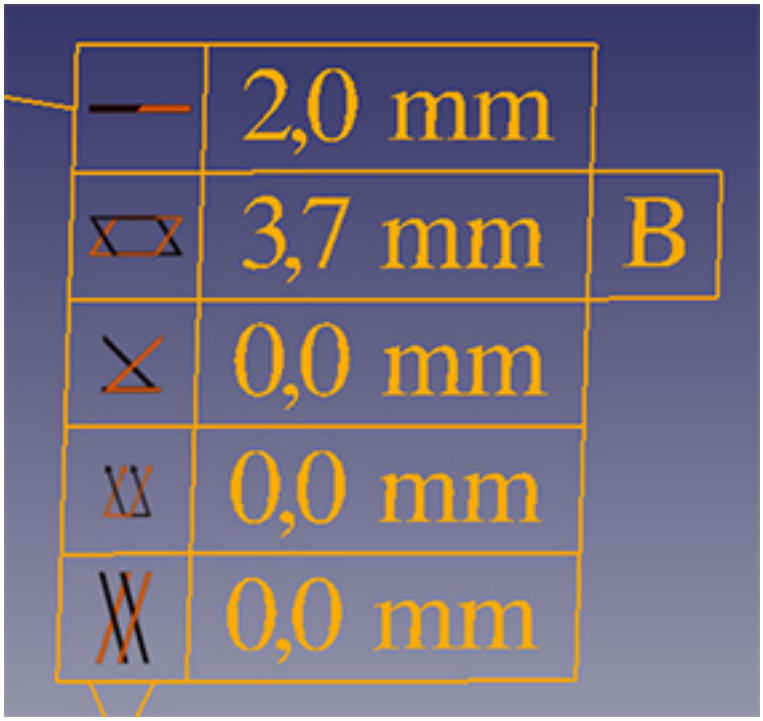

Ilustración 23: Error de espejo en los iconos de las tolerancias geométricas

Posteriormente se descubrió en la documentación de la librería de coin3D que existía un método para realizar planos sobre los que poder aplicar nuestra textura. En este nuevo método le pasamos cuatro puntos como argumento los cuales definirán nuestro plano. Pero, además, para que no se vea la imagen rotada, también hay que especificarlos los planos de representación de nuestra imagen. Esto se consiguió con el método *SoTextureCoordinatePlane*.

No obstante, con esto se crea una especie de trama que empieza en el origen de coordenadas y se va expandiendo mostrando únicamente el trozo correspondiente a nuestro plano. Esto puede provocar que nuestro plano se encuentre a caballo entre la representación de dos imágenes como se puede observar en la Ilustración 24. Para solucionar este problema se ha utilizado otro método *SoTexture2Transform*, con el cual podemos desplazar dicha trama. En nuestro caso se calcula la diferencia que existe en vertical y en horizontal entre el origen de coordenadas y nuestro plano y se aplica dicho desplazamiento a la trama de nuestro icono obteniendo como resultado la Ilustración 25.

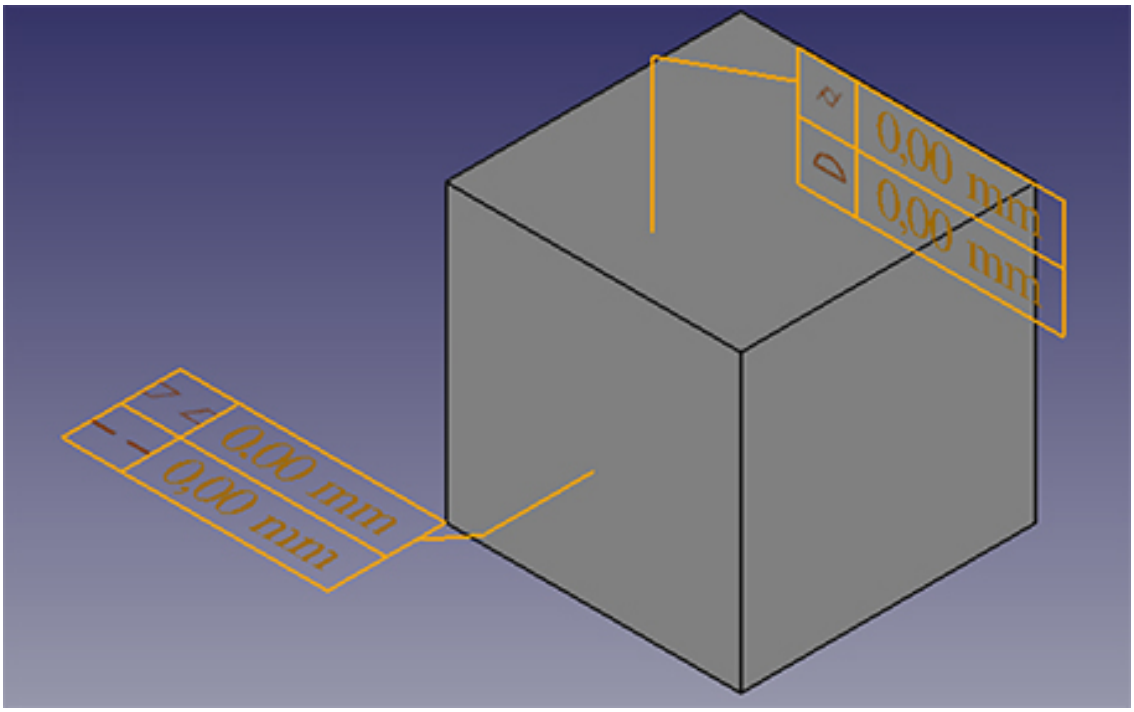

Ilustración 24: Error en la posición de los iconos de las tolerancias geométricas

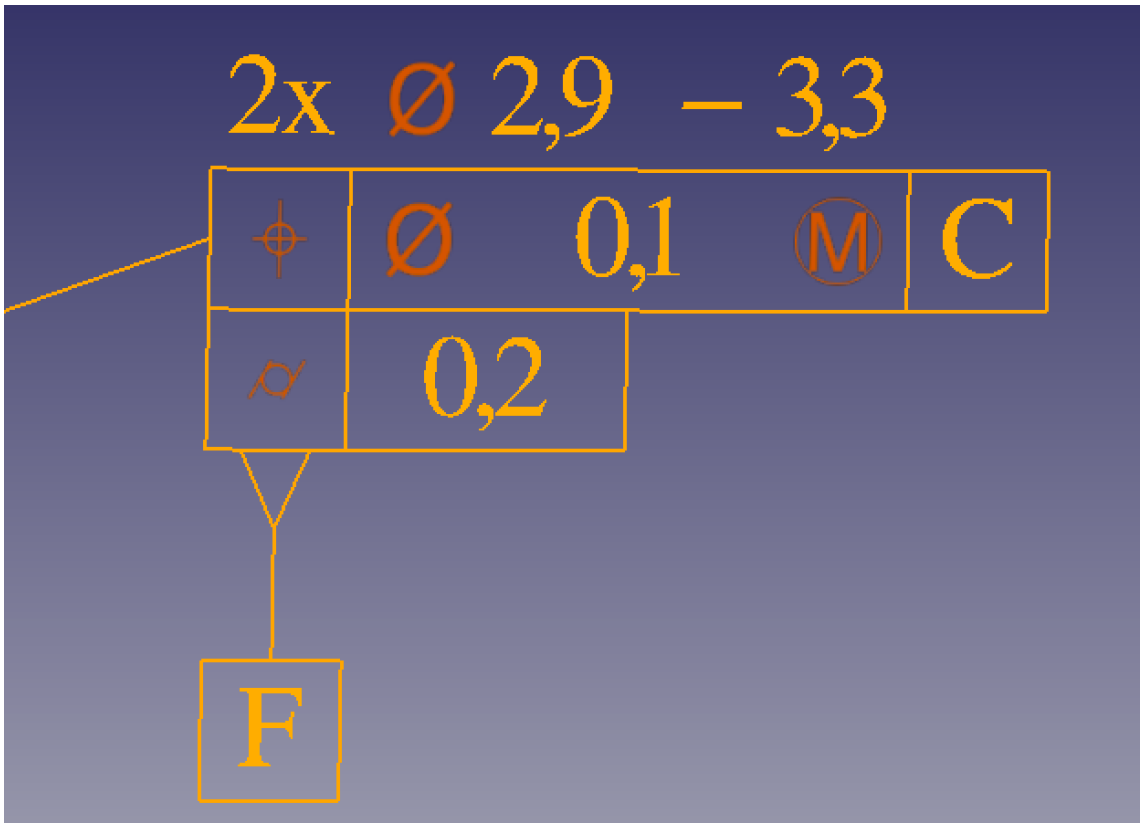

Ilustración 25: Representación de etiquetado GD&T

Como se ha comentado anteriormente, las anotaciones disponen de una variedad de parámetros que se pueden utilizar para personalizar las mismas. En la Ilustración 26 se puede observar la diferencia respecto a la anterior, donde ahora se le ha subido el número de decimales a tres y se le ha activado la opción de mostrar unidades. En la Ilustración 27 se le ha aumentado el número de pixeles del ancho de las líneas.

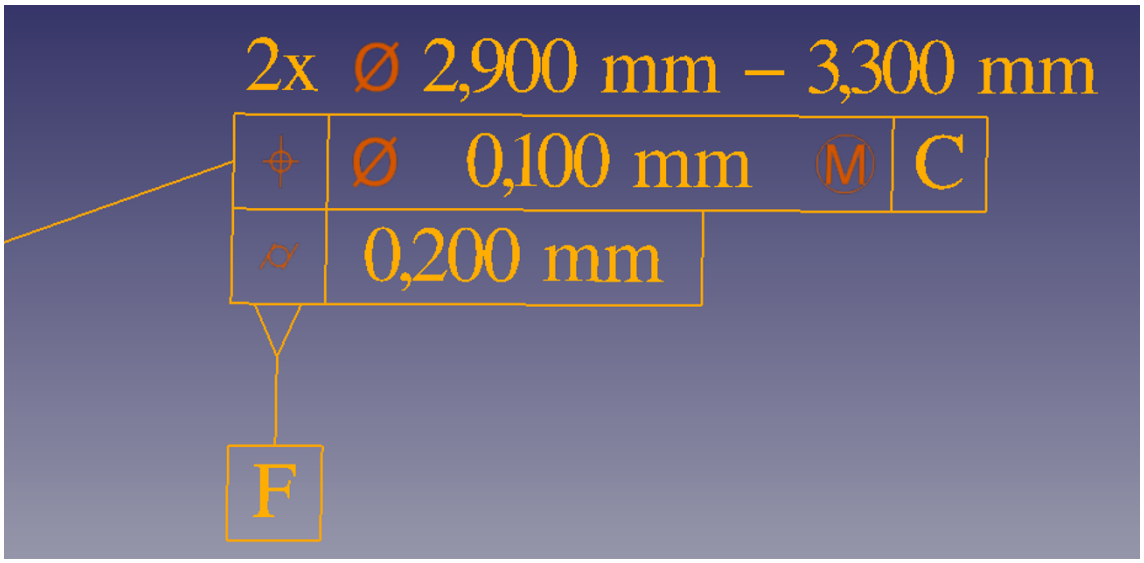

Ilustración 26: Representación de etiquetado GD&T con parámetros modificados

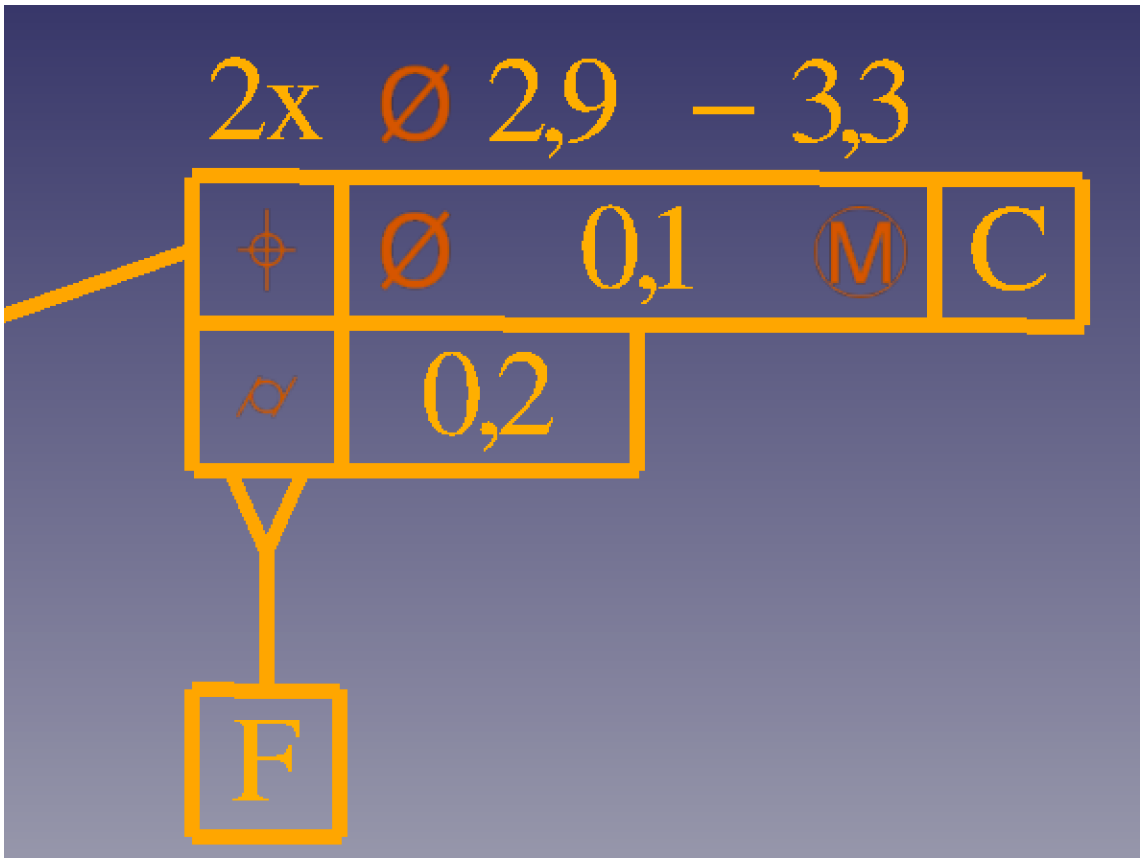

Ilustración 27: Representación de etiquetado GD&T con las líneas más gruesas

En la Ilustración 28 se observa cómo quedaría nuestra anotación al aumentar la propiedad de *LineScale*. Esto, además de escalar las líneas que formarían el marco de la anotación, también redimensiona los iconos ya que el plano que se utiliza para su inserción también se gasta de dicha propiedad para definirlo. Además de todo esto, también podemos cambiar el color de las líneas y de los textos como se puede apreciar en la Ilustración 29.

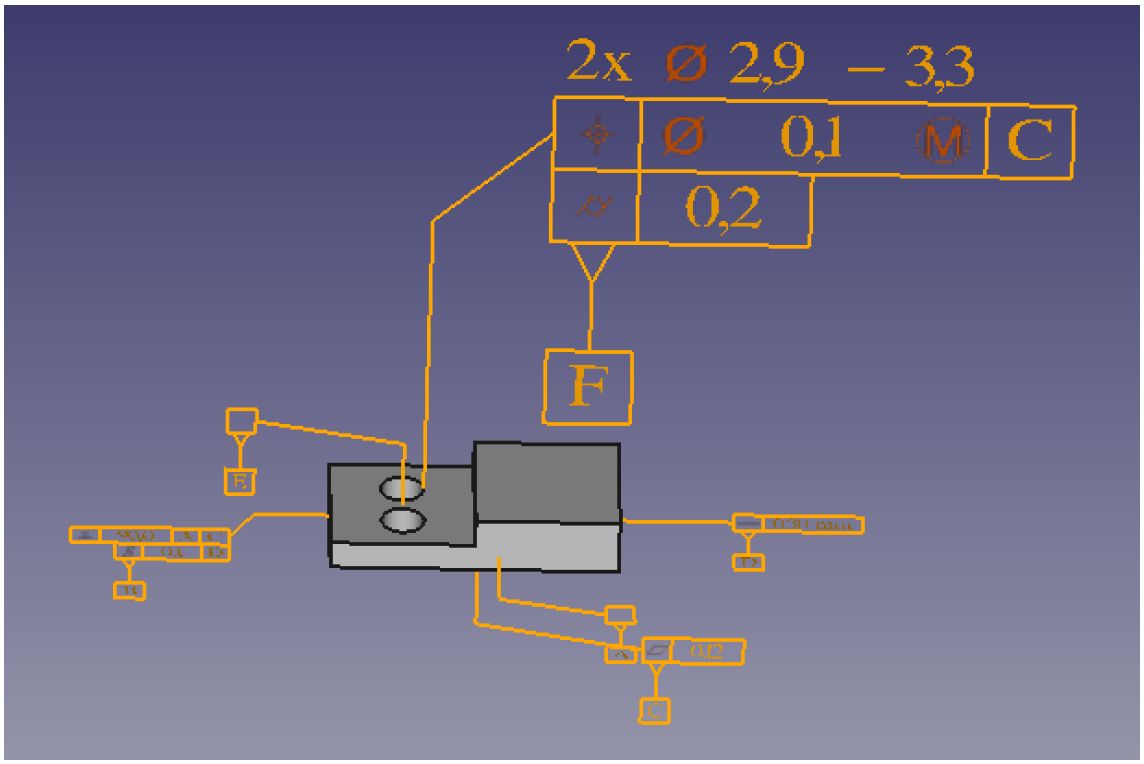

Ilustración 28: Representación de etiquetado GD&T con un escalado en las líneas

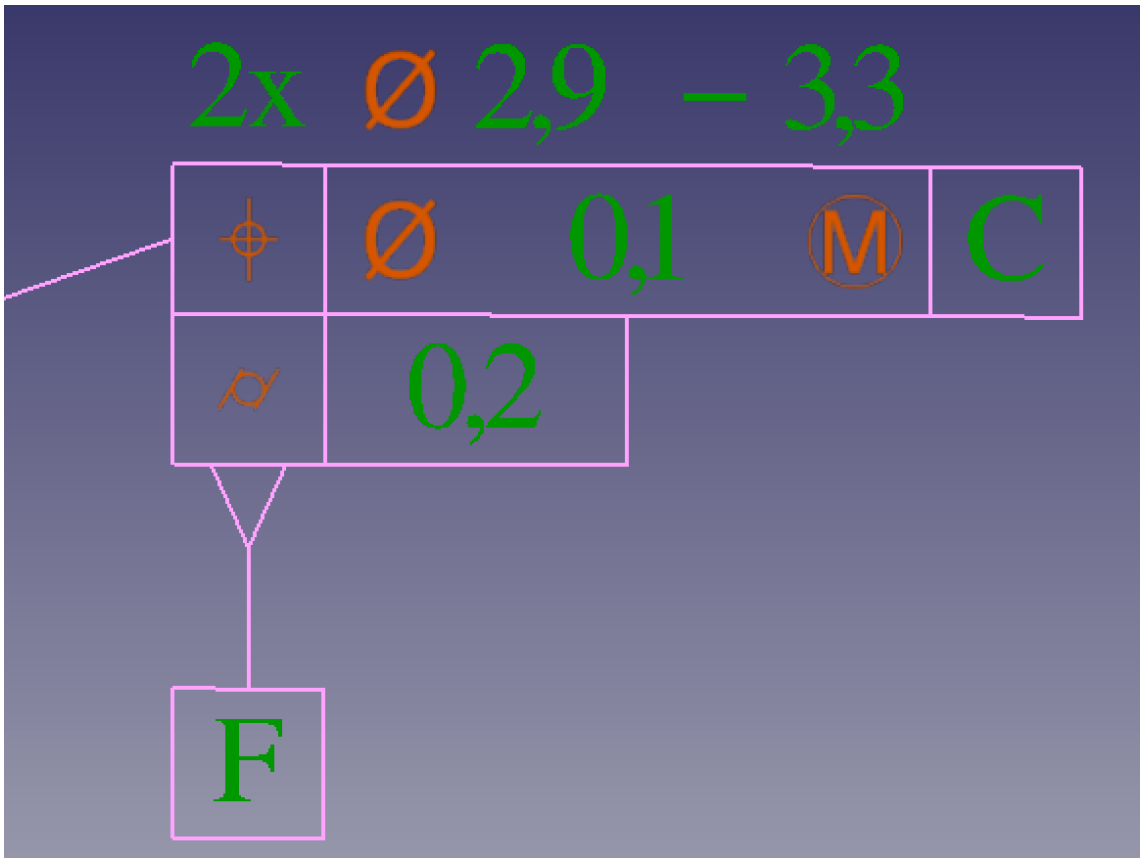

Ilustración 29: Representación de etiquetado GD&T con un color diferente para el texto y líneas

3.5 Preferencias de configuración

Como se ha podido observar, las anotaciones poseen un alto grado de personalización. Por ello, se ha creado un apartado de preferencias para nuestro módulo, accesible desde las propias preferencias de la aplicación, en donde se puede especificar el valor por defecto de dichos valores que les aportan personalización a nuestras anotaciones. Véase la Ilustración 30.

