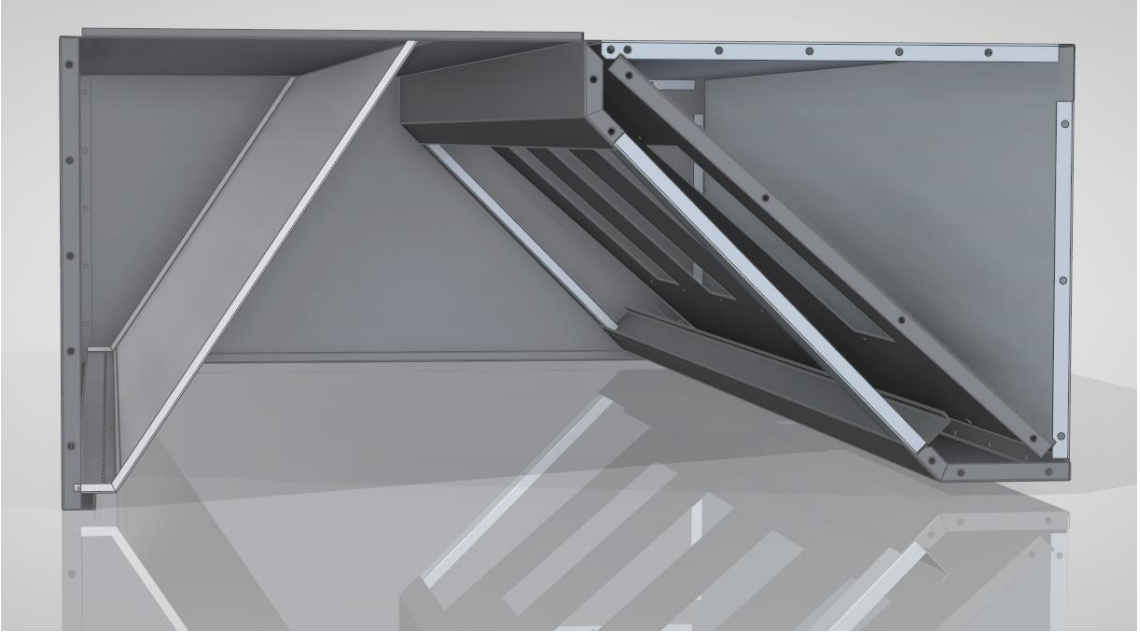
Escola Politècnica Superior d'Enginyeria de Vilanova i la Geltrú

UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANNEXOS



TÍTOL: **Disseny d'una Campana Industrial Segons les Especificacions del Client**

AUTORS: Banila, Sebastian Ionut

DATA DE PRESENTACIÓ: Juliol, 2024

SUMARI

13. ANNEXOS **3**

13.1. ANNEX GENÊRIC 3

13.1.1. Fitxes tècniques de les campanes estudiades 3

13.1.2 Informacions del software granta edupack 17

13.2. CODI COMPLET APLICAT AL PROGRAMA VISUAL STUDIO 26

13. ANNEXOS

13.1. ANNEX GENÊRIC

13.1.1. FITXES TÈCNIQUES DE LES CAMPANES ESTUDIADES

Campana de condensació. Luis Capdevila

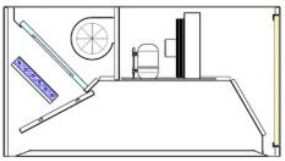
1. CAMPANAS DE EXTRACCIÓN. Condensación

REF: FTE-CO-ES-
Fecha Revisión: 03/

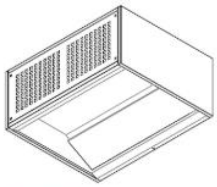
INFORMACIÓN TÉCNICA

Aplicaciones	Campana para retener el vapor de agua procedente de lava-vajillas, hornos que generen vapor u otros aparatos cuando no se dispone de una salida de ventilación exterior.
Formato	Pared. Altura de captación 650mm.
Material	Chapa acero Inoxidable AISI 304 de 1,2 y 1mm grosor.
Potencia unidad básica	750 W
Alimentación	230V 50Hz
Intensidad	4A
Nivel sonoro	60dBA
Temperatura	60º
Acabados	Campana soldada. Construida integramente en acero Inox 304 satinado. Sin tornillería ni remaches visibles desde el exterior. Ausencia de aristas o vértices cortantes en el exterior y en el interior.
Sujeción	Sujetado por cuatro varillas (6 en caso grandes dimensiones). Situado aproximadamente 50x50mm de cada esquina. Min diámetro recomendado M8.
Filtros	Separador de gotas 490x415x50mm AISI 304. Filtros de manta de salida 630x550x25mm. Manta de Carbón activo para reducir el olor (Opcional).
Drenaje	Canal de recogida periférico. Tapón de vaciado de agua 3/8 in.
Requerimientos de ventilación	Fueron tenidas en cuenta las siguientes directrices: DW172, NFPA 96, UL710 Y CTE DB SI.
Normativa	Conforme a normativa EN292, En61800-3, En 60335 y directiva de máquinas 89/392/ECC.

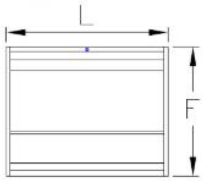
Vista Perfil interior *(Esquema elementos)*



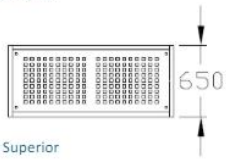
Vista Isométrica



Vista Inferior



Vista Frontal



Vista Superior

Medidas estándar

LONGITUD [mm]					FONDO [mm]		
1050	1500	2000	2500	3000	1000	1200	1500
Campana en un solo tramo							

* Para medidas especiales, consulte con su comercial.

1. CAMPANAS DE EXTRACCIÓN. Condensación

REF: FTE-CO-ES-01-01
Fecha Revisión: 03/02/20

Elementos y detalles de funcionamiento

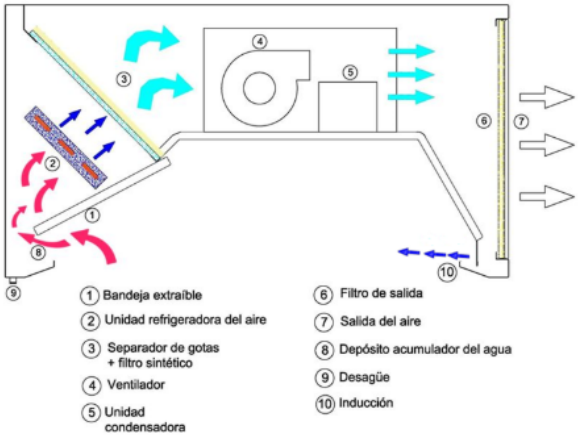
Descripción:

Para conseguir retener el máximo nivel de humedad, la campana dispone de 4 puntos de filtración y de una unidad de refrigeración que disminuye la temperatura del aire e incrementa la condensación del vapor. La campana se sirve de un flujo de inducción (10) frontal regulable que mejora la captación del vapor.

Proceso:

En primer lugar el aire sobresaturado de humedad es empujado por el flujo de inducción hacia la zona de captación. En la entrada de la zona de captación se fuerza el aire a entrar a alta velocidad y a realizar una curva abrupta que centrifuga una parte importante del contenido de humedad en forma de pequeñas gotas.

A continuación el aire es enfriado por una unidad refrigeradora (2) con el objetivo de condensar la mayor cantidad posible de agua que todavía se encuentra en fase vapor. A partir de ese punto, una etapa de filtración mediante un separador de gotas y una posterior mediante un filtro sintético retienen la práctica totalidad de la humedad en forma líquida (3).

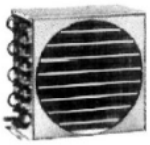


Seguidamente, el aire pasa por el ventilador y el condensador de la unidad refrigeradora y, finalmente, sale al exterior atravesando un último filtro de man filtrante (6). Todos los condensados descienden hasta el colector perimetral de donde pueden ser evacuados a través de los desagües previstos al efecto (9)

Evaporador: Intercambiador de calor donde se produce la transferencia de energía térmica.

Compresor hermético (R-134a)

Condensador de aire forzado



* En caso de medidas especiales y de gran tamaño se duplicaría todo el equipo de ventilación y unidad condensadora para la correcta adaptación a las necesidades del entorno.

Campana inductora. Luis Capdevila

1. CAMPANAS DE EXTRACCIÓN. INDUCTORA.

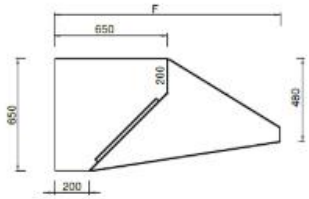
REF: FTE-KIN-ES-01-1022
Fecha Revisión: 14/11/2022

CARACTERÍSTICAS TÉCNICAS

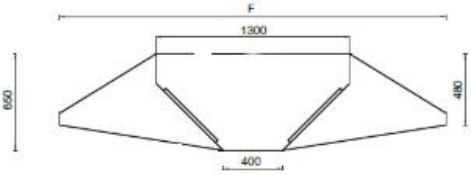
Aplicaciones	Extracción de humos. Campana dedicada a la restauración comercial y colectiva.
Formato	Pared o central. Altura de captación 650mm.
Acabados	Acero Inox. 304 satinadas en las partes visibles. Cara posterior y superior de acero galvanizado. Sin tornilleria ni remaches visibles desde el exterior. Laterales fijados por medio de pernos impactados. Ausencia de aristas o vértices cortantes en el exterior y en el interior.
Filtros	Filtros dinámicos SUPRA 490x490x25mm AISI 430 Filtros de malla y placas ciegas (opcional) bajo pedido. Filtros de carbón (anti olores) bajo pedido.
Drenaje	Canal de recogida periférico. Tapón de vaciado de la grasa 3/8 in.
Luminaria (Opcional)	Luminaria cilíndrica (1x36W o 2x36W) *Bajo pedido:* Tira LED 17W/m con cajón de acero inoxidable. Arco Led cromado [500LM / 5W] IP 65. Arco Led cromado [620LM / 8W] IP 65.
Normativa	Conforme a normativa EN292, EN 61800-3 y EN 60335.

Vista Lateral

Tipo Pared



Tipo Central



DESCRIPCIÓN

El principio básico de esta campana consiste en efectuar una aportación de aire, a temperatura ambiente, para compensar el aire aspirado. A diferencia de la campana clásica, requiere dos grupos de ventilación. El primero de extracción de aire y el segundo de aportación. El aire de aportación se introduce a través de la campana, convenientemente dirigido y a una velocidad constante, lo cual por inducción, provoca una depresión en la zona de humos orientándolos hacia los filtros. El caudal de aportación corresponde aproximadamente al 50 o 70 % del de aspiración.

- Plenums y campana (parte vista) en acero inoxidable AISI 304.
- Aislamientos internos en la sección de la zona de impulsión de aire. El conjunto de esta campana se compone de: • Plenum aspiración. • FILTRO DE LAMAS "SUPRA" INOX A304 de 490x490x25mm. • Plenum impulsión. • Difusor de aire. • Alabe orientador del aire impulsado. • Reglaje paso del aire.

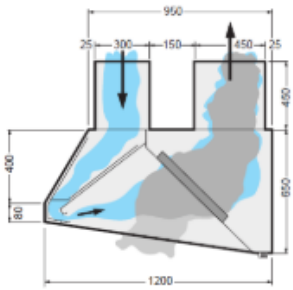
Al evitar cambios de temperatura en el interior del local, se hace especialmente indicada en zonas frías (montaña, etc.), en recintos climatizados y en instalaciones con exigencias de confort.

1. CAMPANAS DE EXTRACCIÓN. INDUCTORA.

REF: FTE-KIN-ES-01-1022
Fecha Revisión: 14/11/2022

Dimensiones estándard

Campana de pared													
Fondo [mm]	**1200**												
Largo [mm]	1000	1200	1700	2000	2200	2400	2700	2900	3400	3900	4400	5100	5600
Caudal de extracción [m3/h]	1500	1800	2700	3000	3200	3400	3800	4000	4500	5000	5400	6300	6900
Caudal de impulsión [m3/h]	1050	1260	1890	2100	2240	2380	2660	2800	3150	3500	3780	4410	4830

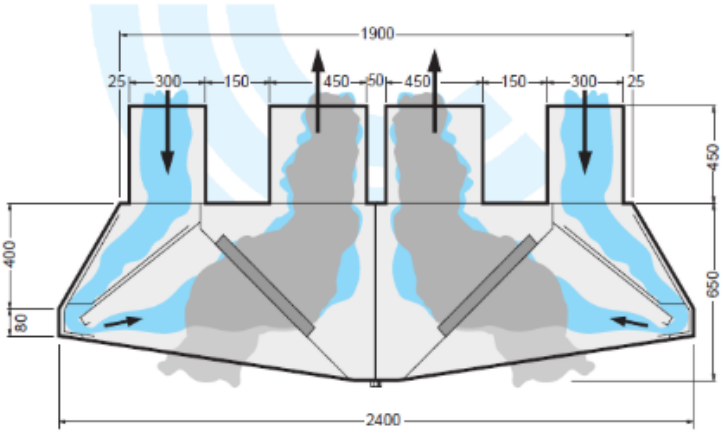


V. LATERAL - LATERAL V. - V. LATERALE

Caudal de impulsión
Drive Air-Flow
Débit d'aspiration

Caudal de extracción
Exhaust Air-Flow
Débit d'extraction

Campana central													
Fondo [mm]	**2400**												
Largo [mm]	1000	1200	1700	2000	2200	2400	2700	2900	3400	3900	4400	5100	5600
Caudal de extracción [m3/h]	3200	3600	4700	5400	6000	6800	7500	7800	8800	9700	11600	12500	13400
Caudal de impulsión [m3/h]	2240	2520	3290	3780	4200	4760	5250	5460	6160	6790	8120	8750	9380



Campana Kendak. Luis Capdevila

1. CAMPANAS DE EXTRACCIÓN. KENDAK

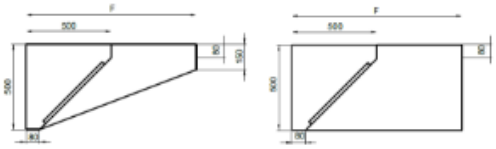
REF: FTE-KS-ES-01-0120
Fecha Revisión: 31/01/2022

INFORMACIÓN TÉCNICA

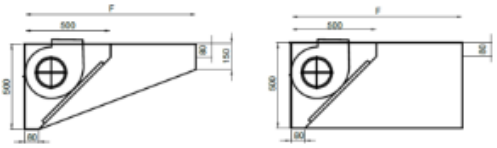
Aplicaciones	Extracción de humos. Campana dedicada a la restauración comercial y colectiva.
Formato	Pared o central. Altura de captación 500mm.
Material	Chapa acero Inoxidable AISI 304 de 1mm grosor.
Acabados	Chapas de acero 304 satinadas en las partes visibles. Cara posterior y superior de acero galvanizado. Sin tornillería ni remaches visibles desde el exterior. Laterales fijados por medio de pernos impactados. Ausencia de aristas o vértices cortantes en el exterior y en el interior.
Filtros	Filtros dinámicos SUPRA 490x490x25mm AISI 430 Filtros de malla y placas ciegas (opcional) bajo pedido.
Drenaje	Canal de recogida periférico. Tapón de vaciado de la grasa 3/8 in.
Plenum	Plenums de aportación y extracción (opcionales). Trampillas de regulación de caudal en las campanas con plenum.
Luminaria (opcional)	Luminaria integrada de bajo consumo [2x36W]. Luminaria Led [2x20W]. Con vidrio templado resistente al calor, bajo pedido. Aro Led cromado [500LM / 5W] IP 65. Aro Led cromado [620LM / 8W] IP 65.
Normativa	Conforme a normativa EN292, En61800-3 y En 60335

Vista lateral

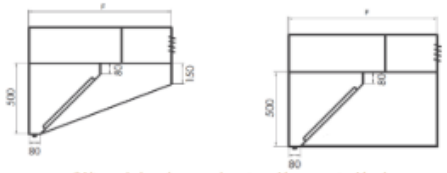
Pared modelo **Max** | Pared modelo **Box**



Modelo **Max** con ventilador | Modelo **Box** con ventilador



Max con plenum de aportación* | **Box** con plenum de aportación*



*Altura de los plenums de extracción y aportación de 250mm. Campanas a partir de 3m, plenum de 300mm.

ACCESORIOS / OPCIONALES

DESCRIPCIÓN	Código
Filtro SUPRA 490x490x25 AISI 430. (Incorporado de serie)	MASUPINX4
Filtros marco inox. malla galva de 490x490x25mm,	MAFILINX1
Filtros ciego AISI 304 de 490x490x50mm.	MAFILNC93
Regulador de vel. Monofásico de 3A con interruptor para luminaria.	CVACRV8
Regulador de vel. Monofásico 10A IP54.	CVACRV9
Incorporación de luminaria [2x36W] hasta 2500mm en campana.	-
Incorporación de luminaria LED [2x20W] hasta 2500mm.	-
Luminaria de vidrio resistente al calor 495mm [1x36W].	CE LUM VID 100 1
Luminaria de vidrio resistente al calor 995mm [2x36W].	CE LUM VID 100 2
Aro Led cromado 500LM, 5W, IP65, 6400K.	CE KLUM 6400K

- Incorporación ventilador (7/7 o 8/9 Tight)
- Incorporación de sistema de extinción de incendios
- Compensación con plenum superior
- Todo en Inoxidable 304 Satinado
- Salida trasera o lateral en campana de pared con ventilador
- Construcción a medida

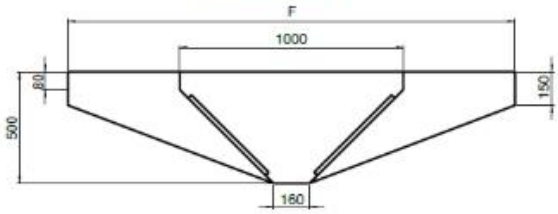
LUIS CAPDEVILA

1/3

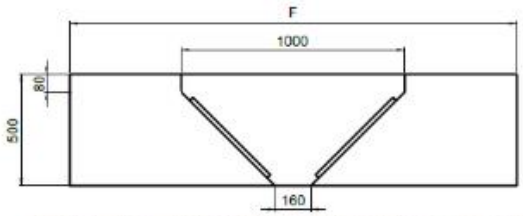
1. CAMPANAS DE EXTRACCIÓN. KENDAK

REF: FTE-KS-ES-01-0120
Fecha Revisión: 31/01/2022

Central Modelo **Max**



Central Modelo **Box**



Campana de pared KENDAK BOX con plenum de compensación instalada en una cocina.

