ANEXO I

Manual Usuario Heclia

INSTALACION:

Heclia se presenta en un archivo comprimido tipo Zip, el cual deberá ser extraído a disco duro del equipo en el que se desea la instalación. La descompresión dejara visible una cartera con denominación HECLIA_INSTALL, la cual contendrá dos archivos ejecutables de instalación:

- ***vc_redist.x86***: Se deberá instalar tal ejecutable que contienen bibliotecas y archivos necesarios para la ejecución de *Heclia*.
- ***HecliaSetup***: Archivo auto instalación de Heclia, que instalará el programa desarrollado en disco duro del equipo.

  La instalación no mostrará accesos directos de Heclia, siendo que en su caso el usuario puede crear uno directamente desde el icono de la Aplicación en la carpeta de instalación C:\\Heclia.

Una vez instalado en la propia carpeta de sistema en la que se instala Heclia, existe disponible icono de desinstalación del programa, el cual eliminará del disco todos los archivos de la aplicación.

No se eliminarán mediante desinstalación archivos generados tipo txt, ni otros archivos generados de bases de datos o proyectos guardados con terminaciones *.sfa, los cuales se deberán borrar manualmente.

Introducción:

- *Heclia* es un programa destinado al cálculo de sistemas de producción basados en fuentes renovales, suponiendo una interface gráfica mediante cuadros de dialogo intuitivos y de fácil desarrollo.

- El usuario deberá ir siguiendo las diferentes pantallas existentes, completando la información necesaria en cada cuadro de dialogo o casilla disponible. Existen casillas editables en las que se deberán introducir los datos y otras en las que el texto no será manipulable (en tono gris).

- Se debe seguir el orden de introducción de datos, empezando por la primera casilla de cada página, y siguiendo secuencialmente los cuadros de texto, no dejando elementos en blanco, o saltando entre pantallas antes de finalizar cada una de las vista de ejecución (en caso contrario se podrían producir cálculos parciales o incorrectos de algunos apartados enlazados).

- El desarrollo o paso entre casillas o cuadros de texto, se aconseja que sea mediante paso de tabulación, siendo que tal actuación, guiará al usuario de manera automática y secuencial por cada apartado en el orden que debe completar la información a introducir.

- El paso mediante tabulación, activa los automatismos internos del programa, garantizando la correcta ejecución interna del proceso de cálculo.

- En caso de realizar navegación por cuadros (retroceder-adelantar pantallas a fin de revisar información introducida previamente o similares), el programa guardará la información introducida previamente, resultando que en caso de efectuarse modificaciones en pantallas o apartados previos, al retornar a pantallas posteriores, se conseja realizar paso de tabulación desde la primera casilla hasta la última antes de seguir avanzando (a fin de garantizar que todas las operaciones han sido realizadas con los datos modificados y no con los preexistentes antes de retroceder en la aplicación).

- Si no se modifican datos en pantallas previas, no será necesario realizar pasos de tabulación en pantallas posteriores (dado que la información preexistente no habrá sido alterada o modificada).

Pantalla principal Heclia:

La pantalla principal de *Heclia*, proporciona acceso a cada una de las acciones o métodos de cálculo de instalaciones del aplicativo, pudiéndose selección la opción requerida por el usuario, a fin de realizar el cálculo de una instalación concreta u obtener información disponible en el programa.

Figura 1: Pantalla inicial aplicación

Las opciones disponibles serán:

- *Abrir proyecto existente:* Mediante la presente función, se podrán buscar proyectos ejecutados en *Heclia*, recuperando la información de tal proyecto para poder continuar con el mismo, realizar modificaciones, etc…
- *Fotovoltaica Aislada:* Lanza el aplicativo desarrollado para el cálculo de instalaciones fotovoltaicas aisladas, sin conexión a red eléctrica.
- *Fotovoltaica conectada a red:* Lanza el aplicativo desarrollado para el cálculo de instalaciones fotovoltaicas conectadas a la red eléctrica de distribución. eléctrica.
- *Generación Mixta:* Aplicación no desarrollada en versión actual.
- *Aerogeneración:* Aplicación no desarrollada en versión actual.
- *Base de datos:* Permite acceder, crear y/o manipular bases de datos de materiales y elementos que componen las instalaciones fotovoltaicas, pudiendo consultar o verificar datos de determinados equipos a fin de poderlos utilizar en el cálculo de instalaciones.