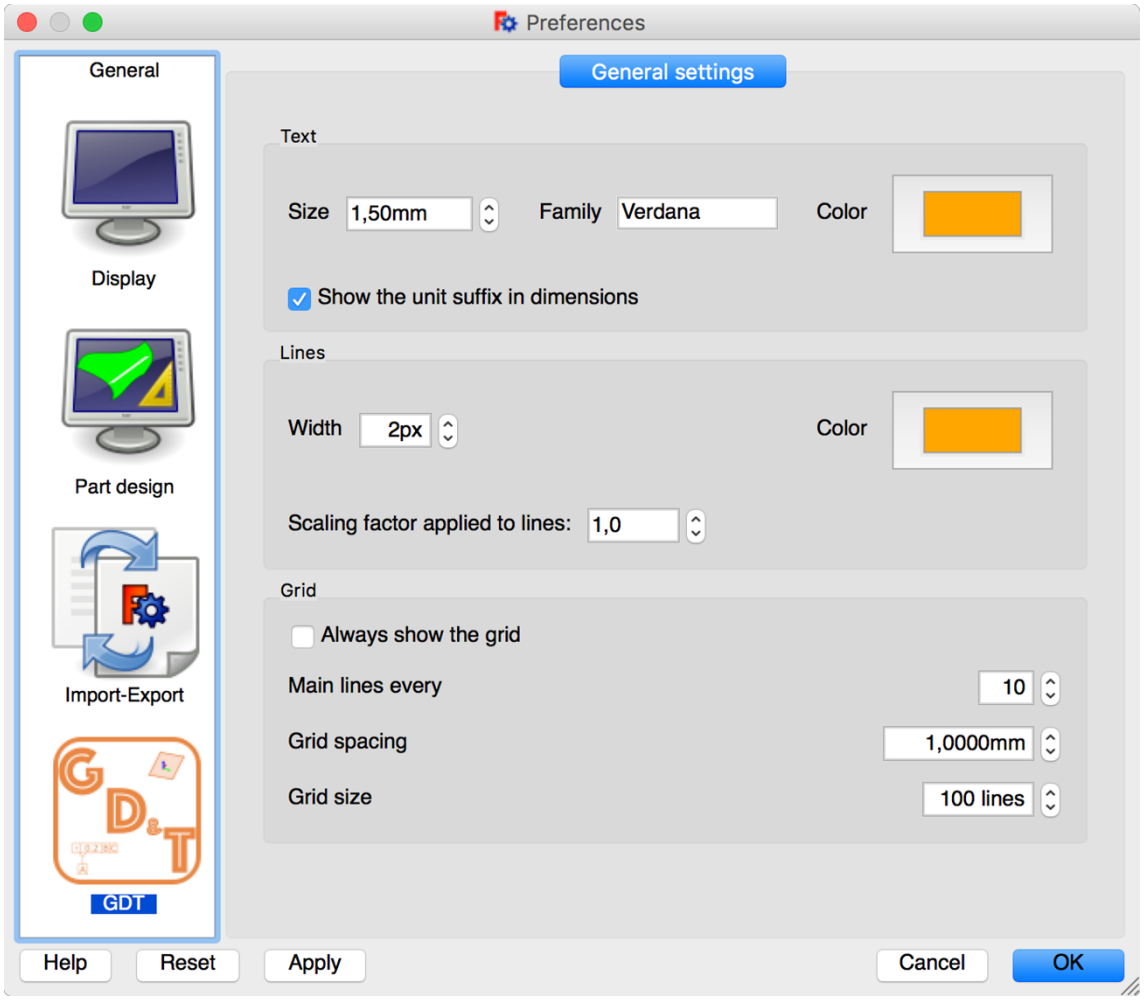

Ilustración 30: Preferencias de configuración para el módulo GD&T

A la hora de crear una anotación, ésta tomará como valor de los parámetros utilizados para personalización los establecidos en estas preferencias. Posteriormente, si se modifica algún parámetro en las preferencias, estos cambios solo se verán reflejados en las nuevas anotaciones.

Además, en la Ilustración 30, también se puede observar un apartado reservado para la representación de la cuadrícula que se utiliza para visualizar los planos de anotación. Como se ha dicho anteriormente, se usa una herramienta del módulo Part para realizar dicha acción. Por lo tanto, al modificar estas variables, en realidad se estarán modificando los valores de configuración del módulo Part. A excepción de la

casilla donde se puede seleccionar si se desea que se muestre siempre esta cuadrícula.

En caso de que esta última casilla esté desmarcada, la cuadrícula solo se mostrará cuándo se esté utilizando un plano de anotación y se esconderá automáticamente al finalizar su utilización para así dejar una vista más clara del escenario.

3.6 Funcionamiento

A la hora de realizar un etiquetado GD&T, lo primero que debemos hacer es definir un plano de anotación sobre el que se pintarán las anotaciones que realicemos. Para ello, primero hay que seleccionar una cara de la pieza y pulsar al comando que hay en la barra de herramientas de añadir plano de anotación. Esto nos definirá un plano sobre esa cara y además le podremos aplicar un offset para colocar nuestro plano de anotación a la altura que queramos en paralelo sobre la cara seleccionada. Definiendo así un plano de anotación.

El siguiente paso que debemos de realizar es la creación de una referencia datum o de una característica geométrica. Aunque cabe destacar que si lo primero que se va a crear es una característica geométrica, ésta no podrá contener ningún sistema datum ya que todavía no habrá ninguno creado. No obstante, este podrá añadirse posteriormente modificando la tolerancia geométrica desde el inventario de elementos de GD&T.

En cualquier caso, a la hora de crear una referencia datum o una característica geométrica, el usuario deberá de elegir algunos parámetros que definirán el elemento a crear. Entre ellos el plano de anotación sobre el que se representará la anotación. Todo seguido se deberá de seleccionar un punto sobre dicho plano. Punto sobre el cual empezará la representación del marco que encapsulará los datos de nuestra anotación.

Además, en cualquier momento, se podrá crear un sistema datum con los elementos de referencia datum que se hayan creado y este sistema se podrá aplicar a las tolerancias geométricas que consideremos oportunas. Esto indicará que dicha tolerancia geométrica será aplicada a la cara o caras que corresponde la anotación respecto de las referencias datum que compone el sistema datum asociado.

Posteriormente, si se desea aplicar una referencia datum o una tolerancia geométrica a una cara o caras que ya tengan asociadas una anotación, nuestro módulo detectará automáticamente que anotación es y añadirá el nuevo elemento a dicha anotación.

Por tanto, el funcionamiento de nuestro módulo se podría resumir en que hay que añadir tolerancias geométricas a nuestro diseño pero que para poder llevar esto a cabo, primero habría que crear diferentes elementos como planos de anotación para colocar nuestra anotación en el lugar deseado, o referencias datum y sistemas datum

para aportarle información necesaria a nuestras tolerancias geométricas. De modo que hay que ir creando diferentes elementos hasta que tengamos nuestra pieza completamente etiquetada con todas las tolerancias geométricas que precisemos indicar.

4 Conclusiones y trabajos futuros

4.1 Conclusiones

A lo largo de este proyecto se ha realizado un módulo desarrollado desde cero completamente en Python con el que se han conseguido cumplir los objetivos inicialmente marcados:

- Con la creación de las anotaciones se ha permitido añadir la información de GD&T al propio diseño, vinculándose así las especificaciones de diseño, fabricación y calidad. Además, está información va ligada a la propia pieza de modo que, si esta es modificada, las anotaciones creadas se desplazarán a la nueva posición dónde se encuentre la cara de la pieza que tenga asignada en sus parámetros.

- Se ha cumplido con el estándar ISO16792 utilizando una simbología propia del estándar que permite realizar una comunicación precisa y efectiva que no lleva a posibles malinterpretaciones de las representaciones utilizadas para indicar las diferentes tolerancias geométricas.

- Con la realización de los diferentes iconos y símbolos en una misma tonalidad de color y la realización de widgets ordenados se ha implementado una interfaz homogénea e integrada con las herramientas de diseño técnico y 3D de la propia aplicación FreeCAD que son fáciles de utilizar para el usuario final.

- Todo esto se ha desarrollado como software libre bajo la licencia LGPLv2+ para una aplicación también de software libre en la que una gran cantidad de usuarios aporta para que la misma siga creciendo. Se puede acceder a una versión actualizada de todo el código desarrollado en el siguiente repositorio de GitHub: https://github.com/juanvanyo/FreeCAD-GDT

Además de haber cumplido todos los objetivos inicialmente marcados, también se han desarrollado otros elementos que sirven para mejorar la experiencia de usuario como sería el alto grado de personalización que hay en las anotaciones, personalización la cual se puede guardar en las preferencias de la propia aplicación para futuros usos, o la autogestión de los nombres de los objetos a crear, los cuales van avanzando numéricamente automáticamente conforme se van creando, o la gestión de diferentes elementos expuestos a lo largo de la memoria que sirven para mejorar la experiencia de usuario.

Por último, también cabe destacar que este proyecto va a servir de herramienta para la realización de diversas tareas por parte del grupo PRAIA del Instituto Tecnológico de Informática. Estas tareas serán principalmente enfocadas al proceso de control de calidad de fabricación de piezas mediante un sistema de inspección con Visión Artificial. De modo que se comparen los datos extraídos con la Visión artificial con los datos del etiquetado GD&T realizado en este proyecto.

4.2 Trabajos futuros

Una vez alcanzados con éxito los objetivos de este proyecto, se plantean los siguientes trabajos que dotarían de mayores prestaciones y funcionalidades a nuestro módulo para FreeCAD:

- Actualmente, todos los datos de los elementemos generados con el módulo de GD&T son guardados en el formato propio de FreeCAD FCStd y con ello se consigue que, al guardar y abrir los archivos con el FreeCAD, esta información se mantenga. Un trabajo interesante a realizar sería la exportación de estos datos a otros formatos más utilizados como por ejemplo el step y así no tener que depender de la aplicación FreeCAD para poder acceder y utilizar los datos de GD&T.

- Actualmente este módulo únicamente está disponible en inglés y por ello sería un trabajo interesante la realización de la traducción a diferentes idiomas para llegar a un mayor número de usuarios.

- Otro trabajo que podría dotar de mayores prestaciones a nuestro módulo sería el implementar algo parecido a lo que realiza la aplicación vista en los antecedentes, GD&T Advisor, con la que se puede diagnosticar nuestra pieza y determinar que tolerancias son las adecuadas para cada parte de la misma además de diversas ayudas que pudieran agilizar el proceso de etiquetado.

5 Referencias

[1] J. Bodnar, «ZetCode,» [En línea]. Available: http://zetcode.com/gui/pysidetutorial/widgets/. [Último acceso: 27 Marzo 2017].

[2] «Coin Documentation,» [En línea]. Available: https://grey.colorado.edu/coin3d/. [Último acceso: 27 Marzo 2017].

[3] FreeCAD, «FreeCAD,» [En línea]. Available: http://freecadweb.org/. [Último acceso: 27 Marzo 2017].

[4] FreeCAD, «FreeCAD Documentation,» [En línea]. Available: https://www.freecadweb.org/wiki/Main_Page. [Último acceso: 27 Marzo 2017].

[5] FreeCAD, «FreeCAD Forum,» [En línea]. Available: http://forum.freecadweb.org/. [Último acceso: 27 Marzo 2017].

[6] hamish2014, «GitHub,» [En línea]. Available: https://github.com/hamish2014/FreeCAD_drawing_dimensioning. [Último acceso: 27 Marzo 2017].

[7] IDA-STEP, «IDA-STEP,» [En línea]. Available: http://www.ida-step.net/components/editors/gdt. [Último acceso: 27 Marzo 2017].

[8] J. Martinez, «FreeCAD: simple Coin3d plot,» [En línea]. Available: http://linuxforanengineer.blogspot.com.es/2014/09/freecad-simple-coin3d-example.html. [Último acceso: 27 Marzo 2017].

[9] Program Creek, «Programcreek,» [En línea]. Available: http://www.programcreek.com/python/. [Último acceso: 27 Marzo 2017].

[10] Python Software Foundation, «Python,» [En línea]. Available: https://www.python.org/. [Último acceso: 27 Marzo 2017].

[11] Sigmetrix, «GD&T Advisor Software,» [En línea]. Available: http://www.sigmetrix.com/products/gdt-software/. [Último acceso: 27 Marzo 2017].

[12] Stack Exchange Inc, «stackoverflow,» [En línea]. Available: http://stackoverflow.com/. [Último acceso: 2017 Marzo 2017].

[13] Tec-Ease, «Tec-Ease,» [En línea]. Available: https://www.tec-ease.com/quality-101-defining-gdt.php. [Último acceso: 27 Marzo 2017].

[14] Y. van Havre, «A FreeCAD manual,» [En línea]. Available: https://www.gitbook.com/book/yorikvanhavre/a-freecad-manual/details. [Último acceso: 27 Marzo 2017].

[15] Y. van Havre, «TaskPanel.py,» [En línea]. Available: https://github.com/yorikvanhavre/FreeCAD/blob/master/src/Mod/TemplatePyMod/TaskPanel.py. [Último acceso: 27 Marzo 2017].

[16] J. Vañó Cerdá, «GitHub,» [En línea]. Available: https://github.com/juanvanyo/FreeCAD-GDT. [Último acceso: 27 Marzo 2017].

[17] R. Wachenchauzer, M. Manterola, M. Curia, M. Medrano y N. Paez, «Algoritmos de Programación con Python,» [En línea]. Available: http://librosweb.es/libro/algoritmos_python/. [Último acceso: 27 Marzo 2017].

[18] Q. Xia, «Module developer's guide to FreeCAD source code,» [En línea]. Available: https://www.iesensor.com/download/FreeCAD_Mod_Dev_Guide__20160920.pdf. [Último acceso: 27 Marzo 2017].

6 Anexos

6.1 Anexo 1: InitGui.py

```
# GeometricDimensioningAndTolerancing gui init module
#***************************************************************************
#*                                                                         *
#*   Copyright (c) 2016 Juan Vañó Cerdá <juavacer@inf.upv.es>              *
#*                                                                         *
#*   This program is free software; you can redistribute it and/or modify  *
#*   it under the terms of the GNU Lesser General Public License (LGPL)    *
#*   as published by the Free Software Foundation; either version 2 of     *
#*   the License, or (at your option) any later version.                   *
#*   for detail see the LICENCE text file.                                 *
#*                                                                         *
#*   This program is distributed in the hope that it will be useful,       *
#*   but WITHOUT ANY WARRANTY; without even the implied warranty of        *
#*   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the         *
#*   GNU Library General Public License for more details.                  *
#*                                                                         *
#*   You should have received a copy of the GNU Library General Public     *
#*   License along with this program; if not, write to the Free Software   *
#*   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  *
#*   USA                                                                   *
#*                                                                         *
#***************************************************************************
import GDT

class GeometricDimensioningAndTolerancingWorkbench ( Workbench ):
        Icon = ':/dd/icons/GDT.svg'
        MenuText = 'GD&T'
        ToolTip = 'Geometric Dimensioning & Tolerancing'

        def GetClassName(self):
                return "Gui::PythonWorkbench"

        def Initialize(self):
                # load the module
                # import GD&T tools
                try:
                        import datumFeature
                        import datumSystem
                        import geometricTolerance
                        import annotationPlane
                        import inventory
                except ImportError:
                        FreeCAD.Console.PrintWarning("Error: Initializing one or more of the GD&T modules failed, GD&T
will not work as expected.\n")

                self.cmdList = ['dd_datumFeature','dd_datumSystem','dd_geometricTolerance','dd_annotationPlane']
                self.inventory = ['dd_inventory']
                self.appendToolbar("GD&T Tools",self.cmdList+self.inventory)
                self.appendMenu("GD&T Tools",self.cmdList+self.inventory)

                FreeCADGui.addIconPath(':/dd/icons')
                FreeCADGui.addPreferencePage( ':/dd/ui/preferences-gdt.ui','GDT' )

                Log ("Loading Geometric Dimensioning & Tolerancing... done\n")

        def Activated(self):
            # do something here if needed...
                Msg ("Geometric Dimensioning & Tolerancing workbench activated\n")

        def Deactivated(self):
            # do something here if needed...
                Msg ("Geometric Dimensioning & Tolerancing workbench desactivated\n")

        def ContextMenu(self, recipient):
        # "This is executed whenever the user right-clicks on screen"
        # "recipient" will be either "view" or "tree"
                showCmdList = True
                if FreeCADGui.Selection.getSelection():
                        for i in range(len(FreeCADGui.Selection.getSelectionEx()[0].SubObjects)):
                                if FreeCADGui.Selection.getSelectionEx()[0].SubObjects[i].ShapeType == 'Face':
```

```
                                        pass
                                else:
                                        showCmdList = False
                        else:
                                showCmdList = False
                        if showCmdList:
                                self.appendContextMenu("",self.cmdList) # add commands to the context menu
                        self.appendContextMenu("",self.inventory)

FreeCADGui.addWorkbench(GeometricDimensioningAndTolerancingWorkbench)
```