*Pueden incluir adicionalmente ventilador, luminarias, plenums y aportación como la versión de pared.

MEDIDAS ESTÁNDAR

LONGITUD [mm]							FONDO [mm]		
Campana de Pared									
1000	1500	2000	2500	3000	3500	Medidas superiores, múltiplos de éstas	750	1000	1200
Campana en un solo tramo									
Campana Central									
1000	1500	2000	2500*[1]			Medidas superiores, múltiplos de éstas	1500	2000	2400
Campana en un solo tramo									

* Precios de medidas estándar incluidos en tarifa.
* Para medidas especiales, consulte con su comercial.

*1 L=2500mm central en una pieza hasta fondo 2000mm.

Campana KENDAK BOX de pared instalada en una cocina.

Campana Mòdul Retenidor Kendak. Luis Capdevila

2. MÓDULOS RETENEDORES DE GRASA. KENDAK.

REF: FTE-MKK-ES-01-1221
Fecha Revisión: 16/12/2021

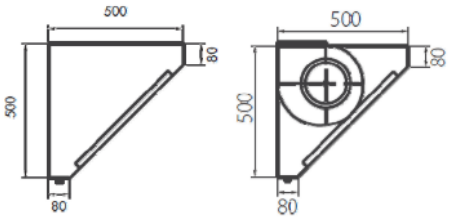
INFORMACIÓN TÉCNICA

Aplicaciones	Extracción y filtración de humos. Módulo ideal para la restauración comercial y colectiva. Permite formar conjuntos de filtros para la aspiración, retención y drenaje de grasas y vapores.
Formato	Pared o central. Altura de captación 650/685mm.
Material	Chapa acero Inoxidable AISI 304 de 1mm grosor.
Ventilador	Ventilador 7/7 4P opcional Motor protegido en carcasa y diseñado para extracciones de humos y vapores grasos. Puede instalarse ventilador totalmente regulable o de tres velocidades.
Acabados	Diseño moderno y funcional de cuidada estética. Sujeción segura del filtro por medio de guía con tope fijo, quedando de éste modo descartada cualquier posibilidad de caída accidental del filtro. Área de aspiración, sin intermedios. Máxima zona útil de aspiración. Ausencia de aristas o vértices cortantes en el exterior y en el interior. Chapas de acero 304 satinadas en las partes visibles. Sin tornillería ni remaches visibles desde el exterior.
Filtros	Filtros dinámicos SUPRA 490x490x25mm AISI 430 Filtros de malla y placas ciegas u otro tipo bajo pedido. Filtros de carbón (anti olores) bajo pedido. Filtro marco inox-malla galvanizado 490x490x25mm. Filtro placas dinámicas Inox.
Drenaje	Sistema de drenaje con gran capacidad de almacenamiento. Tapón de vaciado de grasa 3/8. Inalterable ante altas temperaturas.
Normativa	Conforme a normativa EN292, EN61800-3 y EN60335

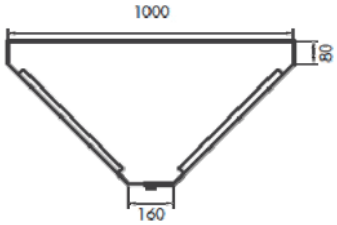
Módulo KENDAK

MEDIDAS VISTA LATERAL

Pared | Pared con ventilador



Central



Medidas estándar

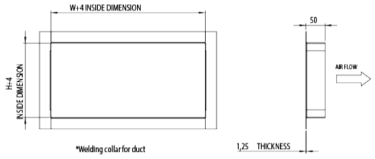
LONGITUD [mm]				FONDO [mm]	ALTO [mm]
Módulo de Pared: KENDAK—KENDAK con ventilador					
1000	1500	2000	2500	500	500
Módulo Central: KENDAK - KENDAK con ventilador					
1000	1500	2000	2500	1000	500

* Precios de medidas estándar incluidos en tarifa.

* Para medidas especiales, consulte con su comercial.

Campana KFM. Halton

KFM – Galley grease hood



Features

- The design follows USPHS guidelines
- Easily removable filters for cleaning
- High level of hygiene facilitated
- Prevention of the build-up of grease deposits, which pose a serious fire hazard
- With Halton Capture JetTM technology reducing the required exhaust airflow rate and improving the capture and containment efficiencies of the hood, while reducing energy use
- High-efficiency grease filtration using Halton KSA multi-cyclone filters
- Supplied as standard with lighting, balancing dampers for both capture and exhaust air and T.A.B.™ airflow measurement taps, which allow accurate and effective balancing of airflows, and efficient commissioning
- Stainless steel welded construction
- High-efficiency grease filtration using Halton KSA multi-cyclone filters

Specification

Halton KFM is a galley grease hood for use in marine & offshore applications to remove contaminated air released by cooking equipment. The KFM hood uses Halton Capture Jet™ technology, which allows the hood to operate with up to 30% lower exhaust airflow rates than traditional hoods.

Weights of standard KFM hoods (kg)

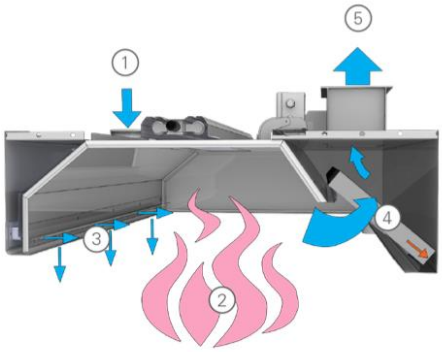
B/L	1000	1500	2000	2500	3000
1100	40	60	80	100	110
1300	46	66	86	106	118
1500	51	71	92	111	126
1700	57	77	98	117	134
1900	63	83	103	123	142

The above table represents an indication of different size of average KFM hoods without Capture Jet technology. Weight does not include the fire damper.

Weights of standard KFM hoods with Capture Jet Technology (kg)

B/L	1000	1500	2000	2500	3000
1100	50	74	100	124	138
1300	56	80	105	129	146
1500	61	86	111	135	154
1700	67	92	117	141	162
1900	73	97	122	147	170

The above table represents an indication of different size of average KFM hoods with Capture Jet technology. Weight does not include fire damper.



1. Supply air enters the Capture JetTM plenum.
2. Contaminated air and heat rises from the cooking appliances.
3. Contaminated air is directed into the hood by Halton patented Capture JetTM technology.
4. KSA multi-cyclone filters remove grease and contaminants from the air stream with the aid of centrifugal effect. According to independent laboratory tests KSA is the most efficient mechanical grease filter on the market.
5. Filtered exhaust air.

Recommended exhaust airflow for KFM

RECOMMENDED EXHAUST AIRFLOW				
NUMBER OF KSA FILTERS	MINIMUM l/s	MAXIMUM l/s	MINIMUM m3/h	MAXIMUM m3/h
1	130	201	468	724
2	259	402	932	1447
3	389	602	1400	2167
4	518	803	1865	2891
5	648	1004	2333	3614
6	778	1205	2801	4338

Note: KSA filter size 500x330x50 mm

Campana UWI. Halton

UWI – Capture Ray™ and Water Wash hood (CE)

Overview

UWI hoods are Halton's "all in one" solution, resulting in some unique benefits. It is a combination of the technologies and benefits of UVI and KWI hoods that make it one of the most efficient kitchen ventilation solutions.

It is suitable for projects and can be used in all closed, open or show kitchens particularly those with an intense level of activity (central kitchens, hotels, hospitals, etc).

The Capture Ray™ technology neutralises the grease carried by the exhausted air and also drastically reduces the odours emissions for the kitchen while the Water Wash automatically carries out the regular cleaning of the filters, with no outside intervention necessary.

UWI hoods are also equipped with the Capture Jet™ technology.

Considerable energy savings

- Up to 40% reduction in exhaust airflow rates due to **Capture Jet™** technology.
- Possibility to extend the airflows reduction to up to 64% with **M.A.R.V.E.L.** energy savings technology.

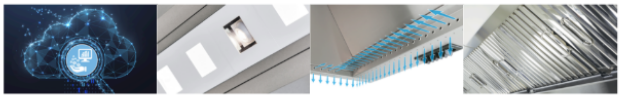
Improved safety, maintenance savings and respect for the neighborhood

- **KSA cyclonic aerosol separators** complying with EN 16282-6, 95% efficient on 10 microns particles or larger (also UL, NSF and LPS certified).
- Neutralisation of remaining grease particles and vapors after mechanical filtration thanks to

Other features

- HACCP certified (Hazard Analysis Critical Control Point).
- No need for an extra duct for the **Capture Jet™** technology.
- Capture Jets are automatically switched off when the hood is not used or operates at a minim airflow.
- Requires CCW control cabinet for the automatic washing cycles management.
- Equipped with **Halton Connect** IoT (Internet of Things) platform with advanced 24/7 distant monitoring capabilities.
- Highest value of ownership thanks to **Halton Connect & Care** smart services available as an option from kitchens commissioning.
- **Halton Skyline** LED culinary light provides the best visual comfort while contributing to further improve the safety and the energy savings.
- When extended to the whole kitchen, the Human Centric version of **Halton Skyline** directly contributes to chefs' and their teams Wellbeing.
- Exhaust airflow rates determined with a EN 16282-1 based calculation method taking into account the loads of the cooking appliances, the configuration of the extract system and its capture and containment efficiency.
- Capture and containment efficiency tested in accordance with the ASTM 1704 standard.
- Quick and easy commissioning. Hoods delivered "ready to install", with all accessories included, such as light fitting, T.A.B.™ taps and dampers for quick balancing on-site.
- Sturdier and easier to clean- Less parts and less joints. Stainless steel construction.

More about the main embedded technologies



Halton Connect | Halton Skyline | Capture Jet™ | KSA Filters

Campana Eco-Plus. Altex

SISTEMA DE FILTRACIÓN

Con porta filtros preparados para alojar diferentes tipos de filtro de 490x490x50 (según norma DIN 4102, Efficiency EN 779:2012 hasta un 95% (según tipo):

- Por **condensación**: filtros de malla de material galvanizado tipo déployé, con doble zigzag, que confieren a los mismos una mayor superficie de contacto por filtro y retícula de media luna que procura una mayor resistencia a la saturación y un mejor drenaje de las condensaciones (40Pa).
- Por **inercia (dinámicos)**: filtros de lamas de acero inox con forma trapezoidal de las mismas, que confieren a estos una mejor filtración con un mayor margen de velocidades de paso de aire, ya que, sus pronunciadas aristas imprimen en el aire una inercia incapaz de seguir por las partículas de grasa en suspensión consiguiendo así mismo una baja pérdida de carga (44 Pa).
- **Mixto**: consiste en un filtro de 490x490x25 de malla, de iguales características que los anteriores de condensación y posterior a éste, un filtro de 490x490x25 mm de carbón activo, que contiene en su interior carbón activo en forma de manta impregnada cuya porosidad permite atrapar en ellos las partículas en suspensión causantes de los olores.

IDEAL PARA TECHOS DE BAJA ALTURA

OPCIONALMENTE CON VENTILADOR

UV — Posibilidad de insertar filtración mediante UV

LED — Posibilidad de instalar luminarias integradas

Posibilidad de instalar regulación automática de caudal

Posibilidad de instalar un sistema de extinción de incendios

Campanas **ECO PLUS**, construida en todas sus partes vistas de acero inoxidable con acabado pulido fino, siendo unas de las más económicas y competitivas del mercado y ofreciendo al mismo tiempo una gran calidad.

Forma de campana especialmente estudiada para dar al conjunto la máxima rigidez, con la mínima altura para mantener, según normativa, los filtros en ángulo de inclinación entre el 45º y 60º sobre la horizontal. Laterales con inclinación y frontal de 150mm, ideal para cocinas con techo de poca altura.

Dispone de un nuevo sistema de sujeción que facilita su instalación, haciendo posible el desplazamiento lateral, hasta conseguir el anclaje a pared perfecto.

El ensamblaje de todas las piezas vistas de la campana se realiza mediante pernos soldados por la parte interna, dando así la posibilidad de desmontar el conjunto si fuera necesario (para entrarla por pasos reducidos o para ampliarla) y un gran acabado, al no ser visibles remaches ni tornillos.

La campana versión **Monobloc** incorpora ventilador con motor incorporado.

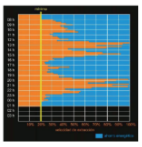
Diseñada para incorporar filtros de 490x490x50

SISTEMA DE DRENAJE

Sistema de drenaje para recoger las grasas retenidas por el colector de filtros, con salida mediante tapón metálico roscado. Canal perimetral que recoge las condensaciones del interior del voladizo o visera y laterales.

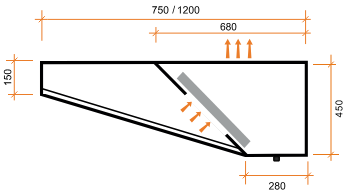
EFICIENCIA ENERGÉTICA

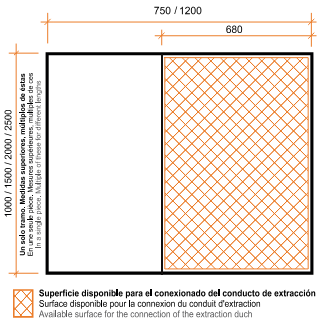
Reducción de hasta un 80% en el consumo energético (en función del uso de la cocina), instalando nuestro exclusivo sistema de regulación automático de caudal (**ACC**).



ECO PLUS

LONGITUDES / LONGUEURS / LENGTHS					FONDO/ PROFONDEUR/ DEPTH	
1000	1500	2000	2500	**Medidas superiores, múltiplos de estas** Mesures supérieures, multiples de ces Multiple of these for different lengths	750	1200
En un solo tramo En une seule pièce In a single piece						





Campana Reversible. Altex

COCINAS DE GRAN PRODUCCIÓN

CONSTRUIDA TOTALMENTE EN ACERO INOXIDABLE AISI 304

INCORPORA PLENUM DE REGULACIÓN INTERIOR

TRAMPILLA DE REGULACIÓN INDEPENDIENTES PARA CADA FILTRO

FALDONES RECTOS

Posibilidad de insertar filtración mediante UV

Posibilidad de instalar luminarias integradas

Posibilidad de instalar regulación automática de caudal

Posibilidad de instalar un sistema de extinción de incendios

Campana industrial **REVERSIBLE** extractora de humos de faldones rectos pared o central, compuesta por un envolvente de forma cúbica construido en acero inoxidable en acabado pulido fino, menos las trampillas de regulación.

Colector pared, central o doble pared, construido igualmente en acero inoxidable con filtros en 45º según normativa y bandeja recoge grasas con tapón de drenaje metálico.

Canal perimetral de recogida de condensaciones de vahos grasos en faldones.

Recomendada para cocinas de gran producción, especialmente diseñada para elementos de cocción como sartenes basculantes, marmitas y hornos de todo tipo, pared o central, construida con faldones y techos de acero inoxidable.

Dispone de sistema de fijación incorporado para facilitar su montaje.

Esta campana incorpora plenum interior de aspiración con regulación de caudal filtro a filtro.

El ensamblaje de todas las piezas vistas de la campana se realiza mediante pernos soldados por la parte interna, dando así la posibilidad de desmontar el conjunto si fuera necesario (para entrarlo por una puerta pequeña ó para ampliarlo), y un gran acabado, al no ser visibles remaches ni tornillos.

Máxima seguridad con canal perimetral de refuerzo en forma de "U "con chapa doblada en punto redondo para evitar cortes.

SISTEMA DE FILTRACIÓN

Con porta filtros preparados para alojar diferentes tipos de filtro de 390x490x50 (según norma DIN 4102, Efficiency EN 779:2012 hasta un 95% (según tipo):

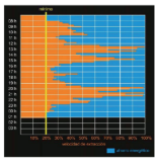
- Por **condensación:** filtros de malla de material galvanizado tipo déployé, con doble zig-zag, que confieren a los mismos una mayor superficie de contacto por filtro y retícula de media luna que procura una mayor resistencia a la saturación y un mejor drenaje de las condensaciones (40Pa).
- Por **inercia** (dinámicos): filtros de lamas de acero inox con forma trapezoidal de las mismas, que confieren a estos una mejor filtración con un mayor margen de velocidades de paso de aire, ya que, sus pronunciadas aristas imprimen en el aire una inercia incapaz de seguir por las partículas de grasa en suspensión consiguiendo así mismo una baja perdida de carga (44 Pa).
- **Mixto:** consiste en un filtro de 390x490x25 de malla, de iguales características que los anteriores de condensación y posterior a éste, un filtro de 390x490x25 mm de carbón activo, que contiene en su interior carbón activo en forma de manta impregnada cuya porosidad permite atrapar en ellos las partículas en suspensión causantes de los olores.

SISTEMA DE DRENAJE

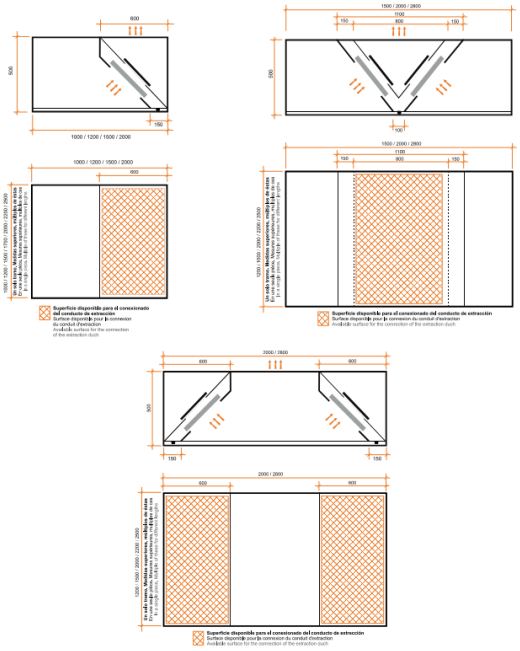
Doble sistema de drenaje, uno para recoger las grasas retenidas por el colector de filtros, con depósito de gran capacidad y salida por tapón metálico roscado y otro para el canal perimetral que recoge las condensaciones del interior del voladizo.