6.2 Anexo 2: GDT.py

```
# -*- coding: utf-8 -*-

#***************************************************************************
#*                                                                         *
#*   Copyright (c) 2016 Juan Vanyo Cerda <juavacer@inf.upv.es>             *
#*                                                                         *
#*   This program is free software; you can redistribute it and/or modify  *
#*   it under the terms of the GNU Lesser General Public License (LGPL)    *
#*   as published by the Free Software Foundation; either version 2 of     *
#*   the License, or (at your option) any later version.                   *
#*   for detail see the LICENCE text file.                                 *
#*                                                                         *
#*   This program is distributed in the hope that it will be useful,       *
#*   but WITHOUT ANY WARRANTY; without even the implied warranty of        *
#*   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the         *
#*   GNU Library General Public License for more details.                  *
#*                                                                         *
#*   You should have received a copy of the GNU Library General Public     *
#*   License along with this program; if not, write to the Free Software   *
#*   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  *
#*   USA                                                                   *
#*                                                                         *
#***************************************************************************

__title__="FreeCAD GDT Workbench"
__author__ = "Juan Vanyo Cerda <juavacer@inf.upv.es>"
__url__ = "http://www.freecadweb.org"

# Description of tool

import numpy
import FreeCAD as App
import FreeCAD, math, sys, os, DraftVecUtils, Draft_rc
from math import pi
from FreeCAD import Vector
from svgLib_dd import SvgTextRenderer, SvgTextParser
import traceback
import Draft
import Part
from pivy import coin
import FreeCADGui, WorkingPlane
if FreeCAD.GuiUp:
    gui = True
else:
    FreeCAD.Console.PrintMessage("FreeCAD Gui not present. GDT module will have some features disabled.")
    gui = True

try:
    from PySide import QtCore,QtGui,QtSvg
except ImportError:
    FreeCAD.Console.PrintMessage("Error: Python-pyside package must be installed on your system to use the Geometric Dimensioning &
Tolerancing module.")

__dir__ = os.path.dirname(__file__)
iconPath = os.path.join( __dir__, 'Gui','Resources', 'icons' )
path_dd_resources =  os.path.join( os.path.dirname(__file__), 'Gui', 'Resources', 'dd_resources.rcc')
resourcesLoaded = QtCore.QResource.registerResource(path_dd_resources)
assert resourcesLoaded

checkBoxState = True
auxDictionaryDS=[]
for i in range(1,100):
    auxDictionaryDS.append('DS'+str(i))
dictionaryAnnotation=[]
for i in range(1,100):
    dictionaryAnnotation.append('Annotation'+str(i))
```

```
#---------------------------------------------------------------------------
# Param functions
#---------------------------------------------------------------------------

def getParamType(param):
    if param in ["lineWidth","gridEvery","gridSize"]:
        return "int"
    elif param in ["textFamily"]:
        return "string"
    elif param in ["textSize","gridSpacing","lineScale"]:
        return "float"
    elif param in ["alwaysShowGrid","showUnit"]:
        return "bool"
    elif param in ["textColor","lineColor"]:
        return "unsigned"
    else:
        return None

def getParam(param,default=None):
    "getParam(parameterName): returns a GDT parameter value from the current config"
    p = FreeCAD.ParamGet("User parameter:BaseApp/Preferences/Mod/GDT")
    t = getParamType(param)
    if t == "int":
        if default == None:
            default = 0
        return p.GetInt(param,default)
    elif t == "string":
        if default == None:
            default = ""
        return p.GetString(param,default)
    elif t == "float":
        if default == None:
            default = 1
        return p.GetFloat(param,default)
    elif t == "bool":
        if default == None:
            default = False
        return p.GetBool(param,default)
    elif t == "unsigned":
        if default == None:
            default = 0
        return p.GetUnsigned(param,default)
    else:
        return None

def setParam(param,value):
    "setParam(parameterName,value): sets a GDT parameter with the given value"
    p = FreeCAD.ParamGet("User parameter:BaseApp/Preferences/Mod/GDT")
    t = getParamType(param)
    if t == "int": p.SetInt(param,value)
    elif t == "string": p.SetString(param,value)
    elif t == "float": p.SetFloat(param,value)
    elif t == "bool": p.SetBool(param,value)
    elif t == "unsigned": p.SetUnsigned(param,value)

#---------------------------------------------------------------------------
# General functions
#---------------------------------------------------------------------------

def stringencodecoin(ustr):
    """stringencodecoin(str): Encodes a unicode object to be used as a string in coin"""
    try:
        from pivy import coin
        coin4 = coin.COIN_MAJOR_VERSION >= 4
    except (ImportError, AttributeError):
        coin4 = False
    if coin4:
        return ustr.encode('utf-8')
    else:
        return ustr.encode('latin1')

def stringplusminus():
    return ' ± ' if coin.COIN_MAJOR_VERSION >= 4 else ' +- '

def getType(obj):
    "getType(object): returns the GDT type of the given object"
    if not obj:
        return None
    if "Proxy" in obj.PropertiesList:
        if hasattr(obj.Proxy,"Type"):
```

```
            return obj.Proxy.Type
    return "Unknown"

def getObjectsOfType(typeList):
    "getObjectsOfType(string): returns a list of objects of the given type"
    listObjectsOfType = []
    objs = FreeCAD.ActiveDocument.Objects
    if not isinstance(typeList,list):
        typeList = [typeList]
    for obj in objs:
        for typ in typeList:
            if typ == getType(obj):
                listObjectsOfType.append(obj)
    return listObjectsOfType

def getAllAnnotationPlaneObjects():
    "getAllAnnotationPlaneObjects(): returns a list of annotation plane objects"
    return getObjectsOfType("AnnotationPlane")

def getAllDatumFeatureObjects():
    "getAllDatumFeatureObjects(): returns a list of datum feature objects"
    return getObjectsOfType("DatumFeature")

def getAllDatumSystemObjects():
    "getAllDatumSystemObjects(): returns a list of datum system objects"
    return getObjectsOfType("DatumSystem")

def getAllGeometricToleranceObjects():
    "getAllGeometricToleranceObjects(): returns a list of geometric tolerance objects"
    return getObjectsOfType("GeometricTolerance")

def getAllGDTObjects():
    "getAllGDTObjects(): returns a list of GDT objects"
    return getObjectsOfType(["AnnotationPlane","DatumFeature","DatumSystem","GeometricTolerance"])

def getAllAnnotationObjects():
    "getAllAnnotationObjects(): returns a list of annotation objects"
    return getObjectsOfType("Annotation")

def getRGB(param):
    color = QtGui.QColor(getParam(param,16753920)>>8)
    r = float(color.red()/255.0)
    g = float(color.green()/255.0)
    b = float(color.blue()/255.0)
    col = (r,g,b,0.0)
    return col

def getRGBText():
    return getRGB("textColor")

def getTextFamily():
    return getParam("textFamily","")

def getTextSize():
    return getParam("textSize",2.2)

def getLineWidth():
    return getParam("lineWidth",2)

def getRGBLine():
    return getRGB("lineColor")

def hideGrid():
    if hasattr(FreeCADGui,"Snapper") and getParam("alwaysShowGrid") == False:
        if FreeCADGui.Snapper.grid:
            if FreeCADGui.Snapper.grid.Visible:
                FreeCADGui.Snapper.grid.off()
                FreeCADGui.Snapper.forceGridOff=True

def showGrid():
    if hasattr(FreeCADGui,"Snapper"):
        if FreeCADGui.Snapper.grid:
            if FreeCADGui.Snapper.grid.Visible == False:
                Draft.setParam("gridEvery",getParam("gridEvery"))
                Draft.setParam("gridSpacing",getParam("gridSpacing"))
                Draft.setParam("gridSize",getParam("gridSize"))
                FreeCADGui.Snapper.grid.setMainlines(getParam("gridEvery"))
                FreeCADGui.Snapper.grid.setSpacing(getParam("gridSpacing"))
                FreeCADGui.Snapper.grid.setSize(getParam("gridSize"))
                FreeCADGui.Snapper.grid.reset()
                FreeCADGui.Snapper.grid.on()
```

```
            FreeCADGui.Snapper.forceGridOff=False
        else:
            FreeCADGui.Snapper.show()

def getSelection():
    "getSelection(): returns the current FreeCAD selection"
    if gui:
        return FreeCADGui.Selection.getSelection()
    return None

def getSelectionEx():
    "getSelectionEx(): returns the current FreeCAD selection (with subobjects)"
    if gui:
        return FreeCADGui.Selection.getSelectionEx()
    return None

def select(obj):
    "select(object): deselects everything and selects only the working faces of the passed object"
    if gui:
        FreeCADGui.Selection.clearSelection()
        for i in range(len(obj.faces)):
            FreeCADGui.Selection.addSelection(obj.faces[i][0],obj.faces[i][1])

def makeContainerOfData():
    ""
    faces = []
    for i in range(len(getSelectionEx())):
        for j in range(len(getSelectionEx()[i].SubElementNames)):
            faces.append((getSelectionEx()[i].Object, getSelectionEx()[i].SubElementNames[j]))
    faces.sort()
    container = ContainerOfData(faces)
    return container

def getAnnotationObj(obj):
    List = getAllAnnotationObjects()
    for l in List:
        if l.faces == obj.faces:
            return l
    return None

def getAnnotationWithDF(obj):
    List = getAllAnnotationObjects()
    for l in List:
        if l.DF == obj:
            return l
    return None

def getAnnotationWithGT(obj):
    List = getAllAnnotationObjects()
    for l in List:
        for gt in l.GT:
            if gt == obj:
                return l
    return None

def getPointsToPlot(obj):
    points = []
    segments = []
    if obj.GT <> [] or obj.DF <> None:
        X = FreeCAD.Vector(1.0,0.0,0.0)
        Y = FreeCAD.Vector(0.0,1.0,0.0)
        Direction = X if abs(X.dot(obj.AP.Direction)) < 0.8 else Y
        Vertical = obj.AP.Direction.cross(Direction).normalize()
        Horizontal = Vertical.cross(obj.AP.Direction).normalize()
        point = obj.selectedPoint
        d = point.distanceToPlane(obj.p1, obj.Direction)
        if obj.circumferenceBool:
            P3 = point + obj.Direction * (-d)
            d2 = (P3 - obj.p1) * Vertical
            P2 = obj.p1 + Vertical * (d2*3/4)
        else:
            P2 = obj.p1 + obj.Direction * (d*3/4)
            P3 = point
        points = [obj.p1, P2, P3]
        segments = [0,1,2]
        existGT = True
        if obj.GT <> []:
            points, segments = getPointsToPlotGT(obj, points, segments, Vertical, Horizontal)
        else:
            existGT = False
        if obj.DF <> None:
```

```
            points, segments = getPointsToPlotDF(obj, existGT, points, segments, Vertical, Horizontal)
        segments = segments + []
    return points, segments

def getPointsToPlotGT(obj, points, segments, Vertical, Horizontal):
    newPoints = points
    newSegments = segments
    if obj.ViewObject.LineScale > 0:
        sizeOfLine = obj.ViewObject.LineScale
    else:
        sizeOfLine = 1.0
    for i in range(len(obj.GT)):
        d = len(newPoints)
        if points[2].x < points[0].x:
            P0 = newPoints[-1] + Vertical * (sizeOfLine) if i == 0 else FreeCAD.Vector(newPoints[-2])
        else:
            P0 = newPoints[-1] + Vertical * (sizeOfLine) if i == 0 else FreeCAD.Vector(newPoints[-1])
        P1 = P0 + Vertical * (-sizeOfLine*2)
        P2 = P0 + Horizontal * (sizeOfLine*2)
        P3 = P1 + Horizontal * (sizeOfLine*2)
        lengthToleranceValue = len(stringencodecoin(displayExternal(obj.GT[i].ToleranceValue, obj.ViewObject.Decimals, 'Length',
obj.ViewObject.ShowUnit)))
        if obj.GT[i].FeatureControlFrameIcon <> '':
            lengthToleranceValue += 2
        if obj.GT[i].Circumference:
            lengthToleranceValue += 2
        P4 = P2 + Horizontal * (sizeOfLine*lengthToleranceValue)
        P5 = P3 + Horizontal * (sizeOfLine*lengthToleranceValue)
        if obj.GT[i].DS == None or obj.GT[i].DS.Primary == None:
            newPoints = newPoints + [P0, P2, P3, P4, P5, P1]
            newSegments = newSegments + [-1, 0+d, 3+d, 4+d, 5+d, 0+d, -1, 1+d, 2+d]
            if points[2].x < points[0].x:
                displacement = newPoints[-3].x - newPoints[-6].x
                for i in range(len(newPoints)-6, len(newPoints)):
                    newPoints[i].x-=displacement
        else:
            P6 = P4 + Horizontal * (sizeOfLine*2)
            P7 = P5 + Horizontal * (sizeOfLine*2)
            if obj.GT[i].DS.Secondary <> None:
                P8 = P6 + Horizontal * (sizeOfLine*2)
                P9 = P7 + Horizontal * (sizeOfLine*2)
                if obj.GT[i].DS.Tertiary <> None:
                    P10 = P8 + Horizontal * (sizeOfLine*2)
                    P11 = P9 + Horizontal * (sizeOfLine*2)
                    newPoints = newPoints + [P0, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P1]
                    newSegments = newSegments + [-1, 0+d, 9+d, 10+d, 11+d, 0+d, -1, 1+d, 2+d, -1, 3+d, 4+d, -1, 5+d, 6+d, -1, 7+d, 8+d]
                    if points[2].x < points[0].x:
                        displacement = newPoints[-3].x - newPoints[-12].x
                        for i in range(len(newPoints)-12, len(newPoints)):
                            newPoints[i].x-=displacement
                else:
                    newPoints = newPoints + [P0, P2, P3, P4, P5, P6, P7, P8, P9, P1]
                    newSegments = newSegments + [-1, 0+d, 7+d, 8+d, 9+d, 0+d, -1, 1+d, 2+d, -1, 3+d, 4+d, -1, 5+d, 6+d]
                    if points[2].x < points[0].x:
                        displacement = newPoints[-3].x - newPoints[-10].x
                        for i in range(len(newPoints)-10, len(newPoints)):
                            newPoints[i].x-=displacement
            else:
                newPoints = newPoints + [P0, P2, P3, P4, P5, P6, P7, P1]
                newSegments = newSegments + [-1, 0+d, 5+d, 6+d, 7+d, 0+d, -1, 1+d, 2+d, -1, 3+d, 4+d]
                if points[2].x < points[0].x:
                    displacement = newPoints[-3].x - newPoints[-8].x
                    for i in range(len(newPoints)-8, len(newPoints)):
                        newPoints[i].x-=displacement
    return newPoints, newSegments

def getPointsToPlotDF(obj, existGT, points, segments, Vertical, Horizontal):
    d = len(points)
    newPoints = points
    newSegments = segments
    if obj.ViewObject.LineScale > 0:
        sizeOfLine = obj.ViewObject.LineScale
    else:
        sizeOfLine = 1.0
    if not existGT:
        P0 = points[-1] + Vertical * (sizeOfLine)
        P1 = P0 + Horizontal * (sizeOfLine*2)
        P2 = P1 + Vertical * (-sizeOfLine*2)
        P3 = P2 + Horizontal * (-sizeOfLine*2)
        newPoints = newPoints + [P0, P1, P2, P3]
        newSegments = newSegments + [-1, 0+d, 1+d, 2+d, 3+d, 0+d]
```

```
        if points[2].x < points[0].x:
            displacement = newPoints[-2].x - newPoints[-1].x
            for i in range(len(newPoints)-4, len(newPoints)):
                newPoints[i].x-=displacement
    d=len(newPoints)
    P0 = newPoints[-1] + Horizontal * (sizeOfLine/2)
    P1 = P0 + Horizontal * (sizeOfLine)
    h = math.sqrt(sizeOfLine*sizeOfLine+(sizeOfLine/2)*(sizeOfLine/2))
    PAux = newPoints[-1] + Horizontal * (sizeOfLine)
    P2 = PAux + Vertical * (-h)
    P3 = PAux + Vertical * (-sizeOfLine*3)
    P4 = P3 + Horizontal * (sizeOfLine)
    P5 = P4 + Vertical * (-sizeOfLine*2)
    P6 = P5 + Horizontal * (-sizeOfLine*2)
    P7 = P6 + Vertical * (sizeOfLine*2)
    newPoints = newPoints + [P0, P1, P2, P3, P4, P5, P6, P7]
    newSegments = newSegments + [-1, 0+d, 2+d, -1, 1+d, 2+d, 3+d, 4+d, 5+d, 6+d, 7+d, 3+d]
    return newPoints, newSegments

def plotStrings(self, fp, points):
    import DraftGeomUtils
    if fp.ViewObject.LineScale > 0:
        sizeOfLine = fp.ViewObject.LineScale
    else:
        sizeOfLine = 1.0
    X = FreeCAD.Vector(1.0,0.0,0.0)
    Y = FreeCAD.Vector(0.0,1.0,0.0)
    Direction = X if abs(X.dot(fp.AP.Direction)) < 0.8 else Y
    Vertical = fp.AP.Direction.cross(Direction).normalize()
    Horizontal = Vertical.cross(fp.AP.Direction).normalize()
    index = 0
    indexIcon = 0
    displacement = 0
    if fp.GT <> []:
        for i in range(len(fp.GT)):
            distance = 0
            # posToleranceValue
            v = (points[7+displacement] - points[5+displacement])
            if v.x <> 0:
                distance = (v.x)/2
            elif v.y <> 0:
                distance = (v.y)/2
            else:
                distance = (v.z)/2
            if fp.GT[i].FeatureControlFrameIcon <> '':
                distance -= sizeOfLine
            if fp.GT[i].Circumference:
                distance += sizeOfLine
            centerPoint = points[5+displacement] + Horizontal * (distance)
            posToleranceValue = centerPoint + Vertical * (sizeOfLine/2)
            # posCharacteristic
            auxPoint = points[3+displacement] + Vertical * (-sizeOfLine*2)

self.points[indexIcon].point.setValues([[auxPoint.x,auxPoint.y,auxPoint.z],[points[5+displacement].x,points[5+displacement].y,points[5+dis
placement].z],[points[4+displacement].x,points[4+displacement].y,points[4+displacement].z],[points[3+displacement].x,points[3+displacem
ent].y,points[3+displacement].z]])
            self.face[indexIcon].numVertices = 4
            s = 1/(sizeOfLine*2)
            dS = FreeCAD.Vector(Horizontal) * s
            dT = FreeCAD.Vector(Vertical) * s
            self.svgPos[indexIcon].directionS.setValue(dS.x, dS.y, dS.z)
            self.svgPos[indexIcon].directionT.setValue(dT.x, dT.y, dT.z)
            displacementH = ((Horizontal*auxPoint)%(sizeOfLine*2))/(sizeOfLine*2)
            displacementV = ((Vertical*auxPoint)%(sizeOfLine*2))/(sizeOfLine*2)
            self.textureTransform[indexIcon].translation.setValue(-displacementH,-displacementV)
            filename = fp.GT[i].CharacteristicIcon
            filename = filename.replace(':/dd/icons', iconPath)
            self.svg[indexIcon].filename = str(filename)
            indexIcon+=1
            # posFeactureControlFrame
            if fp.GT[i].FeatureControlFrameIcon <> '':
                auxPoint1 = points[7+displacement] + Horizontal * (-sizeOfLine*2)
                auxPoint2 = auxPoint1 + Vertical * (sizeOfLine*2)

self.points[indexIcon].point.setValues([[auxPoint1.x,auxPoint1.y,auxPoint1.z],[points[7+displacement].x,points[7+displacement].y,points[7
+displacement].z],[points[6+displacement].x,points[6+displacement].y,points[6+displacement].z],[auxPoint2.x,auxPoint2.y,auxPoint2.z]])
                self.face[indexIcon].numVertices = 4
                self.svgPos[indexIcon].directionS.setValue(dS.x, dS.y, dS.z)
                self.svgPos[indexIcon].directionT.setValue(dT.x, dT.y, dT.z)
                displacementH = ((Horizontal*auxPoint1)%(sizeOfLine*2))/(sizeOfLine*2)
                displacementV = ((Vertical*auxPoint1)%(sizeOfLine*2))/(sizeOfLine*2)
```

```
            self.textureTransform[indexIcon].translation.setValue(-displacementH,-displacementV)
            filename = fp.GT[i].FeatureControlFrameIcon
            filename = filename.replace(':/dd/icons', iconPath)
            self.svg[indexIcon].filename = str(filename)
            indexIcon+=1
        # posDiameter
        if fp.GT[i].Circumference:
            auxPoint1 = points[5+displacement] + Horizontal * (sizeOfLine*2)
            auxPoint2 = auxPoint1 + Vertical * (sizeOfLine*2)

self.points[indexIcon].point.setValues([[points[5+displacement].x,points[5+displacement].y,points[5+displacement].z],[auxPoint1.x,auxPoin
t1.y,auxPoint1.z],[auxPoint2.x,auxPoint2.y,auxPoint2.z],[points[4+displacement].x,points[4+displacement].y,points[4+displacement].z]])
            self.face[indexIcon].numVertices = 4
            self.svgPos[indexIcon].directionS.setValue(dS.x, dS.y, dS.z)
            self.svgPos[indexIcon].directionT.setValue(dT.x, dT.y, dT.z)
            displacementH = ((Horizontal*points[5+displacement])%(sizeOfLine*2))/(sizeOfLine*2)
            displacementV = ((Vertical*points[5+displacement])%(sizeOfLine*2))/(sizeOfLine*2)
            self.textureTransform[indexIcon].translation.setValue(-displacementH,-displacementV)
            filename = iconPath + '/diameter.svg'
            self.svg[indexIcon].filename = str(filename)
            indexIcon+=1

        self.textGT[index].string = self.textGT3d[index].string = stringencodecoin(displayExternal(fp.GT[i].ToleranceValue,
fp.ViewObject.Decimals, 'Length', fp.ViewObject.ShowUnit))
        self.textGTpos[index].translation.setValue([posToleranceValue.x, posToleranceValue.y, posToleranceValue.z])
        self.textGT[index].justification = coin.SoAsciiText.CENTER
        index+=1
        displacement+=6
        if fp.GT[i].DS <> None and fp.GT[i].DS.Primary <> None:
            if fp.GT[i].FeatureControlFrameIcon <> '':
                distance += (sizeOfLine*2)
            if fp.GT[i].Circumference:
                distance -= (sizeOfLine*2)
            posPrimary = posToleranceValue + Horizontal * (distance+sizeOfLine)
            self.textGT[index].string = self.textGT3d[index].string = str(fp.GT[i].DS.Primary.Label)
            self.textGTpos[index].translation.setValue([posPrimary.x, posPrimary.y, posPrimary.z])
            self.textGT[index].justification = coin.SoAsciiText.CENTER
            index+=1
            displacement+=2
            if fp.GT[i].DS.Secondary <> None:
                posSecondary = posPrimary + Horizontal * (sizeOfLine*2)
                self.textGT[index].string = self.textGT3d[index].string = str(fp.GT[i].DS.Secondary.Label)
                self.textGTpos[index].translation.setValue([posSecondary.x, posSecondary.y, posSecondary.z])
                self.textGT[index].justification = coin.SoAsciiText.CENTER
                index+=1
                displacement+=2
                if fp.GT[i].DS.Tertiary <> None:
                    posTertiary = posSecondary + Horizontal * (sizeOfLine*2)
                    self.textGT[index].string = self.textGT3d[index].string = str(fp.GT[i].DS.Tertiary.Label)
                    self.textGTpos[index].translation.setValue([posTertiary.x, posTertiary.y, posTertiary.z])
                    self.textGT[index].justification = coin.SoAsciiText.CENTER
                    index+=1
                    displacement+=2
    if fp.circumferenceBool and True in [l.Circumference for l in fp.GT]:
        # posDiameterTolerance
        auxPoint1 = FreeCAD.Vector(points[4])
        auxPoint2 = auxPoint1 + Horizontal * (sizeOfLine*2)
        auxPoint3 = auxPoint2 + Vertical * (sizeOfLine*2)
        auxPoint4 = auxPoint1 + Vertical * (sizeOfLine*2)

self.points[indexIcon].point.setValues([[auxPoint1.x,auxPoint1.y,auxPoint1.z],[auxPoint2.x,auxPoint2.y,auxPoint2.z],[auxPoint3.x,auxPoint
3.y,auxPoint3.z],[auxPoint4.x,auxPoint4.y,auxPoint4.z]])
        self.face[indexIcon].numVertices = 4
        self.svgPos[indexIcon].directionS.setValue(dS.x, dS.y, dS.z)
        self.svgPos[indexIcon].directionT.setValue(dT.x, dT.y, dT.z)
        displacementH = ((Horizontal*auxPoint1)%(sizeOfLine*2))/(sizeOfLine*2)
        displacementV = ((Vertical*auxPoint1)%(sizeOfLine*2))/(sizeOfLine*2)
        self.textureTransform[indexIcon].translation.setValue(-displacementH,-displacementV)
        filename = iconPath + '/diameter.svg'
        self.svg[indexIcon].filename = str(filename)
        indexIcon+=1
        posDiameterTolerance = auxPoint2 + Vertical * (sizeOfLine/2)
        self.textGT[index].justification = coin.SoAsciiText.LEFT
        self.textGTpos[index].translation.setValue([posDiameterTolerance.x, posDiameterTolerance.y, posDiameterTolerance.z])
        if fp.toleranceSelectBool:
            text = stringencodecoin(displayExternal(fp.diameter, fp.ViewObject.Decimals, 'Length', fp.ViewObject.ShowUnit) +
stringplusminus() + displayExternal(fp.toleranceDiameter, fp.ViewObject.Decimals, 'Length', fp.ViewObject.ShowUnit))
        else:
            text = stringencodecoin(displayExternal(fp.lowLimit, fp.ViewObject.Decimals, 'Length', fp.ViewObject.ShowUnit) + ' - ' +
displayExternal(fp.highLimit, fp.ViewObject.Decimals, 'Length', fp.ViewObject.ShowUnit))
        self.textGT[index].string = self.textGT3d[index].string = text
```

```
            index+=1
        for i in range(index):
            try:
                DirectionAux = FreeCAD.Vector(fp.AP.Direction)
                DirectionAux.x = abs(DirectionAux.x)
                DirectionAux.y = abs(DirectionAux.y)
                DirectionAux.z = abs(DirectionAux.z)
                rotation=(DraftGeomUtils.getRotation(DirectionAux)).Q
                self.textGTpos[i].rotation.setValue(rotation)
            except:
                pass
        for i in range(index,len(self.textGT)):
            if str(self.textGT[i].string) <> "":
                self.textGT[i].string = self.textGT3d[i].string = ""
            else:
                break
        for i in range(indexIcon,len(self.svg)):
            if str(self.face[i].numVertices) <> 0:
                self.face[i].numVertices = 0
                self.svg[i].filename = ""
    else:
        for i in range(len(self.textGT)):
            if str(self.textGT[i].string) <> "" or str(self.svg[i].filename) <> "":
                self.textGT[i].string = self.textGT3d[i].string = ""
                self.face[i].numVertices = 0
                self.svg[i].filename = ""
            else:
                break
    if fp.DF <> None:
        self.textDF.string = self.textDF3d.string = str(fp.DF.Label)
        distance = 0
        v = (points[-3] - points[-2])
        if v.x <> 0:
            distance = (v.x)/2
        elif v.y <> 0:
            distance = (v.y)/2
        else:
            distance = (v.z)/2
        centerPoint = points[-2] + Horizontal * (distance)
        centerPoint = centerPoint + Vertical * (sizeOfLine/2)
        self.textDFpos.translation.setValue([centerPoint.x, centerPoint.y, centerPoint.z])
        try:
            DirectionAux = FreeCAD.Vector(fp.AP.Direction)
            DirectionAux.x = abs(DirectionAux.x)
            DirectionAux.y = abs(DirectionAux.y)
            DirectionAux.z = abs(DirectionAux.z)
            rotation=(DraftGeomUtils.getRotation(DirectionAux)).Q
            self.textDFpos.rotation.setValue(rotation)
        except:
            pass
    else:
        self.textDF.string = self.textDF3d.string = ""
    if fp.GT <> [] or fp.DF <> None:
        if len(fp.faces) > 1:
            # posNumFaces
            centerPoint = points[3] + Horizontal * (sizeOfLine)
            posNumFaces = centerPoint + Vertical * (sizeOfLine/2)
            self.textGT[index].string = self.textGT3d[index].string = (str(len(fp.faces))+'x')
            self.textGTpos[index].translation.setValue([posNumFaces.x, posNumFaces.y, posNumFaces.z])
            self.textGT[index].justification = coin.SoAsciiText.CENTER
            try:
                DirectionAux = FreeCAD.Vector(fp.AP.Direction)
                DirectionAux.x = abs(DirectionAux.x)
                DirectionAux.y = abs(DirectionAux.y)
                DirectionAux.z = abs(DirectionAux.z)
                rotation=(DraftGeomUtils.getRotation(DirectionAux)).Q
                self.textGTpos[index].rotation.setValue(rotation)
            except:
                pass
            index+=1

#---------------------------------------------------------------------------
# UNITS handling
#---------------------------------------------------------------------------

def getDefaultUnit(dim):
    '''return default Unit of Measure for a Dimension based on user preference
    Units Schema'''
    # only Length and Angle so far
    from FreeCAD import Units
    if dim == 'Length':
```

```
        qty = FreeCAD.Units.Quantity(1.0,FreeCAD.Units.Length)
        UOM = qty.getUserPreferred()[2]
    elif dim == 'Angle':
        qty = FreeCAD.Units.Quantity(1.0,FreeCAD.Units.Angle)
        UOM = qty.getUserPreferred()[2]
    else:
        UOM = "xx"
    return UOM

def makeFormatSpec(decimals=4,dim='Length'):
    ''' return a % format spec with specified decimals for a specified
    dimension based on on user preference Units Schema'''
    if dim == 'Length':
        fmtSpec = "%." + str(decimals) + "f "+ getDefaultUnit('Length')
    elif dim == 'Angle':
        fmtSpec = "%." + str(decimals) + "f "+ getDefaultUnit('Angle')
    else:
        fmtSpec = "%." + str(decimals) + "f " + "??"
    return fmtSpec

def displayExternal(internValue,decimals=4,dim='Length',showUnit=True):
    '''return an internal value (ie mm) Length or Angle converted for display according
    to Units Schema in use.'''
    from FreeCAD import Units

    if dim == 'Length':
        qty = FreeCAD.Units.Quantity(internValue,FreeCAD.Units.Length)
        pref = qty.getUserPreferred()
        conversion = pref[1]
        uom = pref[2]
    elif dim == 'Angle':
        qty = FreeCAD.Units.Quantity(internValue,FreeCAD.Units.Angle)
        pref=qty.getUserPreferred()
        conversion = pref[1]
        uom = pref[2]
    else:
        conversion = 1.0
        uom = "??"
    if not showUnit:
        uom = ""
    fmt = "{0:."+ str(decimals) + "f} "+ uom
    displayExt = fmt.format(float(internValue) / float(conversion))
    displayExt = displayExt.replace(".",QtCore.QLocale().decimalPoint())
    return displayExt

#---------------------------------------------------------------------------
# Python Features definitions
#---------------------------------------------------------------------------

    #---------------------------------------------------------------------------
    # Base class for GDT objects
    #---------------------------------------------------------------------------

class _GDTObject:
    "The base class for GDT objects"
    def __init__(self,obj,tp="Unknown"):
        '''Add some custom properties to our GDT feature'''
        obj.Proxy = self
        self.Type = tp

    def __getstate__(self):
        return self.Type

    def __setstate__(self,state):
        if state:
            self.Type = state

    def execute(self,obj):
        '''Do something when doing a recomputation, this method is mandatory'''
        pass

    def onChanged(self, vobj, prop):
        '''Do something when a property has changed'''
        pass

class _ViewProviderGDT:
    "The base class for GDT Viewproviders"

    def __init__(self, vobj):
        '''Set this object to the proxy object of the actual view provider'''
        vobj.Proxy = self
```

```
        self.Object = vobj.Object

    def __getstate__(self):
        return None

    def __setstate__(self, state):
        return None

    def attach(self,vobj):
        '''Setup the scene sub-graph of the view provider, this method is mandatory'''
        self.Object = vobj.Object
        return

    def updateData(self, obj, prop):
        '''If a property of the handled feature has changed we have the chance to handle this here'''
        # fp is the handled feature, prop is the name of the property that has changed
        return

    def getDisplayModes(self, vobj):
        '''Return a list of display modes.'''
        modes=[]
        return modes

    def setDisplayMode(self, mode):
        '''Map the display mode defined in attach with those defined in getDisplayModes.\
                Since they have the same names nothing needs to be done. This method is optional'''
        return mode

    def onChanged(self, vobj, prop):
        '''Here we can do something when a single property got changed'''
        return

    def execute(self,vobj):
        return

    def getIcon(self):
        '''Return the icon in XPM format which will appear in the tree view. This method is\
                optional and if not defined a default icon is shown.'''
        return(":/dd/icons/GDT.svg")

    #-------------------------------------------------------------------------
    # Annotation Plane
    #-------------------------------------------------------------------------

class _AnnotationPlane(_GDTObject):
    "The GDT AnnotationPlane object"
    def __init__(self, obj):
        _GDTObject.__init__(self,obj,"AnnotationPlane")
        obj.addProperty("App::PropertyFloat","Offset","GDT","The offset value to aply in this annotation plane")
        obj.addProperty("App::PropertyLinkSub","faces","GDT","Linked face of the object").faces = (getSelectionEx()[0].Object,
getSelectionEx()[0].SubElementNames[0])
        obj.addProperty("App::PropertyVectorDistance","p1","GDT","Center point of Grid").p1 =
obj.faces[0].Shape.getElement(obj.faces[1][0]).CenterOfMass
        obj.addProperty("App::PropertyVector","Direction","GDT","The normal direction of this annotation plane").Direction =
obj.faces[0].Shape.getElement(obj.faces[1][0]).normalAt(0,0)
        obj.addProperty("App::PropertyVectorDistance","PointWithOffset","GDT","Center point of Grid with offset applied")

    def onChanged(self,vobj,prop):
        if hasattr(vobj,"PointWithOffset"):
            vobj.setEditorMode('PointWithOffset',1)

    def execute(self, fp):
        """Print a short message when doing a recomputation, this method is mandatory" '"
        fp.p1 = fp.faces[0].Shape.getElement(fp.faces[1][0]).CenterOfMass
        fp.Direction = fp.faces[0].Shape.getElement(fp.faces[1][0]).normalAt(0,0)

class _ViewProviderAnnotationPlane(_ViewProviderGDT):
    "A View Provider for the GDT AnnotationPlane object"
    def __init__(self, obj):
        _ViewProviderGDT.__init__(self,obj)

    def updateData(self, obj, prop):
        "called when the base object is changed"
        if prop in ["Point","Direction","Offset"]:
            obj.PointWithOffset = obj.p1 + obj.Direction * obj.Offset

    def doubleClicked(self,obj):
        showGrid()
        FreeCAD.DraftWorkingPlane.alignToPointAndAxis(self.Object.PointWithOffset, self.Object.Direction, 0)
        FreeCADGui.Snapper.grid.set()
        FreeCAD.ActiveDocument.recompute()
```

```
    def getIcon(self):
        return(":/dd/icons/annotationPlane.svg")

def makeAnnotationPlane(Name, Offset):
    ''' Explanation
    '''
    if len(getAllAnnotationPlaneObjects()) == 0:
        group = FreeCAD.ActiveDocument.addObject("App::DocumentObjectGroupPython", "GDT")
        _GDTObject(group)
        _ViewProviderGDT(group.ViewObject)
    else:
        group = FreeCAD.ActiveDocument.getObject("GDT")

    obj = FreeCAD.ActiveDocument.addObject("App::FeaturePython","AnnotationPlane")
    _AnnotationPlane(obj)
    if gui:
        _ViewProviderAnnotationPlane(obj.ViewObject)
    obj.Label = Name
    obj.Offset = Offset
    group.addObject(obj)
    hideGrid()
    for l in getAllAnnotationObjects():
        l.touch()
    FreeCAD.ActiveDocument.recompute()
    return obj

    #---------------------------------------------------------------------------
    # Datum Feature
    #---------------------------------------------------------------------------

class _DatumFeature(_GDTObject):
    "The GDT DatumFeature object"
    def __init__(self, obj):
        _GDTObject.__init__(self,obj,"DatumFeature")

    def execute(self,obj):
        '''Do something when doing a recomputation, this method is mandatory'''
        pass

class _ViewProviderDatumFeature(_ViewProviderGDT):
    "A View Provider for the GDT DatumFeature object"
    def __init__(self, obj):
        _ViewProviderGDT.__init__(self,obj)

    def getIcon(self):
        return(":/dd/icons/datumFeature.svg")

def makeDatumFeature(Name, ContainerOfData):
    ''' Explanation
    '''
    obj = FreeCAD.ActiveDocument.addObject("App::FeaturePython","DatumFeature")
    _DatumFeature(obj)
    if gui:
        _ViewProviderDatumFeature(obj.ViewObject)
    obj.Label = Name
    group = FreeCAD.ActiveDocument.getObject("GDT")
    group.addObject(obj)
    AnnotationObj = getAnnotationObj(ContainerOfData)
    if AnnotationObj == None:
        makeAnnotation(ContainerOfData.faces, ContainerOfData.annotationPlane, DF=obj, GT=[])
    else:
        faces = AnnotationObj.faces
        AP = AnnotationObj.AP
        GT = AnnotationObj.GT
        diameter = AnnotationObj.diameter
        toleranceSelect = AnnotationObj.toleranceSelectBool
        toleranceDiameter = AnnotationObj.toleranceDiameter
        lowLimit = AnnotationObj.lowLimit
        highLimit = AnnotationObj.highLimit
        group = makeAnnotation(faces, AP, DF=obj, GT=GT, modify = True, Object = AnnotationObj, diameter=diameter,
toleranceSelect=toleranceSelect, toleranceDiameter=toleranceDiameter, lowLimit=lowLimit, highLimit=highLimit)
        group.addObject(obj)
    for l in getAllAnnotationObjects():
        l.touch()
    FreeCAD.ActiveDocument.recompute()
    return obj

    #---------------------------------------------------------------------------
    # Datum System
    #---------------------------------------------------------------------------
```

```
class _DatumSystem(_GDTObject):
    "The GDT DatumSystem object"
    def __init__(self, obj):
        _GDTObject.__init__(self,obj,"DatumSystem")
        obj.addProperty("App::PropertyLink","Primary","GDT","Primary datum feature used")
        obj.addProperty("App::PropertyLink","Secondary","GDT","Secondary datum feature used")
        obj.addProperty("App::PropertyLink","Tertiary","GDT","Tertiary datum feature used")

class _ViewProviderDatumSystem(_ViewProviderGDT):
    "A View Provider for the GDT DatumSystem object"
    def __init__(self, obj):
        _ViewProviderGDT.__init__(self,obj)

    def updateData(self, obj, prop):
        "called when the base object is changed"
        if prop in ["Primary","Secondary","Tertiary"]:
            textName = obj.Label.split(":")[0]
            if obj.Primary <> None:
                textName+=': '+obj.Primary.Label
                if obj.Secondary <> None:
                    textName+=' | '+obj.Secondary.Label
                    if obj.Tertiary <> None:
                        textName+=' | '+obj.Tertiary.Label
            obj.Label = textName

    def getIcon(self):
        return(":/dd/icons/datumSystem.svg")

def makeDatumSystem(Name, Primary, Secondary=None, Tertiary=None):
    ''' Explanation
    '''
    obj = FreeCAD.ActiveDocument.addObject("App::FeaturePython","DatumSystem")
    _DatumSystem(obj)
    if gui:
        _ViewProviderDatumSystem(obj.ViewObject)
    obj.Label = Name
    obj.Primary = Primary
    obj.Secondary = Secondary
    obj.Tertiary = Tertiary
    group = FreeCAD.ActiveDocument.getObject("GDT")
    group.addObject(obj)
    for l in getAllAnnotationObjects():
        l.touch()
    FreeCAD.ActiveDocument.recompute()
    return obj

    #---------------------------------------------------------------------------
    # Geometric Tolerance
    #---------------------------------------------------------------------------

class _GeometricTolerance(_GDTObject):
    "The GDT GeometricTolerance object"
    def __init__(self, obj):
        _GDTObject.__init__(self,obj,"GeometricTolerance")
        obj.addProperty("App::PropertyString","Characteristic","GDT","Characteristic of the geometric tolerance")
        obj.addProperty("App::PropertyString","CharacteristicIcon","GDT","Characteristic icon path of the geometric tolerance")
        obj.addProperty("App::PropertyBool","Circumference","GDT","Indicates whether the tolerance applies to a given diameter")
        obj.addProperty("App::PropertyFloat","ToleranceValue","GDT","Tolerance value of the geometric tolerance")
        obj.addProperty("App::PropertyString","FeatureControlFrame","GDT","Feature control frame of the geometric tolerance")
        obj.addProperty("App::PropertyString","FeatureControlFrameIcon","GDT","Feature control frame icon path of the geometric
tolerance")
        obj.addProperty("App::PropertyLink","DS","GDT","Datum system used")

    def onChanged(self,vobj,prop):
        "Do something when a property has changed"
        if hasattr(vobj,"CharacteristicIcon"):
            vobj.setEditorMode('CharacteristicIcon',2)
        if hasattr(vobj,"FeatureControlFrameIcon"):
            vobj.setEditorMode('FeatureControlFrameIcon',2)

class _ViewProviderGeometricTolerance(_ViewProviderGDT):
    "A View Provider for the GDT GeometricTolerance object"
    def __init__(self, obj):
        _ViewProviderGDT.__init__(self,obj)

    def getIcon(self):
        icon = self.Object.CharacteristicIcon
        return icon

def makeGeometricTolerance(Name, ContainerOfData):
```

```
    ''' Explanation
    '''
    obj = FreeCAD.ActiveDocument.addObject("App::FeaturePython","GeometricTolerance")
    _GeometricTolerance(obj)
    if gui:
        _ViewProviderGeometricTolerance(obj.ViewObject)
    obj.Label = Name
    obj.Characteristic = ContainerOfData.characteristic.Label
    obj.CharacteristicIcon = ContainerOfData.characteristic.Icon
    obj.Circumference = ContainerOfData.circumference
    obj.ToleranceValue = ContainerOfData.toleranceValue
    obj.FeatureControlFrame = ContainerOfData.featureControlFrame.toolTip
    obj.FeatureControlFrameIcon = ContainerOfData.featureControlFrame.Icon
    obj.DS = ContainerOfData.datumSystem
    group = FreeCAD.ActiveDocument.getObject("GDT")
    group.addObject(obj)
    AnnotationObj = getAnnotationObj(ContainerOfData)
    if AnnotationObj == None:
        makeAnnotation(ContainerOfData.faces, ContainerOfData.annotationPlane, DF=None, GT=obj, diameter=ContainerOfData.diameter,
toleranceSelect=ContainerOfData.toleranceSelect, toleranceDiameter=ContainerOfData.toleranceDiameter,
lowLimit=ContainerOfData.lowLimit, highLimit=ContainerOfData.highLimit)
    else:
        gt=AnnotationObj.GT
        gt.append(obj)
        faces = AnnotationObj.faces
        AP = AnnotationObj.AP
        DF = AnnotationObj.DF
        if ContainerOfData.circumference:
            diameter = ContainerOfData.diameter
            toleranceSelect = ContainerOfData.toleranceSelect
            toleranceDiameter = ContainerOfData.toleranceDiameter
            lowLimit = ContainerOfData.lowLimit
            highLimit = ContainerOfData.highLimit
        else:
            diameter = AnnotationObj.diameter
            toleranceSelect = AnnotationObj.toleranceSelectBool
            toleranceDiameter = AnnotationObj.toleranceDiameter
            lowLimit = AnnotationObj.lowLimit
            highLimit = AnnotationObj.highLimit
        group = makeAnnotation(faces, AP, DF=DF, GT=gt, modify = True, Object = AnnotationObj, diameter=diameter,
toleranceSelect=toleranceSelect, toleranceDiameter=toleranceDiameter, lowLimit=lowLimit, highLimit=highLimit)
        group.addObject(obj)
    for l in getAllAnnotationObjects():
        l.touch()
    FreeCAD.ActiveDocument.recompute()
    return obj

    #---------------------------------------------------------------------
    # Annotation
    #---------------------------------------------------------------------

class _Annotation(_GDTObject):
    "The GDT Annotation object"
    def __init__(self, obj):
        _GDTObject.__init__(self,obj,"Annotation")
        obj.addProperty("App::PropertyLinkSubList","faces","GDT","Linked faces of the object")
        obj.addProperty("App::PropertyLink","AP","GDT","Annotation plane used")
        obj.addProperty("App::PropertyLink","DF","GDT","Text").DF=None
        obj.addProperty("App::PropertyLinkList","GT","GDT","Text").GT=[]
        obj.addProperty("App::PropertyVectorDistance","p1","GDT","Start point")
        obj.addProperty("App::PropertyVector","Direction","GDT","The normal direction of your annotation plane")
        obj.addProperty("App::PropertyVector","selectedPoint","GDT","Selected point to where plot the annotation")
        obj.addProperty("App::PropertyBool","spBool","GDT","Boolean to confirm that a selected point exists").spBool = False
        obj.addProperty("App::PropertyBool","circumferenceBool","GDT","Boolean to determine if this annotation is over a
circumference").circumferenceBool = False
        obj.addProperty("App::PropertyFloat","diameter","GDT","Diameter")
        obj.addProperty("App::PropertyBool","toleranceSelectBool","GDT","Determinates if use plus-minus or low and high
limits").toleranceSelectBool = True
        obj.addProperty("App::PropertyFloat","toleranceDiameter","GDT","Diameter tolerance (Plus-minus)")
        obj.addProperty("App::PropertyFloat","lowLimit","GDT","Low limit diameter tolerance")
        obj.addProperty("App::PropertyFloat","highLimit","GDT","High limit diameter tolerance")

    def onChanged(self,obj,prop):
        if hasattr(obj,"spBool"):
            obj.setEditorMode('spBool',2)
        if hasattr(obj,"diameter"):
            if obj.circumferenceBool:
                obj.setEditorMode('diameter',0)
            else:
                obj.setEditorMode('diameter',2)
        if hasattr(obj,"toleranceDiameter") and hasattr(obj,"toleranceSelectBool"):
```

```
            if obj.circumferenceBool and obj.toleranceSelectBool:
                obj.setEditorMode('toleranceDiameter',0)
            else:
                obj.setEditorMode('toleranceDiameter',2)
        if hasattr(obj,"lowLimit") and hasattr(obj,"toleranceSelectBool"):
            if obj.circumferenceBool and not obj.toleranceSelectBool:
                obj.setEditorMode('lowLimit',0)
            else:
                obj.setEditorMode('lowLimit',2)
        if hasattr(obj,"highLimit") and hasattr(obj,"toleranceSelectBool"):
            if obj.circumferenceBool and not obj.toleranceSelectBool:
                obj.setEditorMode('highLimit',0)
            else:
                obj.setEditorMode('highLimit',2)

    def execute(self, fp):
        """Print a short message when doing a recomputation, this method is mandatory" '"
        # FreeCAD.Console.PrintMessage('Executed\n')
        auxP1 = fp.p1
        if fp.circumferenceBool:
            vertexex = fp.faces[0][0].Shape.getElement(fp.faces[0][1]).Vertexes
            fp.p1 = vertexex[0].Point if vertexex[0].Point.z > vertexex[1].Point.z else vertexex[1].Point
            fp.Direction = fp.AP.Direction
        else:
            fp.p1 = (fp.faces[0][0].Shape.getElement(fp.faces[0][1]).CenterOfMass).projectToPlane(fp.AP.PointWithOffset, fp.AP.Direction)
            fp.Direction = fp.faces[0][0].Shape.getElement(fp.faces[0][1]).normalAt(0,0)
        diff = fp.p1-auxP1
        if fp.spBool:
            fp.selectedPoint = fp.selectedPoint + diff

class _ViewProviderAnnotation(_ViewProviderGDT):
    "A View Provider for the GDT Annotation object"
    def __init__(self, obj):
        obj.addProperty("App::PropertyFloat","LineWidth","GDT","Line width").LineWidth = getLineWidth()
        obj.addProperty("App::PropertyColor","LineColor","GDT","Line color").LineColor = getRGBLine()
        obj.addProperty("App::PropertyFloat","LineScale","GDT","Line scale").LineScale = getParam("lineScale",1.0)
        obj.addProperty("App::PropertyLength","FontSize","GDT","Font size").FontSize = getTextSize()
        obj.addProperty("App::PropertyString","FontName","GDT","Font name").FontName = getTextFamily()
        obj.addProperty("App::PropertyColor","FontColor","GDT","Font color").FontColor = getRGBText()
        obj.addProperty("App::PropertyInteger","Decimals","GDT","The number of decimals to show").Decimals = FreeCAD.ParamGet("User
parameter:BaseApp/Preferences/Units").GetInt("Decimals",2)
        obj.addProperty("App::PropertyBool","ShowUnit","GDT","Show the unit suffix").ShowUnit = getParam("showUnit",True)
        _ViewProviderGDT.__init__(self,obj)

    def attach(self, obj):
        "called on object creation"
        from pivy import coin
        self.node = coin.SoGroup()
        self.node3d = coin.SoGroup()
        self.lineColor = coin.SoBaseColor()
        self.textColor = coin.SoBaseColor()

        self.data = coin.SoCoordinate3()
        self.data.point.isDeleteValuesEnabled()
        self.lines = coin.SoIndexedLineSet()

        selectionNode = coin.SoType.fromName("SoFCSelection").createInstance()
        selectionNode.documentName.setValue(FreeCAD.ActiveDocument.Name)
        selectionNode.objectName.setValue(obj.Object.Name) # here obj is the ViewObject, we need its associated App Object
        selectionNode.subElementName.setValue("Lines")
        selectionNode.addChild(self.lines)

        self.font = coin.SoFont()
        self.font3d = coin.SoFont()
        self.textDF = coin.SoAsciiText()
        self.textDF3d = coin.SoText2()
        self.textDF.string = "" # some versions of coin crash if string is not set
        self.textDF3d.string = ""
        self.textDFpos = coin.SoTransform()
        self.textDF.justification = self.textDF3d.justification = coin.SoAsciiText.CENTER
        labelDF = coin.SoSeparator()
        labelDF.addChild(self.textDFpos)
        labelDF.addChild(self.textColor)
        labelDF.addChild(self.font)
        labelDF.addChild(self.textDF)
        labelDF3d = coin.SoSeparator()
        labelDF3d.addChild(self.textDFpos)
        labelDF3d.addChild(self.textColor)
        labelDF3d.addChild(self.font3d)
        labelDF3d.addChild(self.textDF3d)
```

```
        self.textGT = []
        self.textGT3d = []
        self.textGTpos = []
        self.svg = []
        self.svgPos = []
        self.points = []
        self.face = []
        self.textureTransform = []
        for i in range(20):
            self.textGT.append(coin.SoAsciiText())
            self.textGT3d.append(coin.SoText2())
            self.textGT[i].string = ""
            self.textGT3d[i].string = ""
            self.textGTpos.append(coin.SoTransform())
            self.textGT[i].justification = self.textGT3d[i].justification = coin.SoAsciiText.CENTER
            labelGT = coin.SoSeparator()
            labelGT.addChild(self.textGTpos[i])
            labelGT.addChild(self.textColor)
            labelGT.addChild(self.font)
            labelGT.addChild(self.textGT[i])
            labelGT3d = coin.SoSeparator()
            labelGT3d.addChild(self.textGTpos[i])
            labelGT3d.addChild(self.textColor)
            labelGT3d.addChild(self.font3d)
            labelGT3d.addChild(self.textGT3d[i])
            self.svg.append(coin.SoTexture2())
            self.face.append(coin.SoFaceSet())
            self.textureTransform.append(coin.SoTexture2Transform())
            self.svgPos.append(coin.SoTextureCoordinatePlane())
            self.face[i].numVertices = 0
            self.points.append(coin.SoVRMLCoordinate())
            image = coin.SoSeparator()
            image.addChild(self.svg[i])
            image.addChild(self.textureTransform[i])
            image.addChild(self.svgPos[i])
            image.addChild(self.points[i])
            image.addChild(self.face[i])
            self.node.addChild(labelGT)
            self.node3d.addChild(labelGT3d)
            self.node.addChild(image)
            self.node3d.addChild(image)

        self.drawstyle = coin.SoDrawStyle()
        self.drawstyle.style = coin.SoDrawStyle.LINES

        self.node.addChild(labelDF)
        self.node.addChild(self.drawstyle)
        self.node.addChild(self.lineColor)
        self.node.addChild(self.data)
        self.node.addChild(self.lines)
        self.node.addChild(selectionNode)
        obj.addDisplayMode(self.node,"2D")

        self.node3d.addChild(labelDF3d)
        self.node3d.addChild(self.lineColor)
        self.node3d.addChild(self.data)
        self.node3d.addChild(self.lines)
        self.node3d.addChild(selectionNode)
        obj.addDisplayMode(self.node3d,"3D")
        self.onChanged(obj,"LineColor")
        self.onChanged(obj,"LineWidth")
        self.onChanged(obj,"FontSize")
        self.onChanged(obj,"FontName")
        self.onChanged(obj,"FontColor")

    def updateData(self, fp, prop):
        "If a property of the handled feature has changed we have the chance to handle this here"
        # fp is the handled feature, prop is the name of the property that has changed
        if prop in "selectedPoint" and hasattr(fp.ViewObject,"Decimals") and hasattr(fp.ViewObject,"ShowUnit") and fp.spBool:
            points, segments = getPointsToPlot(fp)
            # print str(points)
            # print str(segments)
            self.data.point.setNum(len(points))
            cnt=0
            for p in points:
                self.data.point.set1Value(cnt,p.x,p.y,p.z)
                cnt=cnt+1
            self.lines.coordIndex.setNum(len(segments))
            self.lines.coordIndex.setValues(0,len(segments),segments)
            plotStrings(self, fp, points)
        if prop in "faces" and fp.faces <> []:
```

```
            fp.circumferenceBool = True if (True in [l.Closed for l in fp.faces[0][0].Shape.getElement(fp.faces[0][1]).Edges] and
len(fp.faces[0][0].Shape.getElement(fp.faces[0][1]).Vertexes) == 2) else False

    def doubleClicked(self,obj):
        try:
            select(self.Object)
        except:
            select(obj.Object)

    def getDisplayModes(self,obj):
        "Return a list of display modes."
        modes=[]
        modes.append("2D")
        modes.append("3D")
        return modes

    def getDefaultDisplayMode(self):
        "Return the name of the default display mode. It must be defined in getDisplayModes."
        return "2D"

    def setDisplayMode(self,mode):
        return mode

    def onChanged(self, vobj, prop):
        "Here we can do something when a single property got changed"
        if (prop == "LineColor") and hasattr(vobj,"LineColor"):
            if hasattr(self,"lineColor"):
                c = vobj.getPropertyByName("LineColor")
                self.lineColor.rgb.setValue(c[0],c[1],c[2])
        elif (prop == "LineWidth") and hasattr(vobj,"LineWidth"):
            if hasattr(self,"drawstyle"):
                w = vobj.getPropertyByName("LineWidth")
                self.drawstyle.lineWidth = w
        elif (prop == "FontColor") and hasattr(vobj,"FontColor"):
            if hasattr(self,"textColor"):
                c = vobj.getPropertyByName("FontColor")
                self.textColor.rgb.setValue(c[0],c[1],c[2])
        elif (prop == "FontSize") and hasattr(vobj,"FontSize"):
            if hasattr(self,"font"):
                if vobj.FontSize.Value > 0:
                    self.font.size = vobj.FontSize.Value
            if hasattr(self,"font3d"):
                if vobj.FontSize.Value > 0:
                    self.font3d.size = vobj.FontSize.Value*100
            vobj.Object.touch()
        elif (prop == "FontName") and hasattr(vobj,"FontName"):
            if hasattr(self,"font") and hasattr(self,"font3d"):
                self.font.name = self.font3d.name = str(vobj.FontName)
                vobj.Object.touch()
        else:
            self.updateData(vobj.Object, "selectedPoint")

    def getIcon(self):
        return(":/dd/icons/annotation.svg")

def makeAnnotation(faces, AP, DF=None, GT=[], modify=False, Object=None, diameter = 0.0, toleranceSelect = True, toleranceDiameter =
0.0, lowLimit = 0.0, highLimit = 0.0):
    ''' Explanation
    '''
    if not modify:
        obj =
FreeCAD.ActiveDocument.addObject("App::DocumentObjectGroupPython",dictionaryAnnotation[len(getAllAnnotationObjects())])
        _Annotation(obj)
        if gui:
            _ViewProviderAnnotation(obj.ViewObject)
        group = FreeCAD.ActiveDocument.getObject("GDT")
        group.addObject(obj)
        obj.faces = faces
        obj.AP = AP
        if obj.circumferenceBool:
            vertexex = obj.faces[0][0].Shape.getElement(obj.faces[0][1]).Vertexes
            index = [l.Point.z for l in vertexex].index(max([l.Point.z for l in vertexex]))
            obj.p1 = vertexex[index].Point
            obj.Direction = obj.AP.Direction
        else:
            obj.p1 = (obj.faces[0][0].Shape.getElement(obj.faces[0][1]).CenterOfMass).projectToPlane(obj.AP.PointWithOffset,
obj.AP.Direction)
            obj.Direction = obj.faces[0][0].Shape.getElement(obj.faces[0][1]).normalAt(0,0)
    else:
        obj = Object
    obj.DF = DF
```

```
    obj.GT = GT
    obj.diameter = diameter
    obj.toleranceSelectBool = toleranceSelect
    if toleranceSelect:
        obj.toleranceDiameter = toleranceDiameter
        obj.lowLimit = 0.0
        obj.highLimit = 0.0
    else:
        obj.toleranceDiameter = 0.0
        obj.lowLimit = lowLimit
        obj.highLimit = highLimit

    def getPoint(point):
        if point:
            obj.spBool = True
            obj.selectedPoint = point
            hideGrid()
            obj.addObject(obj.DF) if obj.DF <> None else obj.addObject(obj.GT[0])
            select(obj)
            for l in getAllAnnotationObjects():
                l.touch()
            FreeCAD.ActiveDocument.recompute()
            return obj
        else:
            if DF:
                FreeCAD.ActiveDocument.removeObject(obj.DF.Name)
                if checkBoxState:
                    FreeCAD.ActiveDocument.removeObject(getAllDatumSystemObjects()[-1].Name)
            else:
                FreeCAD.ActiveDocument.removeObject(obj.GT[-1].Name)
            FreeCAD.ActiveDocument.removeObject(obj.Name)
            hideGrid()
            for l in getAllAnnotationObjects():
                l.touch()
            FreeCAD.ActiveDocument.recompute()
            return None
    if not obj.spBool:
        return FreeCADGui.Snapper.getPoint(callback=getPoint)
    else:
        hideGrid()
        select(obj)
        for l in getAllAnnotationObjects():
            l.touch()
        FreeCAD.ActiveDocument.recompute()
        return obj

    #---------------------------------------------------------------------
    # Other classes
    #---------------------------------------------------------------------

class Characteristics(object):
    def __init__(self, Label, Icon):
        self.Label = Label
        self.Icon = Icon
        self.Proxy = self

def makeCharacteristics(label=None):
    Label = ['Straightness', 'Flatness', 'Circularity', 'Cylindricity', 'Profile of a line', 'Profile of a surface', 'Perpendicularity', 'Angularity',
'Parallelism', 'Symmetry', 'Position', 'Concentricity','Circular run-out', 'Total run-out']
    Icon = [':/dd/icons/Characteristic/straightness.svg', ':/dd/icons/Characteristic/flatness.svg', ':/dd/icons/Characteristic/circularity.svg',
':/dd/icons/Characteristic/cylindricity.svg', ':/dd/icons/Characteristic/profileOfALine.svg', ':/dd/icons/Characteristic/profileOfASurface.svg',
':/dd/icons/Characteristic/perpendicularity.svg', ':/dd/icons/Characteristic/angularity.svg', ':/dd/icons/Characteristic/parallelism.svg',
':/dd/icons/Characteristic/symmetry.svg', ':/dd/icons/Characteristic/position.svg',
':/dd/icons/Characteristic/concentricity.svg',':/dd/icons/Characteristic/circularRunOut.svg', ':/dd/icons/Characteristic/totalRunOut.svg']
    if label == None:
        characteristics = Characteristics(Label, Icon)
        return characteristics
    else:
        index = Label.index(label)
        icon = Icon[index]
        characteristics = Characteristics(label, icon)
        return characteristics

class FeatureControlFrame(object):
    def __init__(self, Label, Icon, toolTip):
        self.Label = Label
        self.Icon = Icon
        self.toolTip = toolTip
        self.Proxy = self

def makeFeatureControlFrame(toolTip=None):
```

```
    Label = ['','','','','','']
    Icon = ['', ':/dd/icons/FeatureControlFrame/freeState.svg', ':/dd/icons/FeatureControlFrame/leastMaterialCondition.svg',
':/dd/icons/FeatureControlFrame/maximumMaterialCondition.svg', ':/dd/icons/FeatureControlFrame/projectedToleranceZone.svg',
':/dd/icons/FeatureControlFrame/regardlessOfFeatureSize.svg', ':/dd/icons/FeatureControlFrame/tangentPlane.svg',
':/dd/icons/FeatureControlFrame/unequalBilateral.svg']
    ToolTip = ['Feature control frame', 'Free state', 'Least material condition', 'Maximum material condition', 'Projected tolerance zone',
'Regardless of feature size', 'Tangent plane', 'Unequal Bilateral']
    if toolTip == None:
        featureControlFrame = FeatureControlFrame(Label, Icon, ToolTip)
        return featureControlFrame
    elif toolTip == '':
        featureControlFrame = FeatureControlFrame(Label[0], Icon[0], '')
        return featureControlFrame
    else:
        index = ToolTip.index(toolTip)
        icon = Icon[index]
        label = Label[index]
        featureControlFrame = FeatureControlFrame(label, icon, toolTip)
        return featureControlFrame

class ContainerOfData(object):
    def __init__(self, faces = []):
        self.faces = faces
        self.diameter = 0.0
        if self.faces <> []:
            self.Direction = self.faces[0][0].Shape.getElement(self.faces[0][1]).normalAt(0,0)
            self.DirectionAxis = self.faces[0][0].Shape.getElement(self.faces[0][1]).Surface.Axis
            self.p1 = self.faces[0][0].Shape.getElement(self.faces[0][1]).CenterOfMass
            try:
                edge = [l.Closed for l in self.faces[0][0].Shape.getElement(self.faces[0][1]).Edges].index(True)
                self.diameter = self.faces[0][0].Shape.getElement(self.faces[0][1]).Edges[edge].Length/pi
            except:
                pass
        self.circumference = False
        self.toleranceSelect = True
        self.toleranceDiameter = 0.0
        self.lowLimit = 0.0
        self.highLimit = 0.0
        self.OffsetValue = 0
        self.textName = ''
        self.textDS = ['','','']
        self.primary = None
        self.secondary = None
        self.tertiary = None
        self.characteristic = None
        self.toleranceValue = 0.0
        self.featureControlFrame = ''
        self.datumSystem = 0
        self.annotationPlane = 0
        self.annotation = None
        self.combo = ['','','','','','']
        self.Proxy = self

#---------------------------------------------------------------------------
# Customized widgets
#---------------------------------------------------------------------------

class GDTWidget:
    def __init__(self):
        self.dialogWidgets = []
        self.ContainerOfData = None

    def activate( self, idGDT=0, dialogTitle='GD&T Widget', dialogIconPath=':/dd/icons/GDT.svg', endFunction=None, dictionary=None):
        self.dialogTitle=dialogTitle
        self.dialogIconPath = dialogIconPath
        self.endFunction = endFunction
        self.dictionary = dictionary
        self.idGDT=idGDT
        self.ContainerOfData = makeContainerOfData()
        extraWidgets = []
        if dictionary <> None:
            extraWidgets.append(textLabelWidget(Text='Name:',Mask='NNNn', Dictionary=self.dictionary)) #http://doc.qt.io/qt-
5/qlineedit.html#inputMask-prop
        else:
            extraWidgets.append(textLabelWidget(Text='Name:',Mask='NNNn'))
        self.taskDialog = GDTDialog( self.dialogTitle, self.dialogIconPath, self.idGDT, extraWidgets + self.dialogWidgets,
self.ContainerOfData)
        FreeCADGui.Control.showDialog( self.taskDialog )

class GDTDialog:
    def __init__(self, title, iconPath, idGDT, dialogWidgets, ContainerOfData):
```

```
        self.initArgs = title, iconPath, idGDT, dialogWidgets, ContainerOfData
        self.createForm()

    def createForm(self):
        title, iconPath, idGDT, dialogWidgets, ContainerOfData = self.initArgs
        self.form = GDTGuiClass( title, idGDT, dialogWidgets, ContainerOfData)
        self.form.setWindowTitle( title )
        self.form.setWindowIcon( QtGui.QIcon( iconPath ) )

    def reject(self): #close button
        hideGrid()
        FreeCAD.ActiveDocument.recompute()
        FreeCADGui.Control.closeDialog()

    def getStandardButtons(self): #http://forum.freecadweb.org/viewtopic.php?f=10&t=11801
        return 0x00200000 #close button

class GDTGuiClass(QtGui.QWidget):

    def __init__(self, title, idGDT, dialogWidgets, ContainerOfData):
        super(GDTGuiClass, self).__init__()
        self.dd_dialogWidgets = dialogWidgets
        self.title = title
        self.idGDT = idGDT
        self.ContainerOfData = ContainerOfData
        self.initUI( self.title , self.idGDT, self.ContainerOfData)

    def initUI(self, title, idGDT, ContainerOfData):
        self.idGDT = idGDT
        self.ContainerOfData = ContainerOfData
        vbox = QtGui.QVBoxLayout()
        for widg in self.dd_dialogWidgets:
            if widg <> None:
                w = widg.generateWidget(self.idGDT,self.ContainerOfData)
                if isinstance(w, QtGui.QLayout):
                    vbox.addLayout( w )
                else:
                    vbox.addWidget( w )
        hbox = QtGui.QHBoxLayout()
        buttonCreate = QtGui.QPushButton(title)
        buttonCreate.setDefault(True)
        buttonCreate.clicked.connect(self.createObject)
        hbox.addStretch(1)
        hbox.addWidget( buttonCreate )
        hbox.addStretch(1)
        vbox.addLayout( hbox )
        self.setLayout(vbox)

    def createObject(self):
        global auxDictionaryDS
        self.textName = self.ContainerOfData.textName.encode('utf-8')
        if self.idGDT == 1:
            obj = makeDatumFeature(self.textName, self.ContainerOfData)
            if checkBoxState:
                makeDatumSystem(auxDictionaryDS[len(getAllDatumSystemObjects())] + ': ' + self.textName, obj, None, None)
        elif self.idGDT == 2:
            separator = ' | '
            if self.ContainerOfData.textDS[0] <> '':
                if self.ContainerOfData.textDS[1] <> '':
                    if self.ContainerOfData.textDS[2] <> '':
                        self.textName = self.textName + ': ' + separator.join(self.ContainerOfData.textDS)
                    else:
                        self.textName = self.textName + ': ' + separator.join([self.ContainerOfData.textDS[0], self.ContainerOfData.textDS[1]])
                else:
                    self.textName = self.textName + ': ' + self.ContainerOfData.textDS[0]
            else:
                self.textName = self.textName
            makeDatumSystem(self.textName, self.ContainerOfData.primary, self.ContainerOfData.secondary, self.ContainerOfData.tertiary)
        elif self.idGDT == 3:
            makeGeometricTolerance(self.textName, self.ContainerOfData)
        elif self.idGDT == 4:
            makeAnnotationPlane(self.textName, self.ContainerOfData.OffsetValue)
        else:
            pass

        if self.idGDT != 1 and self.idGDT != 3:
            hideGrid()

        FreeCADGui.Control.closeDialog()

def GDTDialog_hbox( label, inputWidget):
```

```
    hbox = QtGui.QHBoxLayout()
    hbox.addWidget( QtGui.QLabel(label) )
    if inputWidget <> None:
        hbox.addStretch(1)
        hbox.addWidget(inputWidget)
    return hbox

class textLabelWidget:
    def __init__(self, Text='Label', Mask=None, Dictionary=None):
        self.Text = Text
        self.Mask = Mask
        self.Dictionary = Dictionary

    def generateWidget( self, idGDT, ContainerOfData ):
        self.idGDT = idGDT
        self.ContainerOfData = ContainerOfData
        self.lineEdit = QtGui.QLineEdit()
        if self.Mask <> None:
            self.lineEdit.setInputMask(self.Mask)
        if self.Dictionary == None:
            self.lineEdit.setText('text')
            self.text = 'text'
        else:
            NumberOfObjects = self.getNumberOfObjects()
            if NumberOfObjects > len(self.Dictionary)-1:
                NumberOfObjects = len(self.Dictionary)-1
            self.lineEdit.setText(self.Dictionary[NumberOfObjects])
            self.text = self.Dictionary[NumberOfObjects]
        self.lineEdit.textChanged.connect(self.valueChanged)
        self.ContainerOfData.textName = self.text.strip()
        return GDTDialog_hbox(self.Text,self.lineEdit)

    def valueChanged(self, argGDT):
        self.text = argGDT.strip()
        self.ContainerOfData.textName = self.text

    def getNumberOfObjects(self):
        "getNumberOfObjects(): returns the number of objects of the same type as the active widget"
        if self.idGDT == 1:
            NumberOfObjects = len(getAllDatumFeatureObjects())
        elif self.idGDT == 2:
            NumberOfObjects = len(getAllDatumSystemObjects())
        elif self.idGDT == 3:
            NumberOfObjects = len(getAllGeometricToleranceObjects())
        elif self.idGDT == 4:
            NumberOfObjects = len(getAllAnnotationPlaneObjects())
        else:
            NumberOfObjects = 0
        return NumberOfObjects

class fieldLabelWidget:
    def __init__(self, Text='Label'):
        self.Text = Text

    def generateWidget( self, idGDT, ContainerOfData ):
        self.idGDT = idGDT
        self.ContainerOfData = ContainerOfData
        FreeCAD.DraftWorkingPlane.alignToPointAndAxis(self.ContainerOfData.p1, self.ContainerOfData.Direction, 0.0)
        FreeCADGui.Snapper.grid.set()
        self.FORMAT = makeFormatSpec(0,'Length')
        self.uiloader = FreeCADGui.UiLoader()
        self.inputfield = self.uiloader.createWidget("Gui::InputField")
        self.inputfield.setText(self.FORMAT % 0)
        self.ContainerOfData.OffsetValue = 0
        QtCore.QObject.connect(self.inputfield,QtCore.SIGNAL("valueChanged(double)"),self.valueChanged)

        return GDTDialog_hbox(self.Text,self.inputfield)

    def valueChanged(self, d):
        self.ContainerOfData.OffsetValue = d
        FreeCAD.DraftWorkingPlane.alignToPointAndAxis(self.ContainerOfData.p1, self.ContainerOfData.Direction,
self.ContainerOfData.OffsetValue)
        FreeCADGui.Snapper.grid.set()

class comboLabelWidget:
    def __init__(self, Text='Label', List=None, Icons=None, ToolTip = None):
        self.Text = Text
        self.List = List
        self.Icons = Icons
        self.ToolTip = ToolTip
```

```
def generateWidget( self, idGDT, ContainerOfData ):
    self.idGDT = idGDT
    self.ContainerOfData = ContainerOfData

    if self.Text == 'Primary:':
        self.k=0
    elif self.Text == 'Secondary:':
        self.k=1
    elif self.Text == 'Tertiary:':
        self.k=2
    elif self.Text == 'Characteristic:':
        self.k=3
    elif self.Text == 'Datum system:':
        self.k=4
    elif self.Text == 'Active annotation plane:':
        self.k=5
    else:
        self.k=6

    self.ContainerOfData.combo[self.k] = QtGui.QComboBox()
    for i in range(len(self.List)):
        if self.Icons <> None:
            self.ContainerOfData.combo[self.k].addItem( QtGui.QIcon(self.Icons[i]), self.List[i] )
        else:
            if self.List[i] == None:
                self.ContainerOfData.combo[self.k].addItem( '' )
            else:
                self.ContainerOfData.combo[self.k].addItem( self.List[i].Label )
    if self.Text == 'Secondary:' or self.Text == 'Tertiary:':
        self.ContainerOfData.combo[self.k].setEnabled(False)
    if self.ToolTip <> None:
        self.ContainerOfData.combo[self.k].setToolTip( self.ToolTip[0] )
    self.comboIndex = self.ContainerOfData.combo[self.k].currentIndex()
    if self.k <> 0 and self.k <> 1:
        self.updateDate(self.comboIndex)
    self.ContainerOfData.combo[self.k].activated.connect(lambda comboIndex = self.comboIndex: self.updateDate(comboIndex))
    return GDTDialog_hbox(self.Text,self.ContainerOfData.combo[self.k])

def updateDate(self, comboIndex):
    if self.ToolTip <> None:
        self.ContainerOfData.combo[self.k].setToolTip( self.ToolTip[comboIndex] )
    if self.Text == 'Primary:':
        self.ContainerOfData.textDS[0] = self.ContainerOfData.combo[self.k].currentText()
        self.ContainerOfData.primary = self.List[comboIndex]
        if comboIndex <> 0:
            self.ContainerOfData.combo[1].setEnabled(True)
        else:
            self.ContainerOfData.combo[1].setEnabled(False)
            self.ContainerOfData.combo[2].setEnabled(False)
            self.ContainerOfData.combo[1].setCurrentIndex(0)
            self.ContainerOfData.combo[2].setCurrentIndex(0)
            self.ContainerOfData.textDS[1] = ''
            self.ContainerOfData.textDS[2] = ''
            self.ContainerOfData.secondary = None
            self.ContainerOfData.tertiary = None
        self.updateItemsEnabled(self.k)
    elif self.Text == 'Secondary:':
        self.ContainerOfData.textDS[1] = self.ContainerOfData.combo[self.k].currentText()
        self.ContainerOfData.secondary = self.List[comboIndex]
        if comboIndex <> 0:
            self.ContainerOfData.combo[2].setEnabled(True)
        else:
            self.ContainerOfData.combo[2].setEnabled(False)
            self.ContainerOfData.combo[2].setCurrentIndex(0)
            self.ContainerOfData.textDS[2] = ''
            self.ContainerOfData.tertiary = None
        self.updateItemsEnabled(self.k)
    elif self.Text == 'Tertiary:':
        self.ContainerOfData.textDS[2] = self.ContainerOfData.combo[self.k].currentText()
        self.ContainerOfData.tertiary = self.List[comboIndex]
        self.updateItemsEnabled(self.k)
    elif self.Text == 'Characteristic:':
        self.ContainerOfData.characteristic = makeCharacteristics(self.List[comboIndex])
    elif self.Text == 'Datum system:':
        self.ContainerOfData.datumSystem = self.List[comboIndex]
    elif self.Text == 'Active annotation plane:':
        self.ContainerOfData.annotationPlane = self.List[comboIndex]
        self.ContainerOfData.Direction = self.List[comboIndex].Direction
        self.ContainerOfData.PointWithOffset = self.List[comboIndex].PointWithOffset
        FreeCAD.DraftWorkingPlane.alignToPointAndAxis(self.ContainerOfData.PointWithOffset, self.ContainerOfData.Direction, 0.0)
        FreeCADGui.Snapper.grid.set()
```

```
    def updateItemsEnabled(self, comboIndex):
        comboIndex0 = comboIndex
        comboIndex1 = (comboIndex+1) % 3
        comboIndex2 = (comboIndex+2) % 3

        for i in range(self.ContainerOfData.combo[comboIndex0].count()):
            self.ContainerOfData.combo[comboIndex0].model().item(i).setEnabled(True)
        if self.ContainerOfData.combo[comboIndex1].currentIndex() <> 0:

self.ContainerOfData.combo[comboIndex0].model().item(self.ContainerOfData.combo[comboIndex1].currentIndex()).setEnabled(False)
        if self.ContainerOfData.combo[comboIndex2].currentIndex() <> 0:

self.ContainerOfData.combo[comboIndex0].model().item(self.ContainerOfData.combo[comboIndex2].currentIndex()).setEnabled(False)
        for i in range(self.ContainerOfData.combo[comboIndex1].count()):
            self.ContainerOfData.combo[comboIndex1].model().item(i).setEnabled(True)
        if self.ContainerOfData.combo[comboIndex0].currentIndex() <> 0:

self.ContainerOfData.combo[comboIndex1].model().item(self.ContainerOfData.combo[comboIndex0].currentIndex()).setEnabled(False)
        if self.ContainerOfData.combo[comboIndex2].currentIndex() <> 0:

self.ContainerOfData.combo[comboIndex1].model().item(self.ContainerOfData.combo[comboIndex2].currentIndex()).setEnabled(False)
        for i in range(self.ContainerOfData.combo[comboIndex2].count()):
            self.ContainerOfData.combo[comboIndex2].model().item(i).setEnabled(True)
        if self.ContainerOfData.combo[comboIndex0].currentIndex() <> 0:

self.ContainerOfData.combo[comboIndex2].model().item(self.ContainerOfData.combo[comboIndex0].currentIndex()).setEnabled(False)
        if self.ContainerOfData.combo[comboIndex1].currentIndex() <> 0:

self.ContainerOfData.combo[comboIndex2].model().item(self.ContainerOfData.combo[comboIndex1].currentIndex()).setEnabled(False)

class groupBoxWidget:
    def __init__(self, Text='Label', List=[]):
        self.Text = Text
        self.List = List

    def generateWidget( self, idGDT, ContainerOfData ):
        self.idGDT = idGDT
        self.ContainerOfData = ContainerOfData
        self.group = QtGui.QGroupBox(self.Text)
        vbox = QtGui.QVBoxLayout()
        for l in self.List:
            vbox.addLayout(l.generateWidget(self.idGDT, self.ContainerOfData))
        self.group.setLayout(vbox)
        return self.group

class fieldLabeCombolWidget:
    def __init__(self, Text='Label', Circumference = [''], Diameter = 0.0, toleranceSelect = True, tolerance = 0.0, lowLimit = 0.0, highLimit =
0.0, List=[''], Icons=None, ToolTip = None):
        self.Text = Text
        self.Circumference = Circumference
        self.Diameter = Diameter
        self.toleranceSelect = toleranceSelect
        self.tolerance = tolerance
        self.lowLimit = lowLimit
        self.highLimit = highLimit
        self.List = List
        self.Icons = Icons
        self.ToolTip = ToolTip

    def generateWidget( self, idGDT, ContainerOfData ):
        self.idGDT = idGDT
        self.ContainerOfData = ContainerOfData
        self.DECIMALS = FreeCAD.ParamGet("User parameter:BaseApp/Preferences/Units").GetInt("Decimals",2)
        self.FORMAT = makeFormatSpec(self.DECIMALS,'Length')
        self.AFORMAT = makeFormatSpec(self.DECIMALS,'Angle')
        self.uiloader = FreeCADGui.UiLoader()
        self.comboCircumference = QtGui.QComboBox()
        self.combo = QtGui.QComboBox()
        for i in range(len(self.Circumference)):
            self.comboCircumference.addItem(QtGui.QIcon(self.Circumference[i]), '' )
        self.comboCircumference.setSizeAdjustPolicy(QtGui.QComboBox.SizeAdjustPolicy(2))
        self.comboCircumference.setToolTip("Indicates whether the tolerance applies to a given diameter")
        self.combo.setSizeAdjustPolicy(QtGui.QComboBox.SizeAdjustPolicy(2))
        for i in range(len(self.List)):
            if self.Icons <> None:
                self.combo.addItem( QtGui.QIcon(self.Icons[i]), self.List[i] )
            else:
                self.combo.addItem( self.List[i] )
        if self.ToolTip <> None:
            self.updateDate()
```

```
        self.combo.activated.connect(self.updateDate)
        self.comboCircumference.activated.connect(self.updateDateCircumference)
        vbox = QtGui.QVBoxLayout()
        hbox1 = QtGui.QHBoxLayout()
        self.inputfield = self.uiloader.createWidget("Gui::InputField")
        self.inputfield.setText(self.FORMAT % 0)
        QtCore.QObject.connect(self.inputfield,QtCore.SIGNAL("valueChanged(double)"),self.valueChanged)
        hbox1.addWidget( QtGui.QLabel(self.Text) )
        hbox1.addWidget(self.comboCircumference)
        hbox1.addStretch(1)
        hbox1.addWidget(self.inputfield)
        hbox1.addStretch(1)
        hbox1.addWidget(self.combo)
        vbox.addLayout(hbox1)
        hbox2 = QtGui.QHBoxLayout()
        self.label = QtGui.QLabel('Diameter:')
        self.inputfield2 = self.uiloader.createWidget("Gui::InputField")
        auxText = displayExternal(self.Diameter,self.DECIMALS,'Length',True)
        self.inputfield2.setText(auxText)
        QtCore.QObject.connect(self.inputfield2,QtCore.SIGNAL("valueChanged(double)"),self.valueChangedDiameter)
        self.comboTolerance = QtGui.QComboBox()
        simbol = '±'
        self.comboTolerance.addItem( simbol[-1] )
        self.comboTolerance.addItem( 'Limit' )
        if self.toleranceSelect:
            self.comboTolerance.setCurrentIndex(0)
        else:
            self.comboTolerance.setCurrentIndex(1)
        self.updateDateTolerance
        self.comboTolerance.activated.connect(self.updateDateTolerance)
        self.labelTolerance = QtGui.QLabel(simbol[-1])
        self.labelLow = QtGui.QLabel('Low')
        self.labelHigh = QtGui.QLabel('High')
        self.inputfieldTolerance = self.uiloader.createWidget("Gui::InputField")
        auxText = displayExternal(self.tolerance,self.DECIMALS,'Length',True)
        self.inputfieldTolerance.setText(auxText)
        QtCore.QObject.connect(self.inputfieldTolerance,QtCore.SIGNAL("valueChanged(double)"),self.valueChangedTolerance)
        self.inputfieldLow = self.uiloader.createWidget("Gui::InputField")
        auxText = displayExternal(self.lowLimit,self.DECIMALS,'Length',True)
        self.inputfieldLow.setText(auxText)
        QtCore.QObject.connect(self.inputfieldLow,QtCore.SIGNAL("valueChanged(double)"),self.valueChangedLow)
        self.inputfieldHigh = self.uiloader.createWidget("Gui::InputField")
        auxText = displayExternal(self.highLimit,self.DECIMALS,'Length',True)
        self.inputfieldHigh.setText(auxText)
        QtCore.QObject.connect(self.inputfieldHigh,QtCore.SIGNAL("valueChanged(double)"),self.valueChangedHigh)

        hbox2.addWidget(self.label)
        hbox2.addStretch(1)
        hbox2.addWidget(self.inputfield2)
        vbox.addLayout(hbox2)
        hbox3 = QtGui.QHBoxLayout()
        hbox3.addWidget(self.comboTolerance)
        hbox3.addStretch(1)
        hbox3.addWidget(self.labelTolerance)
        hbox3.addWidget(self.inputfieldTolerance)
        hbox3.addWidget(self.labelLow)
        hbox3.addWidget(self.inputfieldLow)
        hbox3.addWidget(self.labelHigh)
        hbox3.addWidget(self.inputfieldHigh)
        vbox.addLayout(hbox3)
        self.label.hide()
        self.inputfield2.hide()
        self.label.hide()
        self.inputfield2.hide()
        self.comboTolerance.hide()
        self.labelTolerance.hide()
        self.inputfieldTolerance.hide()
        self.labelLow.hide()
        self.labelHigh.hide()
        self.inputfieldLow.hide()
        self.inputfieldHigh.hide()
        return vbox

    def updateDate(self):
        if self.ToolTip <> None:
            self.combo.setToolTip( self.ToolTip[self.combo.currentIndex()] )
        if self.Text == 'Tolerance value:':
            if self.combo.currentIndex() <> 0:
                self.ContainerOfData.featureControlFrame = makeFeatureControlFrame(self.ToolTip[self.combo.currentIndex()])
            else:
                self.ContainerOfData.featureControlFrame = makeFeatureControlFrame('')
```

```
    def updateDateCircumference(self):
        if self.comboCircumference.currentIndex() <> 0:
            self.ContainerOfData.circumference = True
            self.label.show()
            self.inputfield2.show()
            self.label.show()
            self.inputfield2.show()
            self.comboTolerance.show()
            if self.comboTolerance.currentIndex() == 0:
                self.labelTolerance.show()
                self.inputfieldTolerance.show()
            else:
                self.labelLow.show()
                self.labelHigh.show()
                self.inputfieldLow.show()
                self.inputfieldHigh.show()
        else:
            self.ContainerOfData.circumference = False
            self.label.hide()
            self.inputfield2.hide()
            self.label.hide()
            self.inputfield2.hide()
            self.comboTolerance.hide()
            if self.comboTolerance.currentIndex() == 0:
                self.labelTolerance.hide()
                self.inputfieldTolerance.hide()
            else:
                self.labelLow.hide()
                self.labelHigh.hide()
                self.inputfieldLow.hide()
                self.inputfieldHigh.hide()

    def updateDateTolerance(self):
        if self.comboTolerance.currentIndex() <> 0:
            self.ContainerOfData.toleranceSelect = False
            self.labelTolerance.hide()
            self.inputfieldTolerance.hide()
            self.labelLow.show()
            self.labelHigh.show()
            self.inputfieldLow.show()
            self.inputfieldHigh.show()
        else:
            self.ContainerOfData.toleranceSelect = True
            self.labelTolerance.show()
            self.inputfieldTolerance.show()
            self.labelLow.hide()
            self.labelHigh.hide()
            self.inputfieldLow.hide()
            self.inputfieldHigh.hide()

    def valueChanged(self,d):
        self.ContainerOfData.toleranceValue = d

    def valueChangedDiameter(self,d):
        self.ContainerOfData.diameter = d

    def valueChangedTolerance(self,d):
        self.ContainerOfData.toleranceDiameter = d

    def valueChangedLow(self,d):
        self.ContainerOfData.lowLimit = d

    def valueChangedHigh(self,d):
        self.ContainerOfData.highLimit = d

class CheckBoxWidget:
    def __init__(self, Text='Label'):
        self.Text = Text

    def generateWidget( self, idGDT, ContainerOfData ):
        self.idGDT = idGDT
        self.ContainerOfData = ContainerOfData
        self.checkBox = QtGui.QCheckBox(self.Text)
        self.checkBox.setChecked(True)
        global checkBoxState
        checkBoxState = True
        hbox = QtGui.QHBoxLayout()
        hbox.addWidget(self.checkBox)
        hbox.addStretch(1)
        self.checkBox.stateChanged.connect(self.updateState)
```

```
        return hbox

    def updateState(self):
        global checkBoxState
        if self.checkBox.isChecked():
            checkBoxState = True
        else:
            checkBoxState = False
```

6.3 Anexo 3: inventory.py

```
#***************************************************************************
#*                                                                         *
#*   Copyright (c) 2016 Juan Vanyo Cerda <juavacer@inf.upv.es>             *
#*                                                                         *
#*   This program is free software; you can redistribute it and/or modify  *
#*   it under the terms of the GNU Lesser General Public License (LGPL)    *
#*   as published by the Free Software Foundation; either version 2 of     *
#*   the License, or (at your option) any later version.                   *
#*   for detail see the LICENCE text file.                                 *
#*                                                                         *
#*   This program is distributed in the hope that it will be useful,       *
#*   but WITHOUT ANY WARRANTY; without even the implied warranty of        *
#*   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the         *
#*   GNU Library General Public License for more details.                  *
#*                                                                         *
#*   You should have received a copy of the GNU Library General Public     *
#*   License along with this program; if not, write to the Free Software   *
#*   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  *
#*   USA                                                                   *
#*                                                                         *
#***************************************************************************
import FreeCAD as App
import FreeCADGui as Gui

from GDT import *

class InventoryCommand:
    def __init__(self):
        self.iconPath = ':/dd/icons/inventory.svg'
        self.toolTip = 'Inventory of the elements of GD&T'

    def Activated(self):
        Gui.Control.showDialog( GDTGuiClass() )

    def GetResources(self):
        return {
            'Pixmap' : self.iconPath,
            'MenuText': self.toolTip,
            'ToolTip':  self.toolTip
            }

    def IsActive(self):
        if FreeCADGui.ActiveDocument:
            return True
        else:
            return False

class GDTGuiClass:
    def __init__(self):
        self.widgetsGDT = []
        inventory = getAllAnnotationPlaneObjects() + getAllDatumSystemObjects() + getAllDatumFeatureObjects() +
getAllGeometricToleranceObjects()
        for obj in inventory:
            self.widget = QtGui.QWidget()
            self.widget.setWindowTitle( obj.Label )
            self.widget.setWindowIcon( obj.ViewObject.Icon )
            self.widgetsGDT.append(self.widget)
            self.dialogWidgets = []
            vbox = QtGui.QVBoxLayout()
            hbox = QtGui.QHBoxLayout()
            self.data = ContainerOfData()
            self.data.textName = obj.Label
            if "AnnotationPlane" == getType(obj):
                self.dialogWidgets.append(textLabelWidget_inv(Text = 'Name:', Mask = 'NNNn', Data = self.data, Obj = obj))
                self.dialogWidgets.append(fieldLabelButtonWidget_inv(Text = 'Offset:', Data = self.data, Obj = obj))

            elif "DatumSystem" == getType(obj):
                self.dialogWidgets.append(textLabelWidget_inv(Text = 'Name:', Mask='NNNn', Data = self.data, Obj = obj))
```

```
            listDF = [None] + [l for l in getAllDatumFeatureObjects()]
            self.dialogWidgets.append( groupBoxWidget_inv(Text='Constituents', List=[comboLabelWidget_inv(Text='Primary:',List=listDF,
Data = self.data, Obj = obj),comboLabelWidget_inv(Text='Secondary:',List=listDF, Data = self.data, Obj = obj),
comboLabelWidget_inv(Text='Tertiary:',List=listDF, Data = self.data, Obj = obj)], Data = self.data, Obj = obj) )

        elif "DatumFeature" == getType(obj):
            self.dialogWidgets.append(textLabelWidget_inv(Text = 'Datum feature:', Mask='>A', Data = self.data, Obj = obj))
            self.dialogWidgets.append( comboLabelWidget_inv(Text='In annotation:', List = [l for l in getAllAnnotationObjects()], Data =
self.data, Obj = obj) )

        elif "GeometricTolerance" == getType(obj):
            self.dialogWidgets.append(textLabelWidget_inv(Text = 'Name:', Mask='NNNn', Data = self.data, Obj = obj))
            characteristics = makeCharacteristics()
            self.dialogWidgets.append( comboLabelWidget_inv(Text='Characteristic:', List=characteristics.Label, Icons=characteristics.Icon,
Data = self.data, Obj = obj) )
            featureControlFrame = makeFeatureControlFrame()
            self.dialogWidgets.append( fieldLabeCombolWidget_inv(Text='Tolerance value:', List=featureControlFrame.Label,
Circumference=['',':/dd/icons/diameter.svg'] , Icons=featureControlFrame.Icon, ToolTip=featureControlFrame.toolTip, Data = self.data, Obj
= obj) ) #http://doc.qt.io/qt-5/qlineedit.html#inputMask-prop
            self.dialogWidgets.append( comboLabelWidget_inv(Text='Datum system:', List=[None] + [l for l in
getAllDatumSystemObjects()], Data = self.data, Obj = obj) )
            self.dialogWidgets.append( comboLabelWidget_inv(Text='In annotation:', List=[l for l in getAllAnnotationObjects()], Data =
self.data, Obj = obj) )

        for widg in self.dialogWidgets:
            w = widg.generateWidget()
            if isinstance(w, QtGui.QLayout):
                vbox.addLayout( w )
            else:
                vbox.addWidget( w )

        buttonModify = QtGui.QPushButton('Modify')
        buttonModify.clicked.connect(lambda obj = obj, data = self.data: self.modifyFunc(obj, data))
        buttonDelate = QtGui.QPushButton('Delete')
        buttonDelate.clicked.connect(lambda obj = obj: self.deleteFunc(obj))
        hbox.addStretch(1)
        hbox.addWidget( buttonModify )
        hbox.addWidget( buttonDelate )
        hbox.addStretch(1)
        vbox.addLayout(hbox)
        self.widget.setLayout(vbox)

    self.form = self.widgetsGDT

def modifyFunc(self, obj, data):
    if "AnnotationPlane" == getType(obj):
        obj.Label = data.textName
        obj.Offset = data.OffsetValue

    elif "DatumSystem" == getType(obj):
        obj.Primary = data.primary
        obj.Secondary = data.secondary
        obj.Tertiary = data.tertiary
        textName = data.textName.split(":")[0]
        if data.primary <> None:
            textName+=': '+obj.Primary.Label
            if data.secondary <> None:
                textName+=' | '+obj.Secondary.Label
                if data.tertiary <> None:
                    textName+=' | '+obj.Tertiary.Label
        obj.Label = textName

    elif "DatumFeature" == getType(obj):
        annotationObj = getAnnotationWithDF(obj)
        remove = False
        if data.annotation.DF <> None and annotationObj <> data.annotation:
            QtGui.QMessageBox.critical(
                QtGui.qApp.activeWindow(),
                'ERROR',
                'You can not change the DF to an annotation where one already exists',
                QtGui.QMessageBox.StandardButton.Abort )
        elif annotationObj <> data.annotation:
            data.annotation.addObject(obj)
            data.annotation.DF = obj
            annotationObj.removeObject(obj)
            remove = True
        for l in getAllDatumSystemObjects():
            if l.Primary == obj or l.Secondary == obj or l.Tertiary == obj:
                l.touch
        obj.Label = data.textName
        if remove:
```

```
            annotationObj.DF = None

        elif "GeometricTolerance" == getType(obj):
            annotationObj = getAnnotationWithGT(obj)
            if annotationObj.Label <> data.annotation.Label:
                annotationObj.removeObject(obj)
                gt = annotationObj.GT
                gt.remove(obj)
                annotationObj.GT = gt
                data.annotation.addObject(obj)
                gt = data.annotation.GT
                gt.append(obj)
                data.annotation.GT = gt
            obj.Label = data.textName
            obj.Characteristic = data.characteristic.Label
            obj.CharacteristicIcon = data.characteristic.Icon
            obj.ToleranceValue = data.toleranceValue
            obj.Circumference = data.circumference
            obj.FeatureControlFrame = data.featureControlFrame
            obj.DS = data.datumSystem

        FreeCADGui.Control.closeDialog()
        hideGrid()
        for l in getAllAnnotationObjects():
            l.touch()
        FreeCAD.ActiveDocument.recompute()
        Gui.Control.showDialog( GDTGuiClass() )

    def deleteFunc(self, obj):
        if "AnnotationPlane" == getType(obj):
            ok = True
            for l in getAllAnnotationObjects():
                if l.AP == obj:
                    ok = False
                    break
            if ok:
                FreeCAD.ActiveDocument.removeObject(obj.Name)
            else:
                QtGui.QMessageBox.critical(
                    QtGui.qApp.activeWindow(),
                    'ERROR',
                    'You can not delete AP in use',
                    QtGui.QMessageBox.StandardButton.Abort )

        elif "DatumSystem" == getType(obj):
            FreeCAD.ActiveDocument.removeObject(obj.Name)

        elif "DatumFeature" == getType(obj):
            for l in getAllDatumSystemObjects():
                if l.Primary == obj:
                    l.Primary = l.Secondary
                    l.Secondary = l.Tertiary
                    l.Tertiary = None
                elif l.Secondary == obj:
                    l.Secondary = l.Tertiary
                    l.Tertiary = None
            for l in getAllAnnotationObjects():
                if l.DF == obj:
                    l.DF = None
                    break
            FreeCAD.ActiveDocument.removeObject(obj.Name)

        elif "GeometricTolerance" == getType(obj):
            for l in getAllAnnotationObjects():
                for gt in l.GT:
                    if gt == obj:
                        gtAux = l.GT
                        gtAux.remove(obj)
                        l.GT = gtAux
                        break
            FreeCAD.ActiveDocument.removeObject(obj.Name)

        FreeCADGui.Control.closeDialog()
        hideGrid()
        for l in getAllAnnotationObjects():
            l.touch()
        FreeCAD.ActiveDocument.recompute()
        Gui.Control.showDialog( GDTGuiClass() )

    def getPos(self, actualValue, List):
        for i in range(len(List)):
```

```
            if List[i][0] == actualValue:
                return i

    def reject(self): #close button
        hideGrid()
        FreeCADGui.Control.closeDialog()

    def getStandardButtons(self): #http://forum.freecadweb.org/viewtopic.php?f=10&t=11801
        return 0x00200000 #close button

class textLabelWidget_inv:
    def __init__(self, Text='Label', Mask = None, Data = None, Obj = None):
        self.Text = Text
        self.Mask = Mask
        self.data = Data
        self.obj = Obj

    def generateWidget( self ):
        self.lineEdit = QtGui.QLineEdit()
        if self.Mask <> None:
            self.lineEdit.setInputMask(self.Mask)
        self.lineEdit.setText(self.obj.Label)
        self.text = self.obj.Label
        self.lineEdit.textChanged.connect(lambda Txt = self.text: self.valueChanged(Txt))
        return GDTDialog_hbox_inv(self.Text, self.lineEdit)

    def valueChanged(self, argGDT):
        self.data.textName = argGDT.strip()

class fieldLabelButtonWidget_inv:
    def __init__(self, Text='Label', Data = None, Obj = None):
        self.Text = Text
        self.data = Data
        self.obj = Obj

    def generateWidget( self ):
        hbox = QtGui.QHBoxLayout()
        self.data.OffsetValue = self.obj.Offset
        self.DECIMALS = FreeCAD.ParamGet("User parameter:BaseApp/Preferences/Units").GetInt("Decimals",2)
        self.FORMAT = makeFormatSpec(self.DECIMALS,'Length')
        self.uiloader = FreeCADGui.UiLoader()
        self.inputfield = self.uiloader.createWidget("Gui::InputField")
        self.auxText = displayExternal(self.obj.Offset,self.DECIMALS,'Length',True)
        self.inputfield.setText(self.auxText)
        self.firstAttempt = True
        QtCore.QObject.connect(self.inputfield,QtCore.SIGNAL("valueChanged(double)"),lambda Double = self.auxText:
self.valueChanged(Double))

        self.button = QtGui.QPushButton('Visualize')
        self.button.clicked.connect(self.visualizeFunc)
        hbox.addLayout( GDTDialog_hbox(self.Text,self.inputfield) )
        hbox.addStretch(1)
        hbox.addWidget(self.button)
        return hbox

    def valueChanged(self, d):
        self.data.OffsetValue = d
        if self.firstAttempt == False:
            showGrid()
            FreeCAD.DraftWorkingPlane.alignToPointAndAxis(self.obj.p1, self.obj.Direction, self.data.OffsetValue)
            FreeCADGui.Snapper.grid.set()
        self.firstAttempt = False

    def visualizeFunc(self):
        showGrid()
        FreeCAD.DraftWorkingPlane.alignToPointAndAxis(self.obj.PointWithOffset, self.obj.Direction, 0)
        FreeCADGui.Snapper.grid.set()
        self.inputfield.setText(self.auxText)

class comboLabelWidget_inv:
    def __init__(self, Text='Label', List=[None], Icons=None, ToolTip = None, Data = None, Obj = None):
        self.Text = Text
        self.List = List
        self.Icons = Icons
        self.ToolTip = ToolTip
        self.data = Data
        self.obj = Obj

    def generateWidget( self ):
        if self.Text == 'Primary:':
            self.k=0
```

```
        elif self.Text == 'Secondary:':
            self.k=1
        elif self.Text == 'Tertiary:':
            self.k=2
        elif self.Text == 'Characteristic:':
            self.k=3
        elif self.Text == 'Datum system:':
            self.k=4
        elif self.Text == 'In annotation:':
            self.k=5
        else:
            self.k=6
        self.data.combo[self.k] = QtGui.QComboBox()
        for i in range(len(self.List)):
            if self.List[i] == None:
                self.data.combo[self.k].addItem( '' )
            elif self.Icons <> None:
                self.data.combo[self.k].addItem( QtGui.QIcon(self.Icons[i]), self.List[i] )
            else:
                self.data.combo[self.k].addItem( self.List[i].Label )

        if self.ToolTip <> None:
            self.data.combo[self.k].setToolTip( self.ToolTip[0] )
        self.updateCurrentItem()
        if self.Text == 'Secondary:' and self.data.combo[self.k].currentIndex() == 0 or self.Text == 'Tertiary:' and
self.data.combo[self.k].currentIndex() == 0:
            self.data.combo[self.k].setEnabled(False)
        if self.k == 2:
            self.updateItemsEnabled(self.k)
            if self.data.combo[0].currentIndex() <> 0:
                self.data.combo[1].setEnabled(True)
                if self.data.combo[1].currentIndex() <> 0:
                    self.data.combo[2].setEnabled(True)
        self.data.combo[self.k].activated.connect(lambda comboIndex = self.data.combo[self.k].currentIndex(): self.updateDate(comboIndex))

        return GDTDialog_hbox_inv(self.Text, self.data.combo[self.k])

    def updateCurrentItem(self):
        if self.Text == 'Primary:':
            if self.obj.Primary <> None:
                actualValue = self.obj.Primary.Label
                pos = self.getPos(actualValue)
                self.data.combo[self.k].setCurrentIndex(pos)
            self.data.textDS[0] = self.data.combo[self.k].currentText()
            self.data.primary = self.List[self.data.combo[self.k].currentIndex()]
        elif self.Text == 'Secondary:':
            if self.obj.Secondary <> None:
                actualValue = self.obj.Secondary.Label
                pos = self.getPos(actualValue)
                self.data.combo[self.k].setCurrentIndex(pos)
            self.data.textDS[1] = self.data.combo[self.k].currentText()
            self.data.secondary = self.List[self.data.combo[self.k].currentIndex()]
        elif self.Text == 'Tertiary:':
            if self.obj.Tertiary <> None:
                actualValue = self.obj.Tertiary.Label
                pos = self.getPos(actualValue)
                self.data.combo[self.k].setCurrentIndex(pos)
            self.data.textDS[2] = self.data.combo[self.k].currentText()
            self.data.tertiary = self.List[self.data.combo[self.k].currentIndex()]
        elif self.Text == 'Characteristic:':
            if self.obj.Characteristic <> '':
                actualValue = self.obj.Characteristic
                pos = self.getPos(actualValue)
            self.data.combo[self.k].setCurrentIndex(pos)
            self.data.characteristic = makeCharacteristics(self.List[self.data.combo[self.k].currentIndex()])
        elif self.Text == 'Datum system:':
            if self.obj.DS <> None:
                actualValue = self.obj.DS.Label
                pos = self.getPos(actualValue)
                self.data.combo[self.k].setCurrentIndex(pos)
            self.data.datumSystem = self.List[self.data.combo[self.k].currentIndex()]
        elif self.Text == 'In annotation:':
            annotationObj = getAnnotationWithDF(self.obj) if "DatumFeature" == getType(self.obj) else getAnnotationWithGT(self.obj)
            if annotationObj <> None:
                actualValue = annotationObj.Label
                pos = self.getPos(actualValue)
                self.data.combo[self.k].setCurrentIndex(pos)
            self.data.annotation = self.List[self.data.combo[self.k].currentIndex()]

    def getPos(self, actualValue):
        for i in range(len(self.List)):
```

```
            if isinstance(self.List[i],str):
                if self.List[i] == actualValue:
                    return i
            elif self.List[i] == None:
                pass
            elif self.List[i].Label == actualValue:
                return i

    def updateDate(self, comboIndex):
        if self.ToolTip <> None:
            self.data.combo[self.k].setToolTip( self.ToolTip[self.data.combo[self.k].currentIndex()] )
        if self.Text == 'Primary:':
            self.data.textDS[0] = self.data.combo[self.k].currentText()
            self.data.primary = self.List[self.data.combo[self.k].currentIndex()]
            if self.data.combo[self.k].currentIndex() <> 0:
                self.data.combo[1].setEnabled(True)
            else:
                self.data.combo[1].setEnabled(False)
                self.data.combo[2].setEnabled(False)
                self.data.combo[1].setCurrentIndex(0)
                self.data.combo[2].setCurrentIndex(0)
                self.data.textDS[1] = ''
                self.data.textDS[2] = ''
                self.data.secondary = None
                self.data.tertiary = None
            self.updateItemsEnabled(self.k)
        elif self.Text == 'Secondary:':
            self.data.textDS[1] = self.data.combo[self.k].currentText()
            self.data.secondary = self.List[self.data.combo[self.k].currentIndex()]
            if self.data.combo[self.k].currentIndex() <> 0:
                self.data.combo[2].setEnabled(True)
            else:
                self.data.combo[2].setEnabled(False)
                self.data.combo[2].setCurrentIndex(0)
                self.data.textDS[2] = ''
                self.data.tertiary = None
            self.updateItemsEnabled(self.k)
        elif self.Text == 'Tertiary:':
            self.data.textDS[2] = self.data.combo[self.k].currentText()
            self.data.tertiary = self.List[self.data.combo[self.k].currentIndex()]
            self.updateItemsEnabled(self.k)
        elif self.Text == 'Characteristic:':
            self.data.characteristic = makeCharacteristics(self.List[self.data.combo[self.k].currentIndex()])
        elif self.Text == 'Datum system:':
            self.data.datumSystem = self.List[self.data.combo[self.k].currentIndex()]
        elif self.Text == 'In annotation:':
            self.data.annotation = self.List[self.data.combo[self.k].currentIndex()]

    def updateItemsEnabled(self, comboIndex):
        comboIndex0 = comboIndex
        comboIndex1 = (comboIndex+1) % 3
        comboIndex2 = (comboIndex+2) % 3

        for i in range(self.data.combo[comboIndex0].count()):
            self.data.combo[comboIndex0].model().item(i).setEnabled(True)
        if self.data.combo[comboIndex1].currentIndex() <> 0:
            self.data.combo[comboIndex0].model().item(self.data.combo[comboIndex1].currentIndex()).setEnabled(False)
        if self.data.combo[comboIndex2].currentIndex() <> 0:
            self.data.combo[comboIndex0].model().item(self.data.combo[comboIndex2].currentIndex()).setEnabled(False)
        for i in range(self.data.combo[comboIndex1].count()):
            self.data.combo[comboIndex1].model().item(i).setEnabled(True)
        if self.data.combo[comboIndex0].currentIndex() <> 0:
            self.data.combo[comboIndex1].model().item(self.data.combo[comboIndex0].currentIndex()).setEnabled(False)
        if self.data.combo[comboIndex2].currentIndex() <> 0:
            self.data.combo[comboIndex1].model().item(self.data.combo[comboIndex2].currentIndex()).setEnabled(False)
        for i in range(self.data.combo[comboIndex2].count()):
            self.data.combo[comboIndex2].model().item(i).setEnabled(True)
        if self.data.combo[comboIndex0].currentIndex() <> 0:
            self.data.combo[comboIndex2].model().item(self.data.combo[comboIndex0].currentIndex()).setEnabled(False)
        if self.data.combo[comboIndex1].currentIndex() <> 0:
            self.data.combo[comboIndex2].model().item(self.data.combo[comboIndex1].currentIndex()).setEnabled(False)

class groupBoxWidget_inv:
    def __init__(self, Text='Label', List=[], Data = None, Obj = None):
        self.Text = Text
        self.List = List
        self.data = Data
        self.obj = Obj

    def generateWidget( self ):
        self.group = QtGui.QGroupBox(self.Text)
```

```
        vbox = QtGui.QVBoxLayout()
        for l in self.List:
            vbox.addLayout(l.generateWidget())
        self.group.setLayout(vbox)
        return self.group

class fieldLabeCombolWidget_inv:
    def __init__(self, Text='Label', List=[''], Circumference = [''], Icons=None, ToolTip = None, Data = None, Obj = None):
        self.Text = Text
        self.List = List
        self.Circumference = Circumference
        self.Icons = Icons
        self.ToolTip = ToolTip
        self.data = Data
        self.obj = Obj

    def generateWidget( self ):
        self.DECIMALS = FreeCAD.ParamGet("User parameter:BaseApp/Preferences/Units").GetInt("Decimals",2)
        self.FORMAT = makeFormatSpec(self.DECIMALS,'Length')
        self.AFORMAT = makeFormatSpec(self.DECIMALS,'Angle')
        self.uiloader = FreeCADGui.UiLoader()
        self.comboCircumference = QtGui.QComboBox()
        self.combo = QtGui.QComboBox()
        for i in range(len(self.List)):
            if self.Icons <> None:
                self.combo.addItem( QtGui.QIcon(self.Icons[i]), self.List[i] )
            else:
                self.combo.addItem( self.List[i] )
        for i in range(len(self.Circumference)):
            self.comboCircumference.addItem(QtGui.QIcon(self.Circumference[i]), '' )
        self.comboCircumference.setSizeAdjustPolicy(QtGui.QComboBox.SizeAdjustPolicy(2))
        self.comboCircumference.setToolTip("Indicates whether the tolerance applies to a given diameter")
        self.combo.setSizeAdjustPolicy(QtGui.QComboBox.SizeAdjustPolicy(2))
        actualValue = self.obj.FeatureControlFrame
        pos = self.getPos(actualValue)
        self.combo.setCurrentIndex(pos)
        self.combo.setToolTip( self.ToolTip[self.combo.currentIndex()] )
        self.updateDate()
        self.combo.activated.connect(self.updateDate)
        if self.obj.Circumference:
            self.comboCircumference.setCurrentIndex(1)
            self.data.circumference = True
        else:
            self.comboCircumference.setCurrentIndex(0)
            self.data.circumference = False
        self.comboCircumference.activated.connect(self.updateDateCircumference)
        hbox = QtGui.QHBoxLayout()
        self.inputfield = self.uiloader.createWidget("Gui::InputField")
        auxText = displayExternal(self.obj.ToleranceValue,self.DECIMALS,'Length',True)
        self.inputfield.setText(auxText)
        self.data.toleranceValue = self.obj.ToleranceValue
        QtCore.QObject.connect(self.inputfield,QtCore.SIGNAL("valueChanged(double)"),lambda Double = auxText:
self.valueChanged(Double))
        hbox.addWidget( QtGui.QLabel(self.Text) )
        hbox.addWidget(self.comboCircumference)
        hbox.addStretch(1)
        hbox.addWidget(self.inputfield)
        hbox.addStretch(1)
        hbox.addWidget(self.combo)
        return hbox

    def getPos(self, actualValue):
        if actualValue == '':
            return 0
        else:
            for i in range(len(self.ToolTip)):
                if self.ToolTip[i] == actualValue:
                    return i

    def updateDate(self):
        if self.ToolTip <> None:
            self.combo.setToolTip( self.ToolTip[self.combo.currentIndex()] )
        if self.Text == 'Tolerance value:':
            self.data.featureControlFrame = '' if self.combo.currentIndex() == 0 else self.ToolTip[self.combo.currentIndex()]

    def updateDateCircumference(self):
        if self.comboCircumference.currentIndex() <> 0:
            self.data.circumference = True
        else:
            self.data.circumference = False
```

```
    def valueChanged(self,d):
        self.data.toleranceValue = d

def GDTDialog_hbox_inv( label, inputWidget):
    hbox = QtGui.QHBoxLayout()
    hbox.addWidget( QtGui.QLabel(label) )
    if inputWidget <> None:
        hbox.addStretch(1)
        hbox.addWidget(inputWidget)
    return hbox

FreeCADGui.addCommand('dd_inventory', InventoryCommand())
```