EFICIENCIA ENERGÉTICA

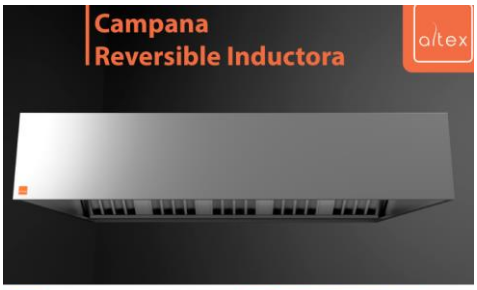
Reducción de hasta un 80% en el consumo energético (en función del uso de la cocina), instalando nuestro exclusivo sistema de regulación automático de caudal (**ACC**).



LONGITUDES / LONGUEURS / LENGTHS							**FONDO**/ PROFONDEUR/ DEPTH
PARED / MURALE / WALL							
1000	1200	1500	2000	2200	2500	**Medidas superiores, múltiplos de estas** Mesures supérieures, multiples de ces Multiple of these for different lengths	1000 1200 1500 2000
En un solo tramo En une seule pièce In a single piece							
CENTRAL / CENTRALE / ISLAND							
1200	1500	2000	2200	2500		**Medidas superiores, múltiplos de estas** Mesures supérieures, multiples de ces Multiple of these for different lengths	1500 2000 2800
En un solo tramo En une seule pièce In a single piece							



Campana Reversible inductora. Altex



COCINAS DE GRAN PRODUCCIÓN

INOX CONSTRUIDA TOTALMENTE EN ACERO INOXIDABLE AISI 304

INCORPORA PLENUM DE REGULACIÓN INTERIOR

TRAMPILLA DE REGULACIÓN INDEPENDIENTES PARA CADA FILTRO

SISTEMA EXCLUSIVO DE BÓVEDA DE INDUCCIÓN

UV Posibilidad de insertar filtración mediante UV

LED Posibilidad de instalar luminarias integradas

MÁXIMA EFICIENCIA MEDIOAMBIENTAL

Posibilidad de instalar regulación automática de caudal

Posibilidad de instalar un sistema de extinción de incendios

Campana industrial extractora de humos de faldones rectos modelo **REVERSIBLE INDUCTORA**, pared o central, compuesta por un envolvente de forma cúbica construido en acero inoxidable AISI 304, en acabado pulido fino.

Colector pared, central o doble pared, construido igualmente en acero inoxidable AISI 304 con filtros en 45º según normativa y bandeja recoge grasas con tapón de drenaje metálico.

Canal perimetral de recogida de condensaciones de vahos grasos en faldones.

Recomendada para cocinas de gran producción, especialmente diseñada para elementos de cocción como sartenes basculantes, marmitas y hornos de todo tipo, pared o central, construida con faldones y techos de acero inoxidable.

Dispone de sistema de fijación incorporado para facilitar su montaje.

Esta campana incorpora plenum interior de aspiración con regulación de caudal filtro a filtro.

Asimismo, incorpora interiormente en su parte frontal superior, plenum de aportación de aire con salidas al interior de la cúpula de inducción.

El ensamblaje de todas las piezas vistas de la campana se realiza mediante pernos soldados por la parte interna, dando así la posibilidad de desmontar el conjunto si fuera necesario (para entrarlo por una puerta pequeña ó para ampliarlo), dando así mismo una gran belleza de líneas, al no ver remaches ni tornillos...

Máxima seguridad con canal perimetral de refuerzo en forma de "U "con chapa doblada en punto redondo para

SISTEMA DE FILTRACIÓN

Con porta filtros preparados para alojar diferentes tipos de filtro de 390x490x50 (según norma DIN 4102, Efficiency EN 779:2012 hasta un 95% (según tipo):

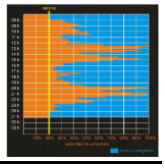
- Por **condensación:** filtros de malla de material galvanizado tipo déployé, con doble zig-zag, que confieren a los mismos una mayor superficie de contacto por filtro y retícula de media luna que procura una mayor resistencia a la saturación y un mejor drenaje de las condensaciones (40Pa).
- Por **inercia** (dinámicos): filtros de lamas de acero inox con forma trapezoidal de las mismas, que confieren a estos una mejor filtración con un mayor margen de velocidades de paso de aire, ya que, sus pronunciadas aristas imprimen en el aire una inercia incapaz de seguir por las partículas de grasa en suspensión consiguiendo así mismo una baja perdida de carga (44 Pa).
- **Mixto:** consiste en un filtro de 390x490x25 de malla, de iguales características que los anteriores de condensación y posterior a éste, un filtro de 390x490x25 mm de carbón activo, que contiene en su interior carbón activo en forma de manta impregnada cuya porosidad permite atrapar en ellos las partículas en suspensión causantes de los olores.

SISTEMA DE DRENAJE

Doble sistema de drenaje, uno para recoger las grasas retenidas por el colector de filtros, con depósito de gran capacidad y salida por tapón metálico roscado y otro para el canal perimetral que recoge las condensaciones del interior del voladizo.

EFICIENCIA ENERGÉTICA

Reducción de hasta un 80% en el consumo energético (en función del uso de la cocina), instalando nuestro exclusivo sistema de regulación automático de caudal (**ACC**).



LONGITUDES / LONGUEURS / LENGTHS

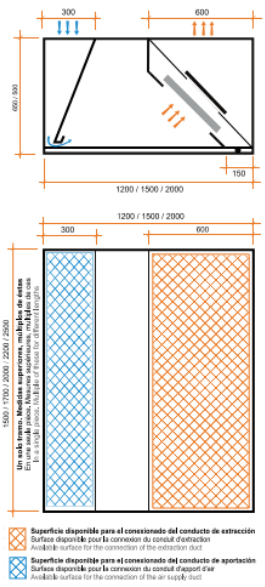
PARED / MURALE / WALL

1000	1200	1500	2000	2200	2500		
En un solo tramo En une seule pièce In a single piece						**Medidas superiores, múltiplos de estas** Mesures supérieures, multiples de ces Multiple of these for different lengths	1000 1200 1500 2000

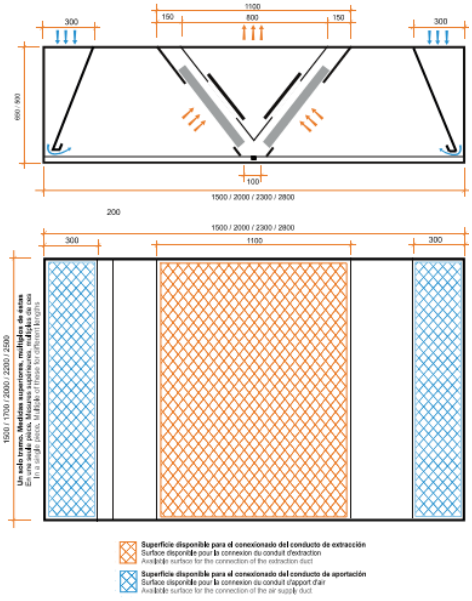
CENTRAL / CENTRALE / ISLAND

1200	1500	2000	2200	2500		
En un solo tramo En une seule pièce In a single piece					**Medidas superiores, múltiplos de estas** Mesures supérieures, multiples de ces Multiple of these for different lengths	1500 2000 2300 2800

Pared / Murale / Wall

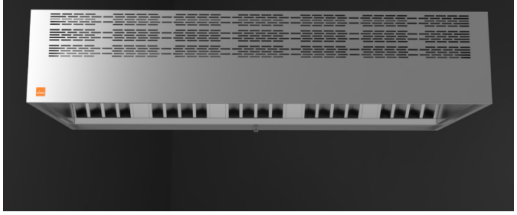


Central / Centrale / Island



Campana Airdoor Compensada. Altex

Campana Airdoor Compensada



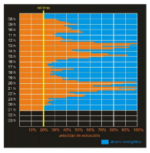
- SISTEMA DE INDUCCIÓN PERIMETRAL
- INCORPORA PLENUM INTERIOR CON TRAMPILLAS DE REGULACIÓN
- AHORRO DE HASTA UN 35% EN CONSUMO
- CORTINA DE AIRE INTEGRADA
- MÁXIMA EFICIENCIA ENERGÉTICA
- UV — Posibilidad de insertar filtración mediante UV
- LED — Posibilidad de instalar luminarias integradas
- Posibilidad de instalar regulación automática de caudal
- Posibilidad de instala sistema de extinción incendios

MÁXIMA EFICIENCIA ENERGÉTICA
- Ahorro de hasta un 35% en el consumo eléctrico
- Reducción del tamaño de las chimeneas y del extractor
- Rápida amortización
- Reciclable

MEJORA 3 ASPECTOS CLAVE EN RELACIÓN A LOS MODELOS CONVENCIONALES:

1. EFICIENCIA ENERGÉTICA

Utiliza una novedosa tecnología que reduce entre un 35% y un 40% del caudal de extracción y de aportación que combinado con nuestro exclusivo sistema de regulación automático de caudal (ACC), puede llegar a reducir en un 65% los caudales respecto a las extracciones convencionales, disminuyendo los costes de instalación, el consumo energético y el impacto medioambiental.



Campana Airdoor Compensada

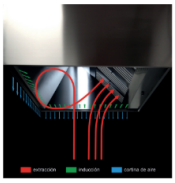
2. PRESTACIONES

Especialmente efectiva en cocciones donde se generan **grandes cantidades de humo y altas temperaturas** (barbacoas, parrillas, asadores de pollos, hornos, tanto de leña como carbón), ya que retiene totalmente, dentro su zona de captación tanto el humo como la temperatura que generan estos tipos de cocción.

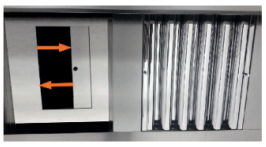
3. CONFORT DEL COCINERO

Con su exclusivo sistema de inducción diagonal consigue un ambiente puro en la cocina y elimina los molestos excesos de temperatura, ya que estos son dirigidos hacia los filtros, apartándolos del cocinero y como consecuencia aumenta la calidad y el confort en la zona de trabajo.

Estas mejoras se consiguen mediante:



- La extracción convencional:
  Extracción de los humos producidos por los elementos de cocción que se dirigen al colector de filtros gracias a la caja de extracción a distancia..
- La inducción horizontal perimetral:
  La línea de difusores horizontales empujan los humos hacia el colector de filtros.
- La cortina de aire perimetral:
  La línea de difusores verticales evitan que los humos se escapen del área de cocción.
- La **aportación de aire**
  A través del deflector situado en la parte frontal, se aporta aire procedente del exterior al local.



Plenum interior de aspiración con trampillas de Regulación de caudal filtro a filtro

La inducción horizontal y la cortina de aire exclusivas de este modelo, se pueden alimentar de aire limpio de dos formas:

- Con ventilador interno de aportación incorporado (este sistema evita la necesidad de instalar conductos de aportación).
- O en aquellas instalaciones en las que se necesita aportación extra de aire (cocinas herméticas o climatizadas) donde el aire se recoge del exterior mediante conducciones como en las aportaciones convencionales.

Primer plano de cortina de aire

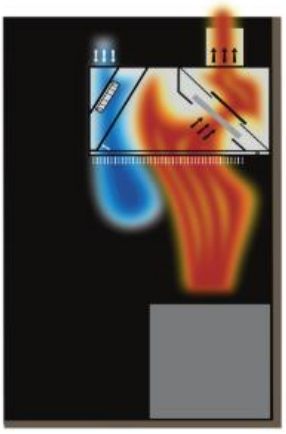
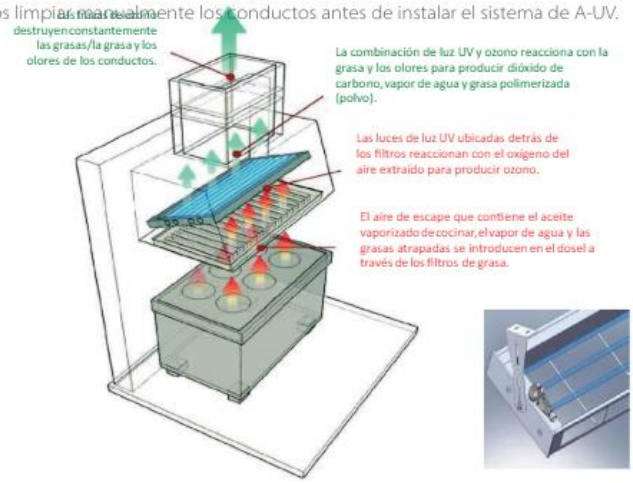
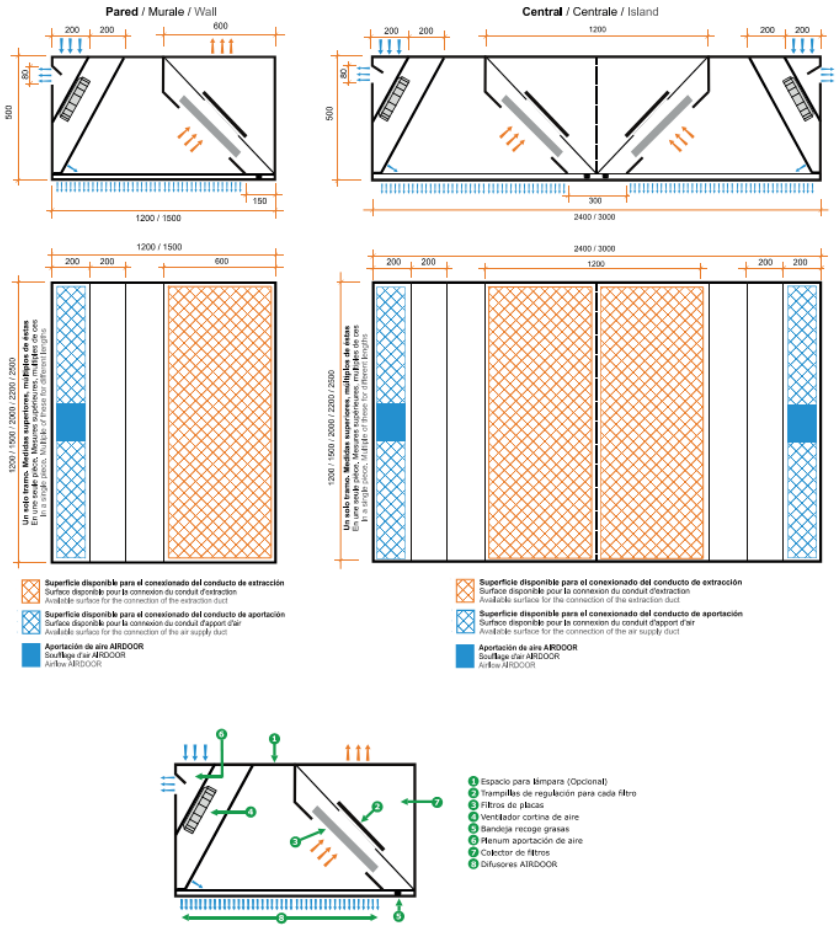


Diagrama de flujo

•Las lámparas se colocan en un armazón de acero, que se instala detrás de los filtros de grasa de la campana extractora o, en caso de que esto no sea posible por falta de espacio, en un recinto justo encima de la campana. La exposición a luz A-UV intensa y la oxidación por ozono provocan la destrucción de los contaminantes del aire, lo que desemboca en la reducción de las emisiones de olor al entorno y la ausencia de depósitos de grasa en los conductos. Este proceso reduce el olor que se emite al entorno. Al mismo tiempo, se genera una pequeña cantidad de exceso de ozono para mantener los conductos limpios y destruir los depósitos de grasa que ya existían en ellos.

Recomendamos limpiar manualmente los conductos antes de instalar el sistema de A-UV.





13.1.2 INFORMACIONS DEL SOFTWARE GRANTA EDUPACK

Informació acer inoxidable

GRANTA EduPack

Acero inoxidable

Descripción

Figura

Leyenda

1. Tostadora Siemens en acero inoxidable austenítico pulido (por Porsche Design) © Granta Design; 2. Tijeras en acero inoxidable ferrítico, que es magnético en contraste con el inoxidable austenítico © Granta Design

Material

Los aceros inoxidables son aleaciones de hierro, cromo, níquel, y a menudo cuatro o cinco elementos adicionales. La aleación transmuta el acero al carbono normal, que se oxida, y es propenso a la fragilidad por debajo de la temperatura ambiente, en un material que supera estas limitaciones. De hecho, la mayoría de los aceros inoxidables resisten la corrosión en entornos normales, y siguen siendo dúctiles a bajas temperaturas.

Composición (resumen)

Fe/<0.25C/16 - 30Cr/3.5 - 37Ni/<10Mn + Si,P,S (+N para la serie 200)

Propiedades generales

Densidad	7,61e3	-	7,87e3	kg/m^3
Precio	* 2,71	-	2,91	EUR/kg
Fecha de primer uso ("-" significa AC)	1915			

Propiedades mecánicas

Módulo de Young	190	-	210	GPa
Módulo de cortante	74	-	82	GPa
Módulo en volumen	140	-	160	GPa
Coeficiente de Poisson	0,27	-	0,28	
Límite elástico	257	-	1,14e3	MPa
Resistencia a tracción	515	-	1,3e3	MPa
Resistencia a compresión	* 252	-	1,2e3	MPa
Elongación	10	-	49	% strain
Dureza-Vickers	170	-	438	HV
Resistencia a fatiga para 10 ^ 7 ciclos	* 256	-	542	MPa
Tenacidad a fractura	* 57	-	137	MPa.m^0.5
Coeficiente de pérdida mecánica (tan delta)	* 3,1e-4	-	0,0012	

Propiedades térmicas

Punto de fusión	1,4e3	-	1,49e3	°C
Máxima temperatura en servicio	640	-	747	°C

Values marked * are estimates.
ANSYS Granta no ofrece garantías para estos datos.