6.4 Anexo 4: annotationPlane.py

```
#***************************************************************************
#*                                                                         *
#*   Copyright (c) 2016 Juan Vanyo Cerda <juavacer@inf.upv.es>             *
#*                                                                         *
#*   This program is free software; you can redistribute it and/or modify  *
#*   it under the terms of the GNU Lesser General Public License (LGPL)    *
#*   as published by the Free Software Foundation; either version 2 of     *
#*   the License, or (at your option) any later version.                   *
#*   for detail see the LICENCE text file.                                 *
#*                                                                         *
#*   This program is distributed in the hope that it will be useful,       *
#*   but WITHOUT ANY WARRANTY; without even the implied warranty of        *
#*   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the         *
#*   GNU Library General Public License for more details.                  *
#*                                                                         *
#*   You should have received a copy of the GNU Library General Public     *
#*   License along with this program; if not, write to the Free Software   *
#*   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  *
#*   USA                                                                   *
#*                                                                         *
#***************************************************************************

from GDT import *
import  DraftTools

gdt = GDTWidget()
gdt.dialogWidgets.append( fieldLabelWidget(Text='Offset:') )
class AnnotationPlaneCommand:
    def __init__(self):
        self.iconPath = ':/dd/icons/annotationPlane.svg'
        self.toolTip = 'Add Annotation Plane'
        self.dictionary = []
        for i in range(1,100):
            self.dictionary.append('AP'+str(i))
        self.idGDT = 4

    def Activated(self):
        showGrid()
        gdt.activate(idGDT = self.idGDT, dialogTitle=self.toolTip, dialogIconPath=self.iconPath, endFunction=self.Activated,
dictionary=self.dictionary)

    def GetResources(self):
        return {
            'Pixmap' : self.iconPath,
            'MenuText': self.toolTip,
            'ToolTip':  self.toolTip
            }

    def IsActive(self):
        if len(getSelection()) == 1:
            return (getSelectionEx()[0].SubObjects[0].ShapeType == 'Face')
        else:
            return False

FreeCADGui.addCommand('dd_annotationPlane', AnnotationPlaneCommand())
```

6.5 Anexo 5: datumFeature.py

```
#***************************************************************************
#*                                                                         *
#*   Copyright (c) 2016 Juan Vanyo Cerda <juavacer@inf.upv.es>             *
#*                                                                         *
```

```
#*   This program is free software; you can redistribute it and/or modify  *
#*   it under the terms of the GNU Lesser General Public License (LGPL)    *
#*   as published by the Free Software Foundation; either version 2 of     *
#*   the License, or (at your option) any later version.                   *
#*   for detail see the LICENCE text file.                                 *
#*                                                                         *
#*   This program is distributed in the hope that it will be useful,       *
#*   but WITHOUT ANY WARRANTY; without even the implied warranty of        *
#*   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the         *
#*   GNU Library General Public License for more details.                  *
#*                                                                         *
#*   You should have received a copy of the GNU Library General Public     *
#*   License along with this program; if not, write to the Free Software   *
#*   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  *
#*   USA                                                                   *
#*                                                                         *
#***************************************************************************

from GDT import *
import  DraftTools

gdt = GDTWidget()
dictionaryDF = []
dictionaryDF = map(chr, range(65, 91)) # 65 = A, 91 = Z
gdt.dialogWidgets.append( textLabelWidget(Text='Datum feature:',Mask='>A', Dictionary=dictionaryDF) ) #http://doc.qt.io/qt-5/qlineedit.html#inputMask-prop
gdt.dialogWidgets.append( comboLabelWidget(Text='Active annotation plane:', List=[]) )
gdt.dialogWidgets.append( CheckBoxWidget(Text = 'Create corresponding Datum System') )

class DatumFeatureCommand:
   def __init__(self):
      self.iconPath = ':/dd/icons/datumFeature.svg'
      self.toolTip = 'Add Datum Feature'
      self.dictionary = []
      for i in range(1,100):
         self.dictionary.append('DF'+str(i))
      self.idGDT = 1

   def Activated(self):
      ContainerOfData = makeContainerOfData()
      if getAnnotationObj(ContainerOfData):
         self.toolTip = 'Add Datum Feature to ' + getAnnotationObj(ContainerOfData).Label
         gdt.dialogWidgets[1] = None
      else:
         self.toolTip = 'Add Datum Feature'
         showGrid()
         gdt.dialogWidgets[1] = comboLabelWidget(Text='Active annotation plane:', List=getAllAnnotationPlaneObjects())
      gdt.activate(idGDT = self.idGDT, dialogTitle=self.toolTip, dialogIconPath=self.iconPath, endFunction=self.Activated,
dictionary=self.dictionary)

   def GetResources(self):
      return {
         'Pixmap' : self.iconPath,
         'MenuText': self.toolTip,
         'ToolTip':  self.toolTip
         }

   def IsActive(self):
      if len(getObjectsOfType('AnnotationPlane')) == 0:
         return False
      if getSelection():
         for i in range(len(getSelectionEx())):
            if len(getSelectionEx()[i].SubObjects) == 0:
               return False
            for j in range(len(getSelectionEx()[i].SubObjects)):
               if getSelectionEx()[i].SubObjects[j].ShapeType == 'Face':
                  pass
               else:
                  return False
         ContainerOfData = makeContainerOfData()
         if getAnnotationObj(ContainerOfData) == None or getAnnotationObj(ContainerOfData).DF == None:
            pass
         else:
            return False
      else:
         return False
      return True

FreeCADGui.addCommand('dd_datumFeature', DatumFeatureCommand())
```