Producción de materia prima: CO2, energía y agua				
Contenido en energía, producción primaria	* 69,1	-	76,2	MJ/kg
Huella de CO2, producción primaria	* 5,18	-	5,71	kg/kg
Agua consumida	* 130	-	140	l/kg
Procesado de material: energía				
Energía en fundición	* 10,6	-	11,7	MJ/kg
Energía de extrusión, laminado en hoja	* 9,45	-	10,4	MJ/kg
Energía de perfilado, forja	* 4,9	-	5,4	MJ/kg
Energía de trefilado	* 34,9	-	38,4	MJ/kg
Energía en sinterización	* 36,8	-	40,6	MJ/kg
Energía de vaporización	* 1,14e4	-	1,26e4	MJ/kg
Energía de desbaste (p/u peso eliminado)	* 1,12	-	1,24	MJ/kg
Energía de mecánizado final (p/u peso eliminado)	* 6,95	-	7,66	MJ/kg
Energía de lijado (p/u peso eliminado)	* 13,5	-	14,9	MJ/kg
Energía en el procesado no convencional (p/u peso eliminado)	* 114	-	126	MJ/kg
Procesado de material: huella de CO2				
CO2 en colada	* 0,797	-	0,879	kg/kg
CO2 en extrusión, laminado en hoja	* 0,709	-	0,781	kg/kg
CO2 en perfilado, forja	* 0,367	-	0,405	kg/kg
CO2 en trefilado	* 2,62	-	2,88	kg/kg
CO2 en sinterización	* 2,9	-	3,2	kg/kg
CO2 en vaporización	* 857	-	945	kg/kg
CO2 en desbaste (p/u peso eliminado)	* 0,0841	-	0,0928	kg/kg
CO2 en mecánizado final (p/u peso eliminado)	* 0,521	-	0,575	kg/kg
CO2 en lijado (p/u peso eliminado)	* 1,01	-	1,12	kg/kg
CO2 en procesado no convencional (p/u peso eliminado)	* 8,57	-	9,45	kg/kg
Reciclado del material: energía, CO2 y fracción reciclable				
Reciclaje	✓			
Contenido en energía, reciclado	* 15,2	-	16,7	MJ/kg
Huella de CO2, reciclado	* 1,19	-	1,31	kg/kg
Fracción reciclable en suministro habitual	36	-	39	%
Reciclado inferior	✓			
Combustión para recuperar energía	✗			
Vertedero	✓			
Biodegradable	✗			
Ratio de toxicidad	No toxico			
Fuente renovable	✗			

Aspectos Medioambientales

Los aceros inoxidables están aprobados por la FDA (US Food and Drug Association), de hecho, son tan inertes que pueden ser implantados en el cuerpo, y son ampliamente utilizados en equipos de procesamiento de alimentos. Además, se pueden reciclar.

Informació acer galvanitzat

General information

Overview

The galvanization of steel enhances its corrosion properties by forming a protective coating of zinc. In the event of water coming into contact with the galvanized steel, an electrochemical cell is set up and this zinc layer corrodes sacrificially instead of the steel beneath it. Hot dip galvanized coatings are readily available on all standard and most proprietary grades of steels. This datasheet describes the properties of galvanized steel in which the substrate is the low carbon steel AISI 1015.

Designation

Galvanized steel (coated steel)

Typical uses

Wall cladding, roof covering.

Included in Materials Data for Simulation	✓

Composition overview

Compositional summary

Fe93-96 (impurities: Mn<1.2, C<0.24, Si<0.24, P<0.048)
Coating: Zn4-5

Form	Other
Material family	Metal (ferrous)
Base material	Fe (Iron)

Composition detail (metals, ceramics and glasses)

C (carbon)	0	-	0,241	%
Fe (iron)	93,2	-	96	%
Mn (manganese)	0	-	1,25	%
P (phosphorus)	0	-	0,0483	%
Si (silicon)	0	-	0,241	%
Zn (zinc)	4	-	5	%

Price

Price	* 0,81	-	0,94	EUR/kg
Price per unit volume	* 6,32e3	-	7,42e3	EUR/m^3

Physical properties

Density	7,8e3	-	7,9e3	kg/m^3

Mechanical properties

Young's modulus	200	-	215	GPa
Specific stiffness	25,5	-	27,4	MN.m/kg
Yield strength (elastic limit)	250	-	395	MPa
Tensile strength	420	-	600	MPa
Specific strength	31,8	-	50,3	kN.m/kg
Elongation	26	-	47	% strain
Tangent modulus	989			MPa
Compressive modulus	* 200	-	215	GPa
Compressive strength	* 250	-	395	MPa
Flexural modulus	200	-	215	GPa

Primary production energy, CO2 and water				
Embodied energy, primary production Sources 40 MJ/kg (Hammond and Jones, 2008)	38,1	-	42	MJ/kg
CO2 footprint, primary production Sources 3.01 kg/kg (Hammond and Jones, 2008)	2,87	-	3,16	kg/kg
Water usage	* 56,3	-	62,2	l/kg
Processing energy, CO2 footprint & water				
Casting energy	* 12	-	13,3	MJ/kg
Casting CO2	* 0,9	-	0,994	kg/kg
Casting water	* 22,7	-	34,1	l/kg
Roll forming, forging energy	* 2,82	-	3,12	MJ/kg
Roll forming, forging CO2	* 0,212	-	0,234	kg/kg
Roll forming, forging water	* 2,76	-	4,14	l/kg
Extrusion, foil rolling energy	* 5,36	-	5,92	MJ/kg
Extrusion, foil rolling CO2	* 0,402	-	0,444	kg/kg
Extrusion, foil rolling water	* 3,84	-	5,76	l/kg
Wire drawing energy	* 19,3	-	21,3	MJ/kg
Wire drawing CO2	* 1,45	-	1,6	kg/kg
Wire drawing water	* 7,27	-	10,9	l/kg
Metal powder forming energy	* 38,9	-	42,8	MJ/kg
Metal powder forming CO2	* 3,11	-	3,43	kg/kg
Metal powder forming water	* 42,4	-	63,5	l/kg
Vaporization energy	* 1,71e4	-	1,89e4	MJ/kg
Vaporization CO2	* 1,28e3	-	1,42e3	kg/kg
Vaporization water	* 7,13e3	-	1,07e4	l/kg
Coarse machining energy (per unit wt removed)	* 0,855	-	0,945	MJ/kg
Coarse machining CO2 (per unit wt removed)	* 0,0641	-	0,0709	kg/kg
Fine machining energy (per unit wt removed)	* 4,28	-	4,73	MJ/kg
Fine machining CO2 (per unit wt removed)	* 0,321	-	0,355	kg/kg
Grinding energy (per unit wt removed)	* 8,08	-	8,93	MJ/kg
Grinding CO2 (per unit wt removed)	* 0,606	-	0,67	kg/kg
Non-conventional machining energy (per unit wt removed)	* 171	-	189	MJ/kg
Non-conventional machining CO2 (per unit wt removed)	* 12,8	-	14,2	kg/kg
Recycling and end of life				
Recycle	✓			
Embodied energy, recycling	* 9,53	-	10,5	MJ/kg
CO2 footprint, recycling	* 0,747	-	0,826	kg/kg
Recycle fraction in current supply	52,3	-	57,8	%
Downcycle	✓			
Combust for energy recovery	✗			
Landfill	✓			
Biodegrade	✗			

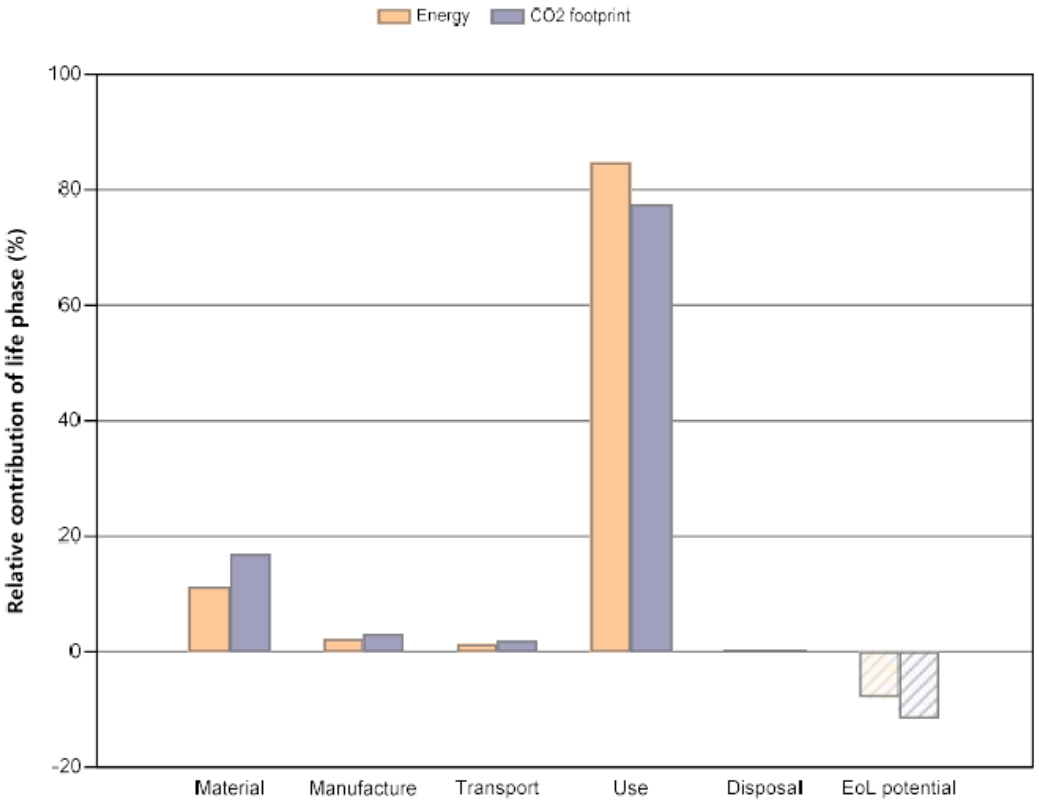
Report de l'aplicació de l'eina Eco-Audit

GRANTA EduPack

Eco Audit Report

Product name	Product
Country of use	Spain
Product life (years)	10

Summary:



Energy details CO2 footprint details

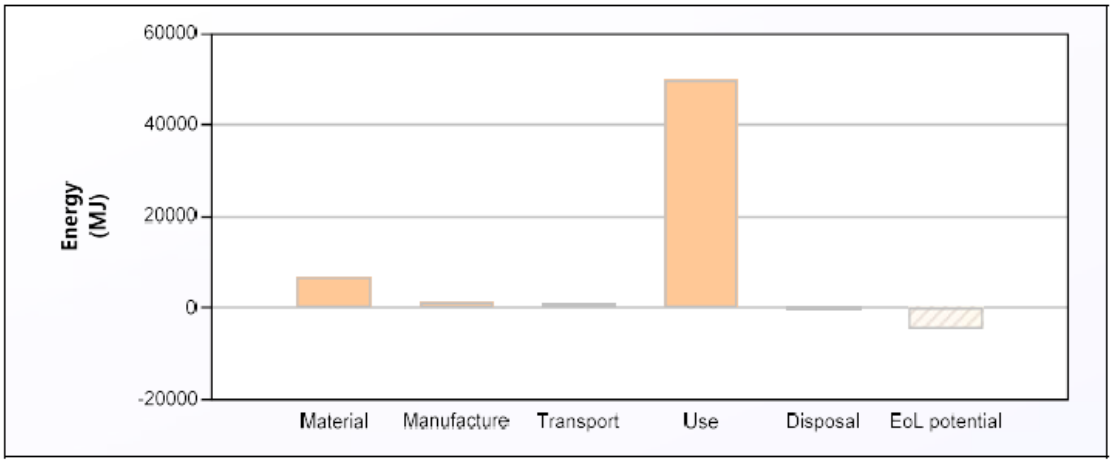
Phase	Energy (MJ)	Energy (%)	CO2 footprint (kg)	CO2 footprint (%)
Material	6,67e+03	11,4	503	17,0
Manufacture	1,29e+03	2,2	96,7	3,3
Transport	766	1,3	55,1	1,9
Use	4,96e+04	84,9	2,29e+03	77,6
Disposal	91	0,2	6,37	0,2
Total (for first life)	**5,84e+04**	**100**	**2,95e+03**	**100**
End of life potential	-4,6e+03		-341	

GRANTA EduPack

Eco Audit Report

Energy Analysis

Summary



	Energy (MJ/year)
Equivalent annual environmental burden (averaged over 10 year product life):	5,84e+03

Detailed breakdown of individual life phases

Material:

Summary

Component	Material	Recycled content* (%)	Part mass (kg)	Qty.	Total mass (kg)	Energy (MJ)	%
Acero Inoxidable	Acero inoxidable	Typical %	1,3e+02	1	1,3e+02	6,7e+03	100,0
Total				1	1,3e+02	6,7e+03	100

*Typical: Includes 'recycle fraction in current supply'

Manufacture:

Summary

Component	Process	Amount processed	Energy (MJ)	%
Acero Inoxidable	Extrusion, foil rolling	1,3e+02 kg	1,3e+03	100,0
Total			1,3e+03	100

Transport:

Summary

Breakdown by transport stage

Stage name	Transport type	Distance (km)	Energy (MJ)	%
1	40 tonne (6 axle) truck	7e+03	7,5e+02	97,5
2	14 tonne (2 axle) truck	1e+02	20	2,5
Total		7,1e+03	7,7e+02	100

Breakdown by components

Component	Mass (kg)	Energy (MJ)	%
Acero Inoxidable	1,3e+02	7,7e+02	100,0
Total	1,3e+02	7,7e+02	100

Use:

Summary

Static mode

Energy input and output type	Electric to mechanical (electric motors)
Country of use	Spain
Power rating (W)	2,2e+02
Usage (hours per day)	10
Usage (days per year)	3,3e+02
Product life (years)	10

Relative contribution of static and mobile modes

Mode	Energy (MJ)	%
Static	5e+04	100,0
Mobile	0	
Total	5e+04	100

Disposal:

Summary

Component	End of life option	Energy (MJ)	%
Acero Inoxidable	Recycle	91	100,0
Total		91	100

EoL potential:

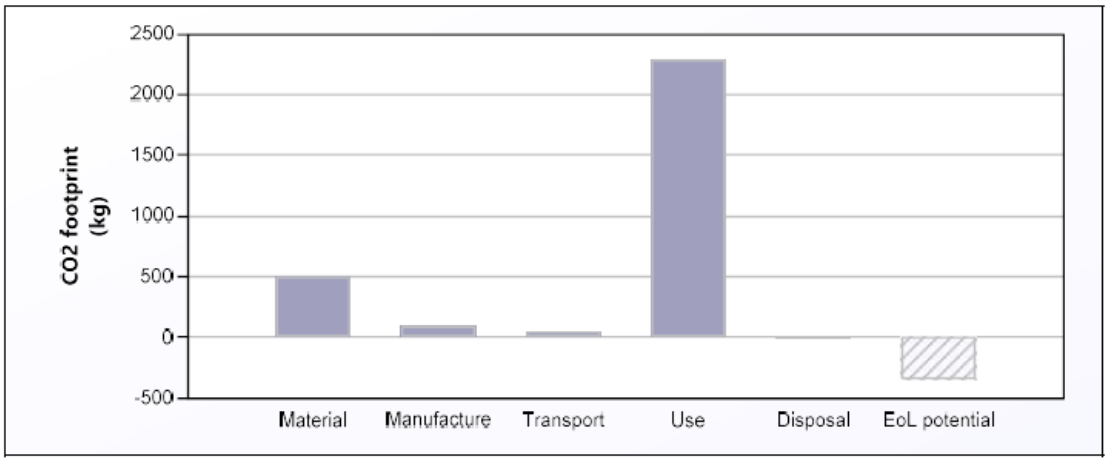
Component	End of life option	Energy (MJ)	%
Acero Inoxidable	Recycle	-4,6e+03	100,0
Total		-4,6e+03	100

GRANTA EduPack

Eco Audit Report

CO2 Footprint Analysis

Summary



	CO2 (kg/year)
Equivalent annual environmental burden (averaged over 10 year product life):	295

Detailed breakdown of individual life phases

Material:

Summary

Component	Material	Recycled content* (%)	Part mass (kg)	Qty.	Total mass (kg)	CO2 footprint (kg)	%
Acero Inoxidable	Acero inoxidable	Typical %	1,3e+02	1	1,3e+02	5e+02	100,0
Total				1	1,3e+02	5e+02	100

*Typical: Includes 'recycle fraction in current supply'

Manufacture:

Summary

Component	Process	Amount processed	CO2 footprint (kg)	%
Acero Inoxidable	Extrusion, foil rolling	1,3e+02 kg	97	100,0
Total			97	100

Transport:

Summary

Breakdown by transport stage

Stage name	Transport type	Distance (km)	CO2 footprint (kg)	%
1	40 tonne (6 axle) truck	7e+03	54	97,5
2	14 tonne (2 axle) truck	1e+02	1,4	2,5
Total		7,1e+03	55	100

Breakdown by components

Component	Mass (kg)	CO2 footprint (kg)	%
Acero Inoxidable	1,3e+02	55	100,0
Total	1,3e+02	55	100

Use:

Summary

Static mode

Energy input and output type	Electric to mechanical (electric motors)
Country of use	Spain
Power rating (W)	2,2e+02
Usage (hours per day)	10
Usage (days per year)	3,3e+02
Product life (years)	10

Relative contribution of static and mobile modes

Mode	CO2 footprint (kg)	%
Static	2,3e+03	100,0
Mobile	0	
Total	2,3e+03	100

Disposal:

Summary

Component	End of life option	CO2 footprint (kg)	%
Acero Inoxidable	Recycle	6,4	100,0
Total		6,4	100

EoL potential:

Component	End of life option	CO2 footprint (kg)	%
Acero Inoxidable	Recycle	-3,4e+02	100,0
Total		-3,4e+02	100