6.6 Anexo 6: datumSystem.py

```
#***************************************************************************
#*                                                                         *
#*   Copyright (c) 2016 Juan Vanyo Cerda <juavacer@inf.upv.es>             *
#*                                                                         *
#*   This program is free software; you can redistribute it and/or modify  *
#*   it under the terms of the GNU Lesser General Public License (LGPL)    *
#*   as published by the Free Software Foundation; either version 2 of     *
#*   the License, or (at your option) any later version.                   *
#*   for detail see the LICENCE text file.                                 *
#*                                                                         *
#*   This program is distributed in the hope that it will be useful,       *
#*   but WITHOUT ANY WARRANTY; without even the implied warranty of        *
#*   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the         *
#*   GNU Library General Public License for more details.                  *
#*                                                                         *
#*   You should have received a copy of the GNU Library General Public     *
#*   License along with this program; if not, write to the Free Software   *
#*   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  *
#*   USA                                                                   *
#*                                                                         *
#***************************************************************************

from GDT import *

gdt = GDTWidget()
gdt.dialogWidgets.append( groupBoxWidget(Text='Constituents',
List=[comboLabelWidget(Text='Primary:',List=[]),comboLabelWidget(Text='Secondary:',List=[]),
comboLabelWidget(Text='Tertiary:',List=[])]) )

class DatumSystemCommand:
    def __init__(self):
        self.iconPath = ':/dd/icons/datumSystem.svg'
        self.toolTip = 'Add Datum System'
        self.dictionary = []
        for i in range(1,100):
            self.dictionary.append('DS'+str(i))
        self.idGDT = 2

    def Activated(self):
        listDF = [None] + getAllDatumFeatureObjects()
        gdt.dialogWidgets[0] = ( groupBoxWidget(Text='Constituents',
List=[comboLabelWidget(Text='Primary:',List=listDF),comboLabelWidget(Text='Secondary:',List=listDF),
comboLabelWidget(Text='Tertiary:',List=listDF)]) )
        gdt.activate(idGDT = self.idGDT, dialogTitle=self.toolTip, dialogIconPath=self.iconPath, endFunction=self.Activated,
dictionary=self.dictionary)

    def GetResources(self):
        return {
            'Pixmap' : self.iconPath,
            'MenuText': self.toolTip,
            'ToolTip':  self.toolTip
            }

    def IsActive(self):
        if FreeCADGui.ActiveDocument:
            return len(getAllDatumFeatureObjects()) > 0
        else:
            return False

FreeCADGui.addCommand('dd_datumSystem', DatumSystemCommand())
```

6.7 Anexo 7: geometricTolerance.py

```
#***************************************************************************
#*                                                                         *
#*   Copyright (c) 2016 Juan Vanyo Cerda <juavacer@inf.upv.es>             *
#*                                                                         *
#*   This program is free software; you can redistribute it and/or modify  *
#*   it under the terms of the GNU Lesser General Public License (LGPL)    *
#*   as published by the Free Software Foundation; either version 2 of     *
#*   the License, or (at your option) any later version.                   *
#*   for detail see the LICENCE text file.                                 *
#*                                                                         *
#*   This program is distributed in the hope that it will be useful,       *
#*   but WITHOUT ANY WARRANTY; without even the implied warranty of        *
```

```
#*   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the        *
#*   GNU Library General Public License for more details.                 *
#*                                                                        *
#*   You should have received a copy of the GNU Library General Public    *
#*   License along with this program; if not, write to the Free Software  *
#*   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  *
#*   USA                                                                  *
#*                                                                        *
#**************************************************************************

from GDT import *
import  DraftTools

gdt = GDTWidget()
Label = ['Straightness', 'Flatness', 'Circularity', 'Cylindricity', 'Profile of a line', 'Profile of a surface', 'Perpendicularity', 'Angularity',
'Parallelism', 'Symmetry', 'Position', 'Concentricity','Circular run-out', 'Total run-out']
Icon = [':/dd/icons/Characteristic/straightness.svg', ':/dd/icons/Characteristic/flatness.svg', ':/dd/icons/Characteristic/circularity.svg',
':/dd/icons/Characteristic/cylindricity.svg', ':/dd/icons/Characteristic/profileOfALine.svg', ':/dd/icons/Characteristic/profileOfASurface.svg',
':/dd/icons/Characteristic/perpendicularity.svg', ':/dd/icons/Characteristic/angularity.svg', ':/dd/icons/Characteristic/parallelism.svg',
':/dd/icons/Characteristic/symmetry.svg', ':/dd/icons/Characteristic/position.svg',
':/dd/icons/Characteristic/concentricity.svg',':/dd/icons/Characteristic/circularRunOut.svg', ':/dd/icons/Characteristic/totalRunOut.svg']
gdt.dialogWidgets.append( comboLabelWidget(Text='Characteristic:', List=Label, Icons=Icon) )
gdt.dialogWidgets.append( fieldLabeCombolWidget(Text='Tolerance value:', Circumference = [], List=[], Icons=[], ToolTip=[]) )
gdt.dialogWidgets.append( comboLabelWidget(Text='Datum system:', List=[]) )
gdt.dialogWidgets.append( comboLabelWidget(Text='Active annotation plane:', List=[]) )

class GeometricToleranceCommand:
   def __init__(self):
      self.iconPath = ':/dd/icons/geometricTolerance.svg'
      self.toolTip = 'Add Geometric Tolerance'
      self.dictionary = []
      for i in range(1,100):
         self.dictionary.append('GT'+str(i))
      self.idGDT = 3
      self.FeatureControlFrame = makeFeatureControlFrame()

   def Activated(self):
      ContainerOfData = makeContainerOfData()
      if getAnnotationObj(ContainerOfData):
         annotation = getAnnotationObj(ContainerOfData)
         self.toolTip = 'Add Geometric Tolerance to ' + annotation.Label
         gdt.dialogWidgets[3] = None
         if annotation.GT == []:
            gdt.dialogWidgets[1] = fieldLabeCombolWidget(Text='Tolerance value:', Circumference = ['',':/dd/icons/diameter.svg'], Diameter
= ContainerOfData.diameter, List=self.FeatureControlFrame.Label, Icons=self.FeatureControlFrame.Icon,
ToolTip=self.FeatureControlFrame.toolTip)
         else:
            if annotation.toleranceSelectBool:
               gdt.dialogWidgets[1] = fieldLabeCombolWidget(Text='Tolerance value:', Circumference = ['',':/dd/icons/diameter.svg'],
Diameter = annotation.diameter, tolerance = annotation.toleranceDiameter, List=self.FeatureControlFrame.Label,
Icons=self.FeatureControlFrame.Icon, ToolTip=self.FeatureControlFrame.toolTip)
            else:
               gdt.dialogWidgets[1] = fieldLabeCombolWidget(Text='Tolerance value:', Circumference = ['',':/dd/icons/diameter.svg'],
Diameter = annotation.diameter, toleranceSelect=False, lowLimit = annotation.lowLimit, highLimit = annotation.highLimit,
List=self.FeatureControlFrame.Label, Icons=self.FeatureControlFrame.Icon, ToolTip=self.FeatureControlFrame.toolTip)
      else:
         self.toolTip = 'Add Geometric Tolerance'
         showGrid()
         gdt.dialogWidgets[3] = comboLabelWidget(Text='Active annotation plane:', List=getAllAnnotationPlaneObjects())
         gdt.dialogWidgets[1] = fieldLabeCombolWidget(Text='Tolerance value:', Circumference = ['',':/dd/icons/diameter.svg'], Diameter =
ContainerOfData.diameter, List=self.FeatureControlFrame.Label, Icons=self.FeatureControlFrame.Icon,
ToolTip=self.FeatureControlFrame.toolTip)
      gdt.dialogWidgets[2] = comboLabelWidget(Text='Datum system:', List=[None]+getAllDatumSystemObjects())
      gdt.activate(idGDT = self.idGDT, dialogTitle=self.toolTip, dialogIconPath=self.iconPath, endFunction=self.Activated,
dictionary=self.dictionary)

   def GetResources(self):
      return {
         'Pixmap' : self.iconPath,
         'MenuText': self.toolTip,
         'ToolTip':  self.toolTip
         }

   def IsActive(self):
      if len(getObjectsOfType('AnnotationPlane')) == 0:
         return False
      if getSelection():
         for i in range(len(getSelectionEx())):
            if len(getSelectionEx()[i].SubObjects) == 0:
               return False
            for j in range(len(getSelectionEx()[i].SubObjects)):
```

```
                if getSelectionEx()[i].SubObjects[j].ShapeType == 'Face':
                    pass
                else:
                    return False
        else:
            return False
        return True

FreeCADGui.addCommand('dd_geometricTolerance', GeometricToleranceCommand())
```