13.2. CODI COMPLET APLICAT AL PROGRAMA VISUAL STUDIO

```
using System;
using System.IO;
using SolidEdgeFramework;
using SolidEdgeFrameworkSupport;
using SolidEdgePart;
using SolidEdgeCommunity;
using SolidEdgeCommunity.Extensions;
using SolidEdgeAssembly;
using SolidEdgeDraft;

namespace TreballDeFiDeGrau
{
    class ModificacioCampana
    {
        [STAThread]
        static void Main(string[] args)
        {
            // Solicitar la entrada del usuario antes de cualquier operación
            // Se solicita que las cotas sean en metros y con coma, ya que en el momento de poner la coma y que solidedge la interprete como punto
            Console.Write("Ponga las siguientes dimensiones en metros, con el separador decimal con una coma");

            // Se solicita la medida longitudinal de la campana extractora, recordando que tiene que tener un valor entre las siguiente dimensiones
            Console.Write("Por favor, introduce el valor de la longitud de la campana extractora (Debe de ser hasta 2900 mm): ");

            // Se crea la variable longitud y filtro, para guardar la longitud y el tipo de filtro que se quiera, si de 390 o 490 mm

            double longitud;
            double filtro;
            while (!double.TryParse(Console.ReadLine(), out longitud))
            {
```

```
    Console.WriteLine("Valor no válido. Por favor, introduce un número.");
    Console.Write("Por favor, introduce el valor de la dimensión: ");
}
double fondo;

Console.Write("Por favor, introduce el valor del fondo de la campana extractora (Debe de ser mayor a 1000 m y hasta 3000 mm) : ");
while (!double.TryParse(Console.ReadLine(), out fondo))
{
    Console.WriteLine("Valor no válido. Por favor, introduce un número.");
    Console.Write("Por favor, introduce el valor de la dimensión: ");
}
double altura;
Console.Write("Por favor, introduce el valor de la altura de la campana extractora (Debe de ser entre 350 mm y 650 mm): ");
while (!double.TryParse(Console.ReadLine(), out altura))
{
    Console.WriteLine("Valor no válido. Por favor, introduce un número.");
    Console.Write("Por favor, introduce el valor de la dimensión: ");
}
while (!double.TryParse(Console.ReadLine(), out filtro))
{
    Console.WriteLine("Valor no válido. Por favor, introduce un número.");
    Console.Write("Por favor, introduce el valor de la dimensión: ");
}

// Se dan las diferentes medidas que ha introducido el usuario, para verificar si son las que necesita
Console.WriteLine($"El valor de la longitud de la campana será de {longitud*1000} mm");
Console.WriteLine($"El valor del fondo de la campana será de {fondo*1000} mm");
Console.WriteLine($"El valor de la altua de la campana será de {altura*1000} mm");
Console.WriteLine($"El valor del ancho del filtro es: {filtro}");

filtro = filtro / 1000;
```

```
// Rutas del archivo del SolidEdge, donde se indican que directorios y con que nombres contienen las piezas necesarias para el programa
string Bandeja = @"C:\Users\sebib\OneDrive\Escritorio\SOLDADA RECTANGULAR\85SOLDBANDEJA.psm";
string Frontis = @"C:\Users\sebib\OneDrive\Escritorio\SOLDADA RECTANGULAR\85SOLDFRONTIS.psm";
string Visera = @"C:\Users\sebib\OneDrive\Escritorio\SOLDADA RECTANGULAR\85SOLDVISERA.psm";
string Trasera = @"C:\Users\sebib\OneDrive\Escritorio\SOLDADA RECTANGULAR\85SOLDTECHOESPALDA.psm";
string Vigueta = @"C:\Users\sebib\OneDrive\Escritorio\SOLDADA RECTANGULAR\85SOLDVIGUETA.psm";
string ChapaPlein = @"C:\Users\sebib\OneDrive\Escritorio\SOLDADA RECTANGULAR\85SOLDCHAPAPLEIN.psm";
string DeflectorAirdoor = @"C:\Users\sebib\OneDrive\Escritorio\SOLDADA RECTANGULAR\85SOLDDEFLAIRDOOR.psm";
string Lateral = @"C:\Users\sebib\OneDrive\Escritorio\SOLDADA RECTANGULAR\85SOLDLATERAL.psm";

// Ruta del archivo del SolidEdge, con el ensamble que se tiene que abrir y el nombre que tiene
string CampanaIndustrial = @"C:\Users\sebib\OneDrive\Escritorio\SOLDADA RECTANGULAR\85SOLD2100X1100_v3.asm";

// Llamar a la función que modifica la dimensión de cada una de las piezas
// En estas funciones se dan como inputs la pieza a modificar y los valores del usuario necesarios que interactúan con las variables
ModificacionBandeja(Bandeja, longitud, altura);
ModificacionFrontis(Frontis, longitud, altura);
ModificacionVisera(Visera, longitud, fondo, altura);
ModificacionChapaPlein(ChapaPlein, longitud, filtro, altura);
ModificacionDeflectorAirdoor(DeflectorAirdoor, longitud, altura);
ModificacionTrasera(Trasera, longitud, altura);
ModificacionLateral(Lateral, fondo, altura);
ModificacionVigueta(Vigueta, longitud, filtro);

// Ruta de salida del archivo DXF que de donde se quieren guardar los desarrollos una vez modificadas las piezas
string outputPath = @"C:\Users\sebib\OneDrive\Escritorio\Prueba\85SOLDBANDEJA.dxf";
string outputFrontis = @"C:\Users\sebib\OneDrive\Escritorio\Prueba\85SOLDFRONTIS.dxf";
string outputVisera = @"C:\Users\sebib\OneDrive\Escritorio\Prueba\85SOLDVISERA.dxf";
string outputTrasera = @"C:\Users\sebib\OneDrive\Escritorio\Prueba\85SOLDTRASERA.dxf";
string outputDeflectorAirdoor = @"C:\Users\sebib\OneDrive\Escritorio\Prueba\85SOLDDEFLAIRDOOR.dxf";
string outputChapaPlein = @"C:\Users\sebib\OneDrive\Escritorio\Prueba\85SOLDCHAPAPLEIN.dxf";
string outputLateral = @"C:\Users\sebib\OneDrive\Escritorio\Prueba\85SOLDLATERAL.dxf";
```

```
    string outputVigueta = @"C:\Users\sebib\OneDrive\Escritorio\Prueba\85SOLDVIGUETA.dxf";

    // Llamar a la función que guarda el desarrollo plano del programa en DXF, una vez modificados
    // En la función se llama cada pieza en concreto y las rutas anteriormente guardadas
    SaveFlatPatternAsDXF(Bandeja, outputPath);
    SaveFrontisPatternAsDXF(Frontis, outputFrontis);
    SaveViseraPatternAsDXF(Visera, outputVisera);
    SaveChapaPleinPatternAsDXF(ChapaPlein, outputChapaPlein);
    SaveDeflectorAirdoorPatternAsDXF(DeflectorAirdoor, outputDeflectorAirdoor);
    SaveTraseraPatternAsDXF(Trasera, outputTrasera);
    SaveLateralPatternAsDXF(Lateral, outputLateral);
    SaveVigueta PatternAsDXF(Vigueta, outputVigueta);

    // Seguidamente se abre el conjunto que el usuario ha pedido, con el cuál este verifica si es el ensamble que pide
    AbrirConjuntoCampana(CampanaIndustrial);

    // Después de verificar que el ensamble es el correcto, se dan los datos básicos que se piden en la oficina técnica para el presupusesto
    // Cada función es el cálculo del tiempo que se tardan en hacer las piezas, pero para ampliar el proyecto se dará un precio
    PrecioBandeja(Bandeja);
    PrecioFrontis(Frontis);
    PrecioVisera(Visera);
    PrecioTrasera(Trasera);
    PrecioLateral(Lateral);
    PrecioDeflectorAirdoor(DeflectorAirdoor);
    PrecioVigueta(Vigueta);
    PrecioChapaPlein(ChapaPlein);

}

// Programa que abre la bandeja y modifica las variables indicadas en el proceso
static void ModificacionBandeja(string Bandeja, double longitud, double altura)
{
```

```
try
{
    OleMessageFilter.Register();

    // Conexión a Solid Edge
    SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

    // Abrir el documento
    SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
    SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(Bandeja);

    // Obtener la colección real de variables
    SolidEdgeFramework.Variables variables = (Variables)seSheetMetal.Variables;

    // Obtener la lista de dimensiones
    SolidEdgeFramework.VariableList listOfDimensions = (VariableList)variables.Query(
        pFindCriterium: "*",
        NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
        VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeDimension);


    // Modificar la dimensión "lb" con el valor proporcionado por el usuario
    SolidEdgeFrameworkSupport.Dimension dimToModifyLb = (Dimension)listOfDimensions.Item("lb");
    Console.WriteLine($"Dimensión 'lb' antes de modificar: {dimToModifyLb.Value}");
    dimToModifyLb.Value = (longitud * 1000 - 2) / 1000;
    Console.WriteLine($"Dimensión 'lb' después de modificar: {dimToModifyLb.Value}");

    SolidEdgeFrameworkSupport.Dimension dimToModifyangulo = (Dimension)listOfDimensions.Item("angulo");
    Console.WriteLine($"Dimensión 'angulo' antes de modificar: {dimToModifyangulo.Value}");
    if (altura > 0.5)
    {
        altura = 0.5;
```

```
    }
    double anguloradianes = Math.Asin((altura * 1000 - 102.65) / 561.22);
    dimToModifyangulo.Value = 3.14159 - anguloradianes;
    Console.WriteLine($"Dimensión 'angulo' después de modificar: {dimToModifyangulo.Value}");

    System.Threading.Thread.Sleep(1000);

    //Obtener lista de variables
    SolidEdgeFramework.VariableList listofVariables = (VariableList)variables.Query(
        pFindCriterium: "*",
        NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
        VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeVariable);

    // Del lista de variables, se quiere modificar la variable llamada "pat"
    SolidEdgeFramework.variable dimToModifyPat = (variable)listofVariables.Item("pat");

    // Creación de una variable nueva que guarde el número que se desea poner
    int patValue = ((((int)((longitud * 1000 - 2) / 150) / 2) + 1));

    // Modificación de la variable "pat", la cuál será igual que la nueva variable creada
    dimToModifyPat.Value = patValue;


    Console.WriteLine($"Dimensión 'pat' después de modificar: {dimToModifyPat.Value}");

    // Guardar los cambios en el documento
    // Tiempos de espera que puedan garantizar el buen funcionamiento del programa
    seSheetMetal.Save();
    System.Threading.Thread.Sleep(1000);
    seSheetMetal.Close();
    System.Threading.Thread.Sleep(1000);
}
```

```
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
        throw;
    }
    finally
    {
        OleMessageFilter.Unregister();
    }
}


// Programa que abre el frontis y modifica las variables indicadas en el proceso
static void ModificacionFrontis(string Frontis, double longitud, double altura)
{
    try
    {
        OleMessageFilter.Register();

        // Conexión a Solid Edge
        SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

        // Abrir el documento
        SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
        SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(Frontis);

        // Obtener la colección real de variables
        SolidEdgeFramework.Variables variables = (Variables)seSheetMetal.Variables;

        // Obtener la lista de dimensiones
        SolidEdgeFramework.VariableList listOfDimensions = (VariableList)variables.Query(
            pFindCriterium: "*",
```

```
    NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
    VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeDimension);

SolidEdgeFrameworkSupport.Dimension dimToModifyhfrontis = (Dimension)listOfDimensions.Item("hfrontis");
Console.WriteLine($"Dimensión 'hfrontis' antes de modificar: {dimToModifyhfrontis.Value}");
double nuevaaltura;
if (altura > 0.5)
{
    nuevaaltura = ((altura * 1000 - 500) + 80) * 0.001;
}
else
{
    nuevaaltura = 80 * 0.001;
}
dimToModifyhfrontis.Value = nuevaaltura;
Console.WriteLine($"Dimensión 'hfrontis' después de modificar: {dimToModifyhfrontis.Value}");
System.Threading.Thread.Sleep(10000);
SolidEdgeFrameworkSupport.Dimension dimToModifyangulof = (Dimension)listOfDimensions.Item("angulo");
Console.WriteLine($"Dimensión 'angulo' antes de modificar: {dimToModifyangulof.Value}");

if (altura > 0.5)
{
    altura = 0.5;
}
double anguloradianes = Math.Asin((altura * 1000 - 102.65) / 561.22);
dimToModifyangulof.Value = 3.14159 / 2 + anguloradianes;
Console.WriteLine($"Dimensión 'angulo' después de modificar: {dimToModifyangulof.Value}");

System.Threading.Thread.Sleep(1000);

// Modificar la dimensión "lf" con el valor proporcionado por el usuario
SolidEdgeFrameworkSupport.Dimension dimToModifyLb = (Dimension)listOfDimensions.Item("lf");
```

```
    Console.WriteLine($"Dimensión 'lf' antes de modificar: {dimToModifyLb.Value}");
    dimToModifyLb.Value = (longitud * 1000 - 2) / 1000;
    Console.WriteLine($"Dimensión 'lf' después de modificar: {dimToModifyLb.Value}");

    //Obtener lista de variables
    SolidEdgeFramework.VariableList listofVariables = (VariableList)variables.Query(
        pFindCriterium: "*",
        NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
        VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeVariable);

    // Calcular el valor de "pat" y modificarlo
    //double patValue = ((((int)((longitud * 1000 - 2) / 150) / 2) + 1) / 1000.0);
    SolidEdgeFramework.variable dimToModifyPat = (variable)listofVariables.Item("pat");
    int patValue = ((((int)((longitud * 1000 - 2) / 150) / 2) + 1));
    dimToModifyPat.Value = patValue;
    Console.WriteLine($"Dimensión 'pat' después de modificar: {dimToModifyPat.Value}");

    // Guardar los cambios en el documento
    seSheetMetal.Save();
    System.Threading.Thread.Sleep(1000);
    seSheetMetal.Close();
    System.Threading.Thread.Sleep(1000);
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
    throw;
}
finally
{
    OleMessageFilter.Unregister();
}
```

```
}


// Programa que abre la visera y modifica las variables indicadas en el proceso
static void ModificacionVisera(string Visera, double longitud, double fondo, double altura)
{
    try
    {
        OleMessageFilter.Register();

        // Conexión a Solid Edge
        SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

        // Abrir el documento
        SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
        SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(Visera);

        // Obtener la colección real de variables
        SolidEdgeFramework.Variables variables = (Variables)seSheetMetal.Variables;

        // Obtener la lista de dimensiones
        SolidEdgeFramework.VariableList listOfDimensions = (VariableList)variables.Query(
            pFindCriterium: "*",
            NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
            VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeDimension);

        double alturabuena = altura;

        if (altura > 0.5)
        {
            altura = 0.5;
        }
```

```
// Modificar la dimensión "lconnect" con el valor proporcionado por el usuario
SolidEdgeFrameworkSupport.Dimension dimToModifylconnect = (Dimension)listOfDimensions.Item("lconnect");
Console.WriteLine($"Dimensión 'lconnect' antes de modificar: {dimToModifylconnect.Value}");
dimToModifylconnect.Value = (longitud * 1000 - 42 - 2) / 1000;
Console.WriteLine($"Dimensión 'lconnect' después de modificar: {dimToModifylconnect.Value}");

System.Threading.Thread.Sleep(1000);

// Modificar la dimensión "lb" con el valor proporcionado por el usuario
SolidEdgeFrameworkSupport.Dimension dimToModifyLb = (Dimension)listOfDimensions.Item("lv");
Console.WriteLine($"Dimensión 'lv' antes de modificar: {dimToModifyLb.Value}");
dimToModifyLb.Value = (longitud * 1000 - 2) / 1000;
Console.WriteLine($"Dimensión 'lv' después de modificar: {dimToModifyLb.Value}");

SolidEdgeFrameworkSupport.Dimension dimToModifyLv = (Dimension)listOfDimensions.Item("fv");
Console.WriteLine($"Dimensión 'fv' antes de modificar: {dimToModifyLb.Value}");
double anguloradianes = Math.Asin((altura * 1000 - 102.65) / 561.22);
double longcolect = Math.Cos(anguloradianes) * 560.94;
double longtrasera = (146.55 + longcolect + 1.3);
dimToModifyLv.Value = (fondo * 1000 - longtrasera - 1 - 1) / 1000;
Console.WriteLine($"Dimensión 'fv' después de modificar: {dimToModifyLb.Value}");

SolidEdgeFrameworkSupport.Dimension dimToModifyhv = (Dimension)listOfDimensions.Item("hv");
Console.WriteLine($"Dimensión 'hv' antes de modificar: {dimToModifyhv.Value}");
dimToModifyhv.Value = (alturabuena * 1000 - 2.3) / 1000;
Console.WriteLine($"Dimensión 'hv' después de modificar: {dimToModifyhv.Value}");

System.Threading.Thread.Sleep(1000);

//Obtener lista de variables
SolidEdgeFramework.VariableList listofVariables = (VariableList)variables.Query(
```

```
        pFindCriterium: "*",
        NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
        VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeVariable);

    // Calcular el valor de "pat" y modificarlo
    SolidEdgeFramework.variable dimToModifyPatv = (variable)listofVariables.Item("patv");
    int patValuev = ((((int)((alturabuena * 1000 - 2.3 - 30) / 100)) + 1));
    dimToModifyPatv.Value = patValuev;
    Console.WriteLine($"Dimensión 'patv' después de modificar: {dimToModifyPatv.Value}");

    // Calcular el valor de "pat" y modificarlo
    //double patValue = ((((int)((longitud * 1000 - 2) / 150) / 2) + 1) / 1000.0);
    SolidEdgeFramework.variable dimToModifyPat = (variable)listofVariables.Item("pat");
    int patValue = ((((int)((longitud * 1000 - 2) / 150) / 2) + 1));
    dimToModifyPat.Value = patValue;
    Console.WriteLine($"Dimensión 'pat' después de modificar: {dimToModifyPat.Value}");

    // Guardar los cambios en el documento
    seSheetMetal.Save();
    System.Threading.Thread.Sleep(1000);
    seSheetMetal.Close();
    System.Threading.Thread.Sleep(1000);
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
    throw;
}
finally
{
    OleMessageFilter.Unregister();
}
```

```
}


// Programa que abre la trasera y modifica las variables indicadas en el proceso
static void ModificacionTrasera(string Trasera, double longitud, double altura)
{
    try
    {
        OleMessageFilter.Register();

        // Conexión a Solid Edge
        SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

        // Abrir el documento
        SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
        SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(Trasera);

        // Obtener la colección real de variables
        SolidEdgeFramework.Variables variables = (Variables)seSheetMetal.Variables;

        // Obtener la lista de dimensiones
        SolidEdgeFramework.VariableList listOfDimensions = (VariableList)variables.Query(
            pFindCriterium: "*",
            NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
            VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeDimension);

        double alturabuena = altura;

        // Modificar la dimensión "lconnect" con el valor proporcionado por el usuario
        SolidEdgeFrameworkSupport.Dimension dimToModifylconnect = (Dimension)listOfDimensions.Item("lconnect");
        Console.WriteLine($"Dimensión 'lconnect' antes de modificar: {dimToModifylconnect.Value}");
        dimToModifylconnect.Value = (longitud * 1000 - 38) / 1000;
```

```
Console.WriteLine($"Dimensión 'lconnect' después de modificar: {dimToModifylconnect.Value}");

System.Threading.Thread.Sleep(1000);

// Modificar la dimensión "lb" con el valor proporcionado por el usuario
SolidEdgeFrameworkSupport.Dimension dimToModifyLb = (Dimension)listOfDimensions.Item("lt");
Console.WriteLine($"Dimensión 'lt' antes de modificar: {dimToModifyLb.Value}");
dimToModifyLb.Value = (longitud * 1000 - 4) / 1000;
Console.WriteLine($"Dimensión 'lt' después de modificar: {dimToModifyLb.Value}");

SolidEdgeFrameworkSupport.Dimension dimToModifyht = (Dimension)listOfDimensions.Item("ht");
Console.WriteLine($"Dimensión 'ht' antes de modificar: {dimToModifyht.Value}");
dimToModifyht.Value = (alturabuena * 1000 - 2.3 - 21.55) / 1000;
Console.WriteLine($"Dimensión 'ht' después de modificar: {dimToModifyht.Value}");

SolidEdgeFrameworkSupport.Dimension dimToModifyft = (Dimension)listOfDimensions.Item("ft");
Console.WriteLine($"Dimensión 'ft' antes de modificar: {dimToModifyft.Value}");
double anguloradianes = Math.Asin((alturabuena * 1000 - 102.65) / 561.22);
double longcolect = Math.Cos(anguloradianes) * 560.94;
if (alturabuena > 0.5)
{
   dimToModifyft.Value = 0.54399;
}
else
{
   dimToModifyft.Value = (146.55 + (longcolect) + 1.3) / 1000;
   Console.WriteLine($"Dimensión 'ft' después de modificar: {dimToModifyft.Value}");
}

System.Threading.Thread.Sleep(1000);

//Obtener lista de variables
```

```
    SolidEdgeFramework.VariableList listofVariables = (VariableList)variables.Query(
        pFindCriterium: "*",
        NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
        VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeVariable);

    // Calcular el valor de "pat" y modificarlo
    //double patValue = ((((int)((longitud * 1000 - 2) / 150) / 2) + 1) / 1000.0);
    SolidEdgeFramework.variable dimToModifyPat = (variable)listofVariables.Item("pat");
    int patValue = ((((int)((longitud * 1000 - 4) / 150) / 2) + 1));
    dimToModifyPat.Value = patValue;
    Console.WriteLine($"Dimensión 'pat' después de modificar: {dimToModifyPat.Value}");

    // Calcular el valor de "pat" y modificarlo
    SolidEdgeFramework.variable dimToModifyPatv = (variable)listofVariables.Item("patv");
    int patValuev = ((((int)((alturabuena * 1000 - 2.3 - 21.55) / 100)) + 1));
    dimToModifyPatv.Value = patValuev;
    Console.WriteLine($"Dimensión 'patv' después de modificar: {dimToModifyPatv.Value}");

    // Calcular el valor de "pat" y modificarlo
    SolidEdgeFramework.variable dimToModifyPatf = (variable)listofVariables.Item("patf");
    int patValuef = ((((int)(((146.55 + (longcolect) + 1.3) / 1000 - 100) / 100)) + 1));
    dimToModifyPatv.Value = patValuef;
    Console.WriteLine($"Dimensión 'patf' después de modificar: {dimToModifyPatf.Value}");

    // Guardar los cambios en el documento
    seSheetMetal.Save();
    System.Threading.Thread.Sleep(1000);
    seSheetMetal.Close();
    System.Threading.Thread.Sleep(1000);
}
catch (Exception e)
{
```

```
        Console.WriteLine(e.Message);
        throw;
    }
    finally
    {
        OleMessageFilter.Unregister();
    }
}


// Programa que abre el deflector airdoor y modifica las variables indicadas en el proceso
static void ModificacionDeflectorAirdoor(string DeflectorAirdoor, double longitud, double altura)
{
    try
    {
        OleMessageFilter.Register();

        // Conexión a Solid Edge
        SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

        // Abrir el documento
        SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
        SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(DeflectorAirdoor);

        // Obtener la colección real de variables
        SolidEdgeFramework.Variables variables = (Variables)seSheetMetal.Variables;

        // Obtener la lista de dimensiones
        SolidEdgeFramework.VariableList listOfDimensions = (VariableList)variables.Query(
            pFindCriterium: "*",
            NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
            VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeDimension);
```

```
// Modificar la dimensión "lb" con el valor proporcionado por el usuario
SolidEdgeFrameworkSupport.Dimension dimToModifyLb = (Dimension)listOfDimensions.Item("ld");
Console.WriteLine($"Dimensión 'ld' antes de modificar: {dimToModifyLb.Value}");
dimToModifyLb.Value = (longitud * 1000 - 4) / 1000;
Console.WriteLine($"Dimensión 'ld' después de modificar: {dimToModifyLb.Value}");

System.Threading.Thread.Sleep(1000);

SolidEdgeFrameworkSupport.Dimension dimToModifyhdefl = (Dimension)listOfDimensions.Item("hdefl");
Console.WriteLine($"Dimensión 'hdefl' antes de modificar: {dimToModifyLb.Value}");
dimToModifyhdefl.Value = (altura * 1000 - 21.8) / 1000;
Console.WriteLine($"Dimensión 'hdefl' después de modificar: {dimToModifyhdefl.Value}");

//Obtener lista de variables
SolidEdgeFramework.VariableList listofVariables = (VariableList)variables.Query(
    pFindCriterium: "*",
    NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
    VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeVariable);

// Calcular el valor de "pat" y modificarlo
//double patValue = ((((int)((longitud * 1000 - 2) / 150) / 2) + 1) / 1000.0);
SolidEdgeFramework.variable dimToModifyPat = (variable)listofVariables.Item("patag");
int patValue = ((((int)((longitud * 1000 - 2) / 33) / 2) + 1));
dimToModifyPat.Value = patValue;
Console.WriteLine($"Dimensión 'patag' después de modificar: {dimToModifyPat.Value}");

// Guardar los cambios en el documento
seSheetMetal.Save();
System.Threading.Thread.Sleep(1000);
seSheetMetal.Close();
System.Threading.Thread.Sleep(1000);
```

```
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
        throw;
    }
    finally
    {
        OleMessageFilter.Unregister();
    }
}


// Programa que abre la chapa plein y modifica las variables indicadas en el proceso
static void ModificacionChapaPlein(string ChapaPlein, double longitud, double filtro, double altura)
{
    try
    {
        OleMessageFilter.Register();

        // Conexión a Solid Edge
        SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

        // Abrir el documento
        SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
        SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(ChapaPlein);

        // Obtener la colección real de variables
        SolidEdgeFramework.Variables variables = (Variables)seSheetMetal.Variables;

        // Obtener la lista de dimensiones
        SolidEdgeFramework.VariableList listOfDimensions = (VariableList)variables.Query(
```

```
    pFindCriterium: "*",
    NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
    VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeDimension);

double alturabuena = altura;

SolidEdgeFrameworkSupport.Dimension dimToModifyDistfiltro = (Dimension)listOfDimensions.Item("V28643");
int patfiltro = ((((int)((longitud * 1000 - 3) / (filtro * 1000)))));
dimToModifyDistfiltro.Value = (((longitud * 1000 - 3) / 1000) - 0.2 * patfiltro) / (patfiltro + 1);

// Modificar la dimensión "lb" con el valor proporcionado por el usuario
SolidEdgeFrameworkSupport.Dimension dimToModifyLb = (Dimension)listOfDimensions.Item("lc");
Console.WriteLine($"Dimensión 'lc' antes de modificar: {dimToModifyLb.Value}");
dimToModifyLb.Value = (longitud * 1000 - 3) / 1000;
Console.WriteLine($"Dimensión 'lc' después de modificar: {dimToModifyLb.Value}");

SolidEdgeFrameworkSupport.Dimension dimToModifyalfa = (Dimension)listOfDimensions.Item("alfa");
double anguloradianes = Math.Asin((alturabuena * 1000 - 102.65) / 561.22);
Console.WriteLine($"Dimensión 'anguloradianes' después de modificar: {anguloradianes}");
double longcolect = Math.Cos(anguloradianes) * 560.94;
Console.WriteLine($"Dimensión 'longcolect' después de modificar: {longcolect}");
double ft;
if (alturabuena > 0.5)
{
    ft = 0.54399;
}
else
{
    ft = (146.55 + (longcolect) + 1.3) / 1000;
}
Console.WriteLine($"Dimensión 'ft' después de modificar: {ft}");
double fp = (ft * 1000 - 21.76) / 1000;
```

```
Console.WriteLine($"Dimensión 'fp' después de modificar: {fp}");
double hp = (altura * 1000 - 44.63) / 1000;
Console.WriteLine($"Dimensión 'hp' después de modificar: {hp}");
System.Threading.Thread.Sleep(1000);
dimToModifyalfa.Value = 3.14159 - Math.Atan(hp / fp);
Console.WriteLine($"EL VALOR QUE SE INTRODUCIRÁ DE ALFA ES... {dimToModifyalfa.Value}");

SolidEdgeFrameworkSupport.Dimension dimToModifydp = (Dimension)listOfDimensions.Item("dc");
dimToModifydp.Value = (hp / (Math.Sin(Math.Atan(hp / fp))));


System.Threading.Thread.Sleep(1000);

//Obtener lista de variables
SolidEdgeFramework.VariableList listofVariables = (VariableList)variables.Query(
    pFindCriterium: "*",
    NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
    VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeVariable);


SolidEdgeFramework.variable dimToModifydistcuad = (variable)listofVariables.Item("distpatcuad");
dimToModifydistcuad.Value = ((((longitud * 1000 - 3) / 1000) - 0.2 * patfiltro) / (patfiltro + 1)) + 0.2;

// Calcular el valor de "pat" y modificarlo
//double patValue = ((((int)((longitud * 1000 - 2) / 150) / 2) + 1) / 1000.0);
SolidEdgeFramework.variable dimToModifyPat = (variable)listofVariables.Item("pat");
int patagValue = ((((int)((longitud * 1000 - 3) / 150) / 2) + 1));
dimToModifyPat.Value = patagValue;
Console.WriteLine($"Dimensión 'pat' después de modificar: {dimToModifyPat.Value}");


//double patValue = ((((int)((longitud * 1000 - 2) / 150) / 2) + 1) / 1000.0);
```

```
        SolidEdgeFramework.variable dimToModifyPat3 = (variable)listofVariables.Item("patag3");
        int patag3 = ((((int)((longitud * 1000 - 3) / 150) / 2) + 1));
        dimToModifyPat3.Value = patag3;
        Console.WriteLine($"Dimensión 'patag3' después de modificar: {dimToModifyPat.Value}");

        // Calcular el valor de "pat" y modificarlo
        //double patValue = ((((int)((longitud * 1000 - 2) / 150) / 2) + 1) / 1000.0);
        SolidEdgeFramework.variable dimToModifyPatfiltro = (variable)listofVariables.Item("pat_cuad");
        dimToModifyPatfiltro.Value = patfiltro;


        // Guardar los cambios en el documento
        seSheetMetal.Save();
        System.Threading.Thread.Sleep(1000);
        seSheetMetal.Close();
        System.Threading.Thread.Sleep(1000);
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
        throw;
    }
    finally
    {
        OleMessageFilter.Unregister();
    }
}


// Programa que abre el lateral y modifica las variables indicadas en el proceso
static void ModificacionLateral(string Lateral, double fondo, double altura)
{
```

```
try
{
    OleMessageFilter.Register();

    // Conexión a Solid Edge
    SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

    // Abrir el documento
    SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
    SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(Lateral);

    // Obtener la colección real de variables
    SolidEdgeFramework.Variables variables = (Variables)seSheetMetal.Variables;

    // Obtener la lista de dimensiones
    SolidEdgeFramework.VariableList listOfDimensions = (VariableList)variables.Query(
        pFindCriterium: "*",
        NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
        VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeDimension);


    // Modificar la dimensión "lb" con el valor proporcionado por el usuario
    SolidEdgeFrameworkSupport.Dimension dimToModifyLb = (Dimension)listOfDimensions.Item("fl");
    Console.WriteLine($"Dimensión 'fl' antes de modificar: {dimToModifyLb.Value}");
    dimToModifyLb.Value = (fondo * 1000) / 1000;
    Console.WriteLine($"Dimensión 'fl' después de modificar: {dimToModifyLb.Value}");

    SolidEdgeFrameworkSupport.Dimension dimToModifypestsup = (Dimension)listOfDimensions.Item("pestsup");
    Console.WriteLine($"Dimensión 'pestsup' antes de modificar: {dimToModifyLb.Value}");
    dimToModifypestsup.Value = (fondo * 1000 - 3.3) / 1000;
    Console.WriteLine($"Dimensión 'pestsup' después de modificar: {dimToModifyLb.Value}");
```

```
    SolidEdgeFrameworkSupport.Dimension dimToModifyhl = (Dimension)listOfDimensions.Item("hl");
    Console.WriteLine($"Dimensión 'hl' antes de modificar: {dimToModifyhl.Value}");
    dimToModifyhl.Value = (altura * 1000 - 1.3) / 1000;
    Console.WriteLine($"Dimensión 'hl' después de modificar: {dimToModifyhl.Value}");


    //Obtener lista de variables
    SolidEdgeFramework.VariableList listofVariables = (VariableList)variables.Query(
        pFindCriterium: "*",
        NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
        VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeVariable);

    // Calcular el valor de "pat" y modificarlo
    SolidEdgeFramework.variable dimToModifyPatv = (variable)listofVariables.Item("patv");
    int patValuev = ((((int)((altura * 1000 - 2.3) / 100))));
    dimToModifyPatv.Value = patValuev;
    Console.WriteLine($"Dimensión 'patv' después de modificar: {dimToModifyPatv.Value}");

    // Guardar los cambios en el documento
    seSheetMetal.Save();
    System.Threading.Thread.Sleep(1000);
    seSheetMetal.Close();
    System.Threading.Thread.Sleep(1000);
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
    throw;
}
finally
{
    OleMessageFilter.Unregister();
```

```
    }
}


// Programa que abre la vigueta y modifica las variables indicadas en el proceso
static void ModificacionVigueta(string Vigueta, double longitud, double filtro)
{
    try
    {
        OleMessageFilter.Register();

        // Conexión a Solid Edge
        SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

        // Abrir el documento
        SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
        SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(Vigueta);

        // Obtener la colección real de variables
        SolidEdgeFramework.Variables variables = (Variables)seSheetMetal.Variables;

        // Obtener la lista de dimensiones
        SolidEdgeFramework.VariableList listOfDimensions = (VariableList)variables.Query(
            pFindCriterium: "*",
            NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
            VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeDimension);

        // Modificar la dimensión "lb" con el valor proporcionado por el usuario
        SolidEdgeFrameworkSupport.Dimension dimtoaf = (Dimension)listOfDimensions.Item("af");
        Console.WriteLine($"Dimensión 'af' antes de modificar: {dimtoaf.Value}");
        int qtfiltro = ((((int)((longitud * 1000 - 3) / (filtro * 1000)))));
        double af = ((longitud * 1000 - qtfiltro * (filtro * 1000 + 5)) / 2) / 1000;
```

```
        dimtoaf.Value = af;
        Console.WriteLine($"Dimensión 'af' después de modificar: {dimtoaf.Value}");

        System.Threading.Thread.Sleep(1000);

        // Guardar los cambios en el documento
        seSheetMetal.Save();
        System.Threading.Thread.Sleep(1000);
        seSheetMetal.Close();
        System.Threading.Thread.Sleep(1000);
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
        throw;
    }
    finally
    {
        OleMessageFilter.Unregister();
    }
}


// Programa que abre la bandeja y guarda el desarrollo en el directorio indicado
static void SaveFlatPatternAsDXF(string Bandeja, string outputPath)
{
    try
    {
        OleMessageFilter.Register();

        // Conexión a Solid Edge
        SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);
```

```
// Abrir el documento
SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(Bandeja);

// Guardar el patrón plano como DXF
SolidEdgePart.FlatPatternModels flatPatternModels = seSheetMetal.FlatPatternModels;
bool UseFlatPatternModel = true;
bool flag = false;

if (flatPatternModels.Count > 0)
{
    for (int Index = 1; Index <= flatPatternModels.Count; ++Index)
    {
        flag = flatPatternModels.Item((object)Index).IsUpToDate;
        if (flag)
            break;
    }
    if (!flag)
        UseFlatPatternModel = false;
}
else
{
    UseFlatPatternModel = false;
}

// Guardar como DXF
seSheetMetal.Models.SaveAsFlatDXFEx(outputPath, null, null, null, UseFlatPatternModel);
Console.WriteLine("El archivo DXF ha sido guardado en: " + outputPath);

seSheetMetal.Save();
System.Threading.Thread.Sleep(1000);
```

```
        seSheetMetal.Close();
        System.Threading.Thread.Sleep(1000);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error al guardar el archivo DXF: " + ex.Message);
    }
    finally
    {
        OleMessageFilter.Unregister();
    }
    System.Threading.Thread.Sleep(1000);
}

// Programa que abre el frontis y guarda el desarrollo en el directorio indicado
static void SaveFrontisPatternAsDXF(string Frontis, string outputFrontis)
{
    try
    {
        OleMessageFilter.Register();

        // Conexión a Solid Edge
        SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

        // Abrir el documento
        SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
        SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(Frontis);

        // Guardar el patrón plano como DXF
        SolidEdgePart.FlatPatternModels flatPatternModels = seSheetMetal.FlatPatternModels;
        bool UseFlatPatternModel = true;
        bool flag = false;
```

```
    if (flatPatternModels.Count > 0)
    {
        for (int Index = 1; Index <= flatPatternModels.Count; ++Index)
        {
            flag = flatPatternModels.Item((object)Index).IsUpToDate;
            if (flag)
                break;
        }
        if (!flag)
            UseFlatPatternModel = false;
    }
    else
    {
        UseFlatPatternModel = false;
    }

    // Guardar como DXF
    seSheetMetal.Models.SaveAsFlatDXFEx(outputFrontis, null, null, null, UseFlatPatternModel);
    Console.WriteLine("El archivo DXF ha sido guardado en: " + outputFrontis);

    seSheetMetal.Save();
    System.Threading.Thread.Sleep(1000);
    seSheetMetal.Close();
    System.Threading.Thread.Sleep(1000);
}
catch (Exception ex)
{
    Console.WriteLine("Error al guardar el archivo DXF: " + ex.Message);
}
finally
{
```

```
        OleMessageFilter.Unregister();
    }
    System.Threading.Thread.Sleep(1000);
}

// Programa que abre la visera y guarda el desarrollo en el directorio indicado
static void SaveViseraPatternAsDXF(string Visera, string outputVisera)
{
    try
    {
        OleMessageFilter.Register();

        // Conexión a Solid Edge
        SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

        // Abrir el documento
        SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
        SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(Visera);

        // Guardar el patrón plano como DXF
        SolidEdgePart.FlatPatternModels flatPatternModels = seSheetMetal.FlatPatternModels;
        bool UseFlatPatternModel = true;
        bool flag = false;

        if (flatPatternModels.Count > 0)
        {
            for (int Index = 1; Index <= flatPatternModels.Count; ++Index)
            {
                flag = flatPatternModels.Item((object)Index).IsUpToDate;
                if (flag)
                    break;
            }
```

```
            if (!flag)
                UseFlatPatternModel = false;
        }
        else
        {
            UseFlatPatternModel = false;
        }

        // Guardar como DXF
        seSheetMetal.Models.SaveAsFlatDXFEx(outputVisera, null, null, null, UseFlatPatternModel);
        Console.WriteLine("El archivo DXF ha sido guardado en: " + outputVisera);

        seSheetMetal.Save();
        System.Threading.Thread.Sleep(1000);
        seSheetMetal.Close();
        System.Threading.Thread.Sleep(1000);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error al guardar el archivo DXF: " + ex.Message);
    }
    finally
    {
        OleMessageFilter.Unregister();
    }
    System.Threading.Thread.Sleep(1000);
}

// Programa que abre la visera y guarda el desarrollo en el directorio indicado
static void SaveTraseraPatternAsDXF(string Trasera, string outputTrasera)
{
    try
```

```
{
    OleMessageFilter.Register();

    // Conexión a Solid Edge
    SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

    // Abrir el documento
    SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
    SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(Trasera);

    // Guardar el patrón plano como DXF
    SolidEdgePart.FlatPatternModels flatPatternModels = seSheetMetal.FlatPatternModels;
    bool UseFlatPatternModel = true;
    bool flag = false;

    if (flatPatternModels.Count > 0)
    {
        for (int Index = 1; Index <= flatPatternModels.Count; ++Index)
        {
            flag = flatPatternModels.Item((object)Index).IsUpToDate;
            if (flag)
                break;
        }
        if (!flag)
            UseFlatPatternModel = false;
    }
    else
    {
        UseFlatPatternModel = false;
    }

    // Guardar como DXF
```

```
        seSheetMetal.Models.SaveAsFlatDXFEx(outputTrasera, null, null, null, UseFlatPatternModel);
        Console.WriteLine("El archivo DXF ha sido guardado en: " + outputTrasera);

        seSheetMetal.Save();
        System.Threading.Thread.Sleep(1000);
        seSheetMetal.Close();
        System.Threading.Thread.Sleep(1000);

    }
    catch (Exception ex)
    {
        Console.WriteLine("Error al guardar el archivo DXF: " + ex.Message);
    }
    finally
    {
        OleMessageFilter.Unregister();
    }
    System.Threading.Thread.Sleep(1000);
}

// Programa que abre el deflector airdoor y guarda el desarrollo en el directorio indicado
static void SaveDeflectorAirdoorPatternAsDXF(string DeflectorAirdoor, string outputDeflectorAirdoor)
{
    try
    {
        OleMessageFilter.Register();

        // Conexión a Solid Edge
        SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

        // Abrir el documento
        SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
```

```
    SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(DeflectorAirdoor);

    // Guardar el patrón plano como DXF
    SolidEdgePart.FlatPatternModels flatPatternModels = seSheetMetal.FlatPatternModels;
    bool UseFlatPatternModel = true;
    bool flag = false;

    if (flatPatternModels.Count > 0)
    {
        for (int Index = 1; Index <= flatPatternModels.Count; ++Index)
        {
            flag = flatPatternModels.Item((object)Index).IsUpToDate;
            if (flag)
                break;
        }
        if (!flag)
            UseFlatPatternModel = false;
    }
    else
    {
        UseFlatPatternModel = false;
    }
    // Guardar como DXF
    seSheetMetal.Models.SaveAsFlatDXFEx(outputDeflectorAirdoor, null, null, null, UseFlatPatternModel);
    Console.WriteLine("El archivo DXF ha sido guardado en: " + outputDeflectorAirdoor);

    seSheetMetal.Save();
    System.Threading.Thread.Sleep(1000);
    seSheetMetal.Close();
    System.Threading.Thread.Sleep(1000);
}
catch (Exception ex)
```

```
    {
        Console.WriteLine("Error al guardar el archivo DXF: " + ex.Message);
    }
    finally
    {
        OleMessageFilter.Unregister();
    }
    System.Threading.Thread.Sleep(1000);
}

static void SaveChapaPleinPatternAsDXF(string ChapaPlein, string outputChapaPlein)
{
    try
    {
        OleMessageFilter.Register();

        // Conexión a Solid Edge
        SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

        // Abrir el documento
        SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
        SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(ChapaPlein);

        // Guardar el patrón plano como DXF
        SolidEdgePart.FlatPatternModels flatPatternModels = seSheetMetal.FlatPatternModels;
        bool UseFlatPatternModel = true;
        bool flag = false;

        if (flatPatternModels.Count > 0)
        {
            for (int Index = 1; Index <= flatPatternModels.Count; ++Index)
            {
```

```
            flag = flatPatternModels.Item((object)Index).IsUpToDate;
            if (flag)
                break;
        }
        if (!flag)
            UseFlatPatternModel = false;
    }
    else
    {
        UseFlatPatternModel = false;
    }

    // Guardar como DXF
    seSheetMetal.Models.SaveAsFlatDXFEx(outputChapaPlein, null, null, null, UseFlatPatternModel);
    Console.WriteLine("El archivo DXF ha sido guardado en: " + outputChapaPlein);

    seSheetMetal.Save();
    System.Threading.Thread.Sleep(1000);
    seSheetMetal.Close();
    System.Threading.Thread.Sleep(1000);
  }
  catch (Exception ex)
  {
    Console.WriteLine("Error al guardar el archivo DXF: " + ex.Message);
  }
  finally
  {
    OleMessageFilter.Unregister();
  }
  System.Threading.Thread.Sleep(1000);
}
```

```
static void SaveLateralPatternAsDXF(string Lateral, string outputLateral)
{
    try
    {
        OleMessageFilter.Register();

        // Conexión a Solid Edge
        SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

        // Abrir el documento
        SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
        SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(Lateral);

        // Guardar el patrón plano como DXF
        SolidEdgePart.FlatPatternModels flatPatternModels = seSheetMetal.FlatPatternModels;
        bool UseFlatPatternModel = true;
        bool flag = false;

        if (flatPatternModels.Count > 0)
        {
            for (int Index = 1; Index <= flatPatternModels.Count; ++Index)
            {
                flag = flatPatternModels.Item((object)Index).IsUpToDate;
                if (flag)
                    break;
            }
            if (!flag)
                UseFlatPatternModel = false;
        }
        else
        {
            UseFlatPatternModel = false;
```

```
        }

        // Guardar como DXF
        seSheetMetal.Models.SaveAsFlatDXFEx(outputLateral, null, null, null, UseFlatPatternModel);
        Console.WriteLine("El archivo DXF ha sido guardado en: " + outputLateral);

        seSheetMetal.Save();
        System.Threading.Thread.Sleep(1000);
        seSheetMetal.Close();
        System.Threading.Thread.Sleep(1000);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error al guardar el archivo DXF: " + ex.Message);
    }
    finally
    {
        OleMessageFilter.Unregister();
    }
    System.Threading.Thread.Sleep(1000);
}

static void SaveViguetaPatternAsDXF(string Vigueta, string outputVigueta)
{
    try
    {
        OleMessageFilter.Register();

        // Conexión a Solid Edge
        SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

        // Abrir el documento
```

```
SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(Vigueta);

// Guardar el patrón plano como DXF
SolidEdgePart.FlatPatternModels flatPatternModels = seSheetMetal.FlatPatternModels;
bool UseFlatPatternModel = true;
bool flag = false;

if (flatPatternModels.Count > 0)
{
    for (int Index = 1; Index <= flatPatternModels.Count; ++Index)
    {
        flag = flatPatternModels.Item((object)Index).IsUpToDate;
        if (flag)
            break;
    }
    if (!flag)
        UseFlatPatternModel = false;
}
else
{
    UseFlatPatternModel = false;
}

// Guardar como DXF
seSheetMetal.Models.SaveAsFlatDXFEx(outputVigueta, null, null, null, UseFlatPatternModel);
Console.WriteLine("El archivo DXF ha sido guardado en: " + outputVigueta);

seSheetMetal.Save();
System.Threading.Thread.Sleep(1000);
seSheetMetal.Close();
System.Threading.Thread.Sleep(1000);
```

```
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error al guardar el archivo DXF: " + ex.Message);
    }
    finally
    {
        OleMessageFilter.Unregister();
    }
    System.Threading.Thread.Sleep(1000);
}

static void AbrirConjuntoCampana(string CampanaIndustrial)
{
    try
    {
        OleMessageFilter.Register();

        // Conexión a Solid Edge
        SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

        // Abrir el documento
        SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
        SolidEdgeAssembly.AssemblyDocument seAssembly = (AssemblyDocument)seDocuments.Open(CampanaIndustrial);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error al guardar el archivo DXF: " + ex.Message);
    }
    finally
    {
        OleMessageFilter.Unregister();
```

```
    }
    System.Threading.Thread.Sleep(1000);
}

static void PrecioBandeja(string Bandeja)
{
    try
    {
        OleMessageFilter.Register();

        // Conexión a Solid Edge
        SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

        // Abrir el documento
        SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
        SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(Bandeja);

        // Obtener la colección real de variables
        SolidEdgeFramework.Variables variables = (Variables)seSheetMetal.Variables;

        // Obtener la lista de dimensiones
        SolidEdgeFramework.VariableList listOfDimensions2 = (VariableList)variables.Query(
            pFindCriterium: "*",
            NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
            VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeDimension);

        SolidEdgeFrameworkSupport.Dimension desX = (Dimension)listOfDimensions2.Item("TamañoCorteX_Modelo_Chapa_Desarrollada");
        Console.WriteLine($"Dimensión 'DESARROLLO X' después de modificar: {desX.Value}");

        SolidEdgeFrameworkSupport.Dimension desY = (Dimension)listOfDimensions2.Item("TamañoCorteY_Modelo_Chapa_Desarrollada");
        Console.WriteLine($"Dimensión 'DESARROLLO Y' después de modificar: {desY.Value}");
```

```
//partematerial
double pesobandeja = desX.Value * desY.Value * 8.1;
double preciomatbandeja = pesobandeja * 4.20;

//partepunzonado
double ngolpesx = 2 * Math.Round(desX.Value / 0.11);
double ngolpesy = 2 * Math.Round(desY.Value / 0.11);
double ngolpes = ngolpesx + ngolpesy + 22;
double tiempopreppunzo = 5;
double tiempopunzo = ngolpes / 80;
double preciopunzonar = 30;
double preciopiezapunzonada = ((tiempopreppunzo + tiempopunzo) / 60) * preciopunzonar;

//partedoblado
double nmontajes = 1.0;
double ndobles = 10.0;
double tiempomontaje = nmontajes * 15.0;
double tiempodoble = 0.0;
double precioplegar = 30.0;
if (desX.Value > 1.8)
{
   tiempodoble = 15.0 / 60.0;
}
else
{
   tiempodoble = 10.0 / 60.0;
}
double tiempodoblado = ndobles * tiempodoble;
double preciopiezadoblada = ((tiempodoblado + tiempomontaje) / 60.0) * precioplegar;

double preciopieza = preciopiezapunzonada + preciomatbandeja + preciopiezadoblada;
```

```
        Console.WriteLine($"El precio del material de la bandeja es: {preciomatbandeja}");
        Console.WriteLine($"El precio de punzonar la bandeja es: {preciopiezapunzonada}");
        Console.WriteLine($"El precio de doblar la bandeja es: {preciopiezadoblada}");
        Console.WriteLine($"El precio total de la bandeja es: {preciopieza}");

        System.Threading.Thread.Sleep(2000);

        seSheetMetal.Save();
        System.Threading.Thread.Sleep(1000);
        seSheetMetal.Close();
        System.Threading.Thread.Sleep(1000);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error al guardar el archivo DXF: " + ex.Message);
    }
    finally
    {
        OleMessageFilter.Unregister();
    }
    System.Threading.Thread.Sleep(1000);
}

static void PrecioFrontis(string Frontis)
{
    try
    {
        OleMessageFilter.Register();

        // Conexión a Solid Edge
        SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);
```

```
// Abrir el documento
SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(Frontis);

// Obtener la colección real de variables
SolidEdgeFramework.Variables variables = (Variables)seSheetMetal.Variables;

// Obtener la lista de dimensiones
SolidEdgeFramework.VariableList listOfDimensions2 = (VariableList)variables.Query(
    pFindCriterium: "*",
    NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
    VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeDimension);

SolidEdgeFrameworkSupport.Dimension desX = (Dimension)listOfDimensions2.Item("TamañoCorteX_Modelo_Chapa_Desarrollada");
Console.WriteLine($"Dimensión 'DESARROLLO X' después de modificar: {desX.Value}");

SolidEdgeFrameworkSupport.Dimension desY = (Dimension)listOfDimensions2.Item("TamañoCorteY_Modelo_Chapa_Desarrollada");
Console.WriteLine($"Dimensión 'DESARROLLO Y' después de modificar: {desY.Value}");

//partematerial
double pesofrontis = desX.Value * desY.Value * 8.1;
double preciomatfrontis = pesofrontis * 4.20;

//partepunzonado
double ngolpesx = 2 * Math.Round(desX.Value / 0.11);
double ngolpesy = 2 * Math.Round(desY.Value / 0.11);
double ngolpes = ngolpesx + ngolpesy + 22;
double tiempopreppunzo = 5;
double tiempopunzo = ngolpes / 80;
double preciopunzonar = 30;
double preciopiezapunzonada = ((tiempopreppunzo + tiempopunzo) / 60) * preciopunzonar;
```

```
//partedoblado
double nmontajes = 1.0;
double ndobles = 5.0;
double tiempomontaje = nmontajes * 15.0;
double tiempodoble = 0.0;
double precioplegar = 30.0;

if (desX.Value > 1.8)
{
   tiempodoble = 15.0 / 60.0;
}
else
{
   tiempodoble = 10.0 / 60.0;
}

double tiempodoblado = ndobles * tiempodoble;
double preciopiezadoblada = ((tiempodoblado + tiempomontaje) / 60.0) * precioplegar;

double preciopieza = preciopiezapunzonada + preciomatfrontis + preciopiezadoblada;

Console.WriteLine($"El precio del material el frontis es: {preciomatfrontis}");
Console.WriteLine($"El precio de punzonar el frontis es: {preciopiezapunzonada}");
Console.WriteLine($"El precio de doblar el frontis es: {preciopiezadoblada}");
Console.WriteLine($"El precio total del frontis es: {preciopieza}");

System.Threading.Thread.Sleep(2000);

seSheetMetal.Save();
System.Threading.Thread.Sleep(1000);
seSheetMetal.Close();
System.Threading.Thread.Sleep(1000);
```

```
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error al guardar el archivo DXF: " + ex.Message);
    }
    finally
    {
        OleMessageFilter.Unregister();
    }
    System.Threading.Thread.Sleep(1000);
}

static void PrecioVisera(string Visera)
{
    try
    {
        OleMessageFilter.Register();

        // Conexión a Solid Edge
        SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

        // Abrir el documento
        SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
        SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(Visera);

        // Obtener la colección real de variables
        SolidEdgeFramework.Variables variables = (Variables)seSheetMetal.Variables;

        // Obtener la lista de dimensiones
        SolidEdgeFramework.VariableList listOfDimensions2 = (VariableList)variables.Query(
            pFindCriterium: "*",
            NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
```

```
    VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeDimension);

SolidEdgeFrameworkSupport.Dimension desX = (Dimension)listOfDimensions2.Item("TamañoCorteX_Modelo_Chapa_Desarrollada");
Console.WriteLine($"Dimensión 'DESARROLLO X' después de modificar: {desX.Value}");

SolidEdgeFrameworkSupport.Dimension desY = (Dimension)listOfDimensions2.Item("TamañoCorteY_Modelo_Chapa_Desarrollada");
Console.WriteLine($"Dimensión 'DESARROLLO Y' después de modificar: {desY.Value}");

//partematerial
double pesovisera = desX.Value * desY.Value * 8.1;
double preciomatvisera = pesovisera * 4.20;

//partepunzonado
double ngolpesx = 2 * Math.Round(desX.Value / 0.11);
double ngolpesy = 2 * Math.Round(desY.Value / 0.11);
double ngolpes = ngolpesx + ngolpesy + 35;
double tiempopreppunzo = 5;
double tiempopunzo = ngolpes / 80;
double preciopunzonar = 30;
double preciopiezapunzonada = ((tiempopreppunzo + tiempopunzo) / 60) * preciopunzonar;

//partedoblado
double nmontajes = 1.0;
double ndobles = 10.0;
double tiempomontaje = nmontajes * 15.0;
double tiempodoble = 0.0;
double precioplegar = 30.0;

if (desX.Value > 1.8)
{
    tiempodoble = 25.0 / 60.0;
}
```

```
        else
        {
            tiempodoble = 20.0 / 60.0;
        }

        double tiempodoblado = ndobles * tiempodoble;
        double preciopiezadoblada = ((tiempodoblado + tiempomontaje) / 60.0) * precioplegar;

        double preciopieza = preciopiezapunzonada + preciomatvisera + preciopiezadoblada;

        Console.WriteLine($"El precio del material de la visera es: {preciomatvisera}");
        Console.WriteLine($"El precio de punzonar la visera es: {preciopiezapunzonada}");
        Console.WriteLine($"El precio de doblar la visera es: {preciopiezadoblada}");
        Console.WriteLine($"El precio total de la visera es: {preciopieza}");

        System.Threading.Thread.Sleep(2000);

        seSheetMetal.Save();
        System.Threading.Thread.Sleep(1000);
        seSheetMetal.Close();
        System.Threading.Thread.Sleep(1000);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error al guardar el archivo DXF: " + ex.Message);
    }
    finally
    {
        OleMessageFilter.Unregister();
    }
    System.Threading.Thread.Sleep(1000);
}
```

```
static void PrecioTrasera(string Trasera)
{
    try
    {
        OleMessageFilter.Register();

        // Conexión a Solid Edge
        SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

        // Abrir el documento
        SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
        SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(Trasera);

        // Obtener la colección real de variables
        SolidEdgeFramework.Variables variables = (Variables)seSheetMetal.Variables;

        // Obtener la lista de dimensiones
        SolidEdgeFramework.VariableList listOfDimensions2 = (VariableList)variables.Query(
            pFindCriterium: "*",
            NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
            VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeDimension);

        SolidEdgeFrameworkSupport.Dimension desX = (Dimension)listOfDimensions2.Item("TamañoCorteX_Modelo_Chapa_Desarrollada");
        Console.WriteLine($"Dimensión 'DESARROLLO X' después de modificar: {desX.Value}");

        SolidEdgeFrameworkSupport.Dimension desY = (Dimension)listOfDimensions2.Item("TamañoCorteY_Modelo_Chapa_Desarrollada");
        Console.WriteLine($"Dimensión 'DESARROLLO Y' después de modificar: {desY.Value}");

        //partematerial
        double pesotrasera = desX.Value * desY.Value * 8.1;
        double preciomattrasera = pesotrasera * 1.20;
```

```
//partepunzonado
double ngolpesx = 2 * Math.Round(desX.Value / 0.11);
double ngolpesy = 2 * Math.Round(desY.Value / 0.11);
double ngolpes = ngolpesx + ngolpesy + 25;
double tiempopreppunzo = 5;
double tiempopunzo = ngolpes / 80;
double preciopunzonar = 30;
double preciopiezapunzonada = ((tiempopreppunzo + tiempopunzo) / 60) * preciopunzonar;

//partedoblado
double nmontajes = 1.0;
double ndobles = 4.0;
double tiempomontaje = nmontajes * 15.0;
double tiempodoble = 0.0;
double precioplegar = 30.0;

if (desX.Value > 1.8)
{
   tiempodoble = 25.0 / 60.0;
}
else
{
   tiempodoble = 20.0 / 60.0;
}

double tiempodoblado = ndobles * tiempodoble;
double preciopiezadoblada = ((tiempodoblado + tiempomontaje) / 60.0) * precioplegar;

double preciopieza = preciopiezapunzonada + preciomattrasera + preciopiezadoblada;

Console.WriteLine($"El precio del material de la trasera es: {preciomattrasera}");
Console.WriteLine($"El precio de punzonar la trasera es: {preciopiezapunzonada}");
```

```
        Console.WriteLine($"El precio de doblar la trasera es: {preciopiezadoblada}");
        Console.WriteLine($"El precio total de la trasera es: {preciopieza}");

        System.Threading.Thread.Sleep(2000);

        seSheetMetal.Save();
        System.Threading.Thread.Sleep(1000);
        seSheetMetal.Close();
        System.Threading.Thread.Sleep(1000);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error al guardar el archivo DXF: " + ex.Message);
    }
    finally
    {
        OleMessageFilter.Unregister();
    }
    System.Threading.Thread.Sleep(1000);
}

static void PrecioLateral(string Lateral)
{
    try
    {
        OleMessageFilter.Register();

        // Conexión a Solid Edge
        SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

        // Abrir el documento
        SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
```

```
SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(Lateral);

// Obtener la colección real de variables
SolidEdgeFramework.Variables variables = (Variables)seSheetMetal.Variables;

// Obtener la lista de dimensiones
SolidEdgeFramework.VariableList listOfDimensions2 = (VariableList)variables.Query(
    pFindCriterium: "*",
    NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
    VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeDimension);

SolidEdgeFrameworkSupport.Dimension desX = (Dimension)listOfDimensions2.Item("TamañoCorteX_Modelo_Chapa_Desarrollada");
Console.WriteLine($"Dimensión 'DESARROLLO X' después de modificar: {desX.Value}");

SolidEdgeFrameworkSupport.Dimension desY = (Dimension)listOfDimensions2.Item("TamañoCorteY_Modelo_Chapa_Desarrollada");
Console.WriteLine($"Dimensión 'DESARROLLO Y' después de modificar: {desY.Value}");

//partematerial
double pesolateral = desX.Value * desY.Value * 8.1;
double preciomatlateral = pesolateral * 4.20;

//partepunzonado
double ngolpesx = 2 * Math.Round(desX.Value / 0.11);
double ngolpesy = 2 * Math.Round(desY.Value / 0.11);
double ngolpes = ngolpesx + ngolpesy + 20;
double tiempopreppunzo = 5;
double tiempopunzo = ngolpes / 80;
double preciopunzonar = 30;
double preciopiezapunzonada = ((tiempopreppunzo + tiempopunzo) / 60) * preciopunzonar;

//partedoblado
double nmontajes = 1.0;
```

```
double ndobles = 8.0;
double tiempomontaje = nmontajes * 15.0;
double tiempodoble = 0.0;
double precioplegar = 30.0;

if (desX.Value > 1.3)
{
   tiempodoble = 15.0 / 60.0;
}
else
{
   tiempodoble = 10.0 / 60.0;
}

double tiempodoblado = ndobles * tiempodoble;
double preciopiezadoblada = ((tiempodoblado + tiempomontaje) / 60.0) * precioplegar;

double preciopieza = preciopiezapunzonada + preciomatlateral + preciopiezadoblada;
double preciopiezas = preciopieza * 2;

Console.WriteLine($"El precio del material del lateral individual es: {preciomatlateral}");
Console.WriteLine($"El precio del material de los dos laterales es: {preciomatlateral * 2}");
Console.WriteLine($"El precio de punzonar el lateral es: {preciopiezapunzonada}");
Console.WriteLine($"El precio de punzonar los dos laterales es: {preciopiezapunzonada * 2}");
Console.WriteLine($"El precio de doblar el lateral es: {preciopiezadoblada}");
Console.WriteLine($"El precio de doblar los dos laterales es: {preciopiezadoblada * 2}");
Console.WriteLine($"El precio total del lateral es: {preciopieza}");
Console.WriteLine($"El precio total de los dos laterales es: {preciopieza * 2}");

System.Threading.Thread.Sleep(2000);

seSheetMetal.Save();
```

```
        System.Threading.Thread.Sleep(1000);
        seSheetMetal.Close();
        System.Threading.Thread.Sleep(1000);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error al guardar el archivo DXF: " + ex.Message);
    }
    finally
    {
        OleMessageFilter.Unregister();
    }
    System.Threading.Thread.Sleep(1000);
}

static void PrecioDeflectorAirdoor(string DeflectorAirdoor)
{
    try
    {
        OleMessageFilter.Register();

        // Conexión a Solid Edge
        SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

        // Abrir el documento
        SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
        SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(DeflectorAirdoor);

        // Obtener la colección real de variables
        SolidEdgeFramework.Variables variables = (Variables)seSheetMetal.Variables;

        // Obtener la lista de dimensiones
```

```
SolidEdgeFramework.VariableList listOfDimensions2 = (VariableList)variables.Query(
    pFindCriterium: "*",
    NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
    VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeDimension);

SolidEdgeFrameworkSupport.Dimension desX = (Dimension)listOfDimensions2.Item("TamañoCorteX_Modelo_Chapa_Desarrollada");
Console.WriteLine($"Dimensión 'DESARROLLO X' después de modificar: {desX.Value}");

SolidEdgeFrameworkSupport.Dimension desY = (Dimension)listOfDimensions2.Item("TamañoCorteY_Modelo_Chapa_Desarrollada");
Console.WriteLine($"Dimensión 'DESARROLLO Y' después de modificar: {desY.Value}");

//partematerial
double pesodeflector = desX.Value * desY.Value * 8.1;
double preciomatdeflector = pesodeflector * 4.20;

//partepunzonado
double ngolpesx = 2 * Math.Round(desX.Value / 0.11);
double ngolpesy = 2 * Math.Round(desY.Value / 0.11);
double ngolpes = ngolpesx + ngolpesy + 25;
double tiempopreppunzo = 5;
double tiempopunzo = ngolpes / 80;
double preciopunzonar = 30;
double preciopiezapunzonada = ((tiempopreppunzo + tiempopunzo) / 60) * preciopunzonar;

//partedoblado
double nmontajes = 1.0;
double ndobles = 5.0;
double tiempomontaje = nmontajes * 15.0;
double tiempodoble = 0.0;
double precioplegar = 30.0;

if (desX.Value > 1.8)
```

```
    {
        tiempodoble = 20.0 / 60.0;
    }
    else
    {
        tiempodoble = 15.0 / 60.0;
    }

    double tiempodoblado = ndobles * tiempodoble;
    double preciopiezadoblada = ((tiempodoblado + tiempomontaje) / 60.0) * precioplegar;

    double preciopieza = preciopiezapunzonada + preciomatdeflector + preciopiezadoblada;
    double preciopiezas = preciopieza * 2;

    Console.WriteLine($"El precio del material del deflector es: {preciomatdeflector}");
    Console.WriteLine($"El precio de punzonar el deflector es: {preciopiezapunzonada}");
    Console.WriteLine($"El precio de doblar el deflector es: {preciopiezadoblada}");
    Console.WriteLine($"El precio total del deflector es: {preciopieza}");

    System.Threading.Thread.Sleep(2000);

    seSheetMetal.Save();
    System.Threading.Thread.Sleep(1000);
    seSheetMetal.Close();
    System.Threading.Thread.Sleep(1000);
}
catch (Exception ex)
{
    Console.WriteLine("Error al guardar el archivo DXF: " + ex.Message);
}
finally
{
```

```
        OleMessageFilter.Unregister();
    }
    System.Threading.Thread.Sleep(1000);
}

static void PrecioVigueta(string Vigueta)
{
    try
    {
        OleMessageFilter.Register();

        // Conexión a Solid Edge
        SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

        // Abrir el documento
        SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
        SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(Vigueta);

        // Obtener la colección real de variables
        SolidEdgeFramework.Variables variables = (Variables)seSheetMetal.Variables;

        // Obtener la lista de dimensiones
        SolidEdgeFramework.VariableList listOfDimensions2 = (VariableList)variables.Query(
            pFindCriterium: "*",
            NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
            VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeDimension);

        SolidEdgeFrameworkSupport.Dimension desX = (Dimension)listOfDimensions2.Item("TamañoCorteX_Modelo_Chapa_Desarrollada");
        Console.WriteLine($"Dimensión 'DESARROLLO X' después de modificar: {desX.Value}");

        SolidEdgeFrameworkSupport.Dimension desY = (Dimension)listOfDimensions2.Item("TamañoCorteY_Modelo_Chapa_Desarrollada");
        Console.WriteLine($"Dimensión 'DESARROLLO Y' después de modificar: {desY.Value}");
```

```
//partematerial
double pesovigueta = desX.Value * desY.Value * 8.1;
double preciomatvigueta = pesovigueta * 4.20;

//partepunzonado
double ngolpesx = 2 * Math.Round(desX.Value / 0.11);
double ngolpesy = 2 * Math.Round(desY.Value / 0.11);
double ngolpes = ngolpesx + ngolpesy + 4;
double tiempopreppunzo = 5;
double tiempopunzo = ngolpes / 80;
double preciopunzonar = 30;
double preciopiezapunzonada = ((tiempopreppunzo + tiempopunzo) / 60) * preciopunzonar;

//partedoblado
double nmontajes = 1.0;
double ndobles = 4.0;
double tiempomontaje = nmontajes * 15.0;
double tiempodoble = 0.0;
double precioplegar = 30.0;

if (desX.Value > 1.3)
{
   tiempodoble = 15.0 / 60.0;
}
else
{
   tiempodoble = 10.0 / 60.0;
}

double tiempodoblado = ndobles * tiempodoble;
double preciopiezadoblada = ((tiempodoblado + tiempomontaje) / 60.0) * precioplegar;
```

```
        double preciopieza = preciopiezapunzonada + preciomatvigueta + preciopiezadoblada;
        double preciopiezas = preciopieza * 2;

        Console.WriteLine($"El precio del material de la vigueta individual es: {preciomatvigueta}");
        Console.WriteLine($"El precio del material de las dos viguetas es: {preciomatvigueta * 2}");
        Console.WriteLine($"El precio de punzonar la vigueta es: {preciopiezapunzonada}");
        Console.WriteLine($"El precio de punzonar las dos viguetas es: {preciopiezapunzonada * 2}");
        Console.WriteLine($"El precio de doblar la vigueta es: {preciopiezadoblada}");
        Console.WriteLine($"El precio de doblar las dos viguetas es: {preciopiezadoblada * 2}");
        Console.WriteLine($"El precio total de la vigueta es: {preciopieza}");
        Console.WriteLine($"El precio total de las dos viguetas es: {preciopieza * 2}");

        System.Threading.Thread.Sleep(2000);

        seSheetMetal.Save();
        System.Threading.Thread.Sleep(1000);
        seSheetMetal.Close();
        System.Threading.Thread.Sleep(1000);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error al guardar el archivo DXF: " + ex.Message);
    }
    finally
    {
        OleMessageFilter.Unregister();
    }
    System.Threading.Thread.Sleep(1000);
}

static void PrecioChapaPlein(string ChapaPlein)
```

```
{
    try
    {
        OleMessageFilter.Register();

        // Conexión a Solid Edge
        SolidEdgeFramework.Application seApplication = SolidEdgeCommunity.SolidEdgeUtils.Connect(true);

        // Abrir el documento
        SolidEdgeFramework.Documents seDocuments = seApplication.Documents;
        SolidEdgePart.SheetMetalDocument seSheetMetal = (SheetMetalDocument)seDocuments.Open(ChapaPlein);

        // Obtener la colección real de variables
        SolidEdgeFramework.Variables variables = (Variables)seSheetMetal.Variables;

        // Obtener la lista de dimensiones
        SolidEdgeFramework.VariableList listOfDimensions2 = (VariableList)variables.Query(
            pFindCriterium: "*",
            NamedBy: SolidEdgeConstants.VariableNameBy.seVariableNameByBoth,
            VarType: SolidEdgeConstants.VariableVarType.SeVariableVarTypeDimension);

        SolidEdgeFrameworkSupport.Dimension desX = (Dimension)listOfDimensions2.Item("TamañoCorteX_Modelo_Chapa_Desarrollada");
        Console.WriteLine($"Dimensión 'DESARROLLO X' después de modificar: {desX.Value}");

        SolidEdgeFrameworkSupport.Dimension desY = (Dimension)listOfDimensions2.Item("TamañoCorteY_Modelo_Chapa_Desarrollada");
        Console.WriteLine($"Dimensión 'DESARROLLO Y' después de modificar: {desY.Value}");


        //partematerial
        double pesochapaplein = desX.Value * desY.Value * 8.1;
        double preciomatchapaplein = pesochapaplein * 1.20;
```

```
//partepunzonado
double ngolpesx = 2 * Math.Round(desX.Value / 0.11);
double ngolpesy = 2 * Math.Round(desY.Value / 0.11);
double ngolpes = ngolpesx + ngolpesy + 4;
double tiempopreppunzo = 5;
double tiempopunzo = ngolpes / 80;
double preciopunzonar = 30;
double preciopiezapunzonada = ((tiempopreppunzo + tiempopunzo) / 60) * preciopunzonar;

//partedoblado
double nmontajes = 1.0;
double ndobles = 4.0;
double tiempomontaje = nmontajes * 15.0;
double tiempodoble = 0.0;
double precioplegar = 30.0;

if (desX.Value > 1.8)
{
   tiempodoble = 20.0 / 60.0;
}
else
{
   tiempodoble = 15.0 / 60.0;
}

double tiempodoblado = ndobles * tiempodoble;
double preciopiezadoblada = ((tiempodoblado + tiempomontaje) / 60.0) * precioplegar;

double preciopieza = preciopiezapunzonada + preciomatchapaplein + preciopiezadoblada;
double preciopiezas = preciopieza * 2;

Console.WriteLine($"El precio del material de la chapaplein es: {preciomatchapaplein}");
```

```
            Console.WriteLine($"El precio de punzonar la chapaplein es: {preciopiezapunzonada}");
            Console.WriteLine($"El precio de doblar la chapaplein es: {preciopiezadoblada}");
            Console.WriteLine($"El precio total de la chapa plein es: {preciopieza}");

            System.Threading.Thread.Sleep(50000);

            seSheetMetal.Save();
            System.Threading.Thread.Sleep(1000);
            seSheetMetal.Close();
            System.Threading.Thread.Sleep(1000);
        }
        catch (Exception ex)
        {
            Console.WriteLine("Error al guardar el archivo DXF: " + ex.Message);
        }
        finally
        {
            OleMessageFilter.Unregister();
        }
        System.Threading.Thread.Sleep(1000);
    }
  }
}
```