Abrir proyecto existente:

La apertura de archivos se basa en un cuadro de dialogo estándar que permite buscar los archivos guardados en la carpeta que el usuario los tenga almacenados, seleccionándolos y recuperando los datos de los archivos almacenados en disco duro o dispositivos móviles.

Heclia utiliza archivos *"*.sfa"* que almacenan los datos de cada sesión o proyecto guardado. Para recuperar dicho proyecto bastará con seleccionarlo y abrir. El proyecto no se abre en una ventana concreta (estos es, no se lanza la aplicación específica), sino que carga los datos, de tal manera que posteriormente al entrar en el módulo deseado (Fotovoltaica asilada, red, etc…) los datos del proyecto estarán cargados en cada una de las casillas o elementos de la aplicación. Una vez abierto el proyecto, el usuario deberá entrar en el módulo correspondiente al mismo para visualizar la información.

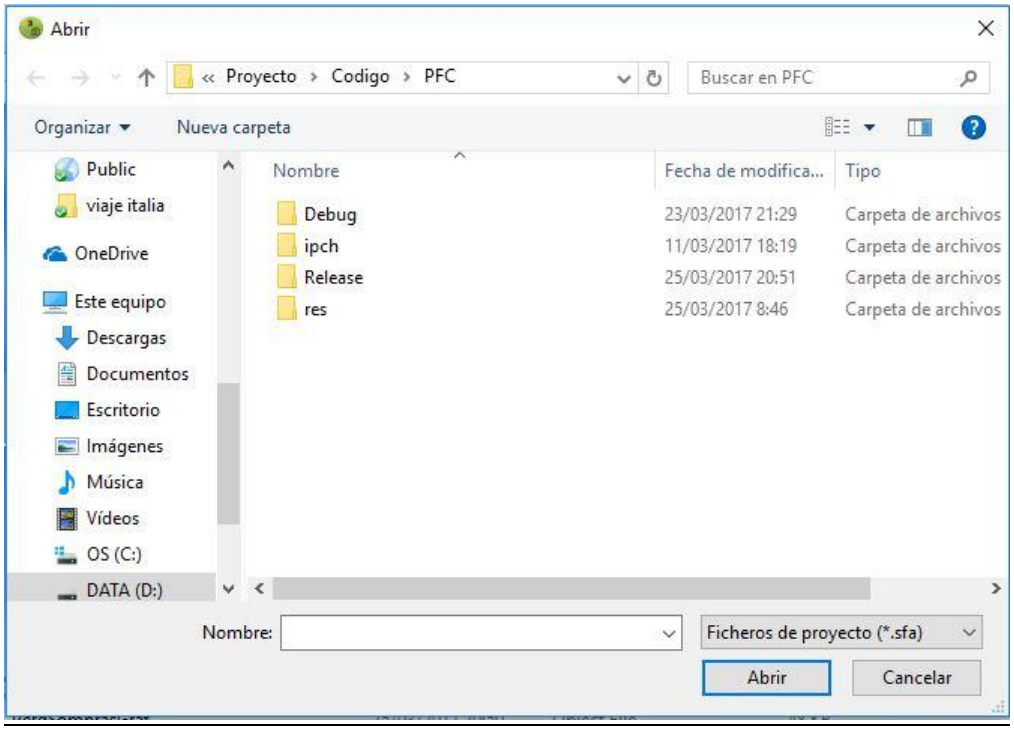

Figura 2: Cuadro dialogo abrir proyecto

Existen archivos asociados al proyecto con terminación *"*.sfa.dat"*, los cuales son archivos internos de sistema que no se deben abrir, dado que los mismos no cargarán la información del proyecto, aun cuando el nombre que reflejan, sea idéntico al archivo asociado al proyecto con terminación *"*.sfa"*.

Fotovoltaica Aislada:

La primera pantalla del aplicativo para cálculo de instalaciones aisladas, requerirá al usuario la introducción de datos de los puntos de consumo de las instalaciones sobre las que se desea calcular el consumo energético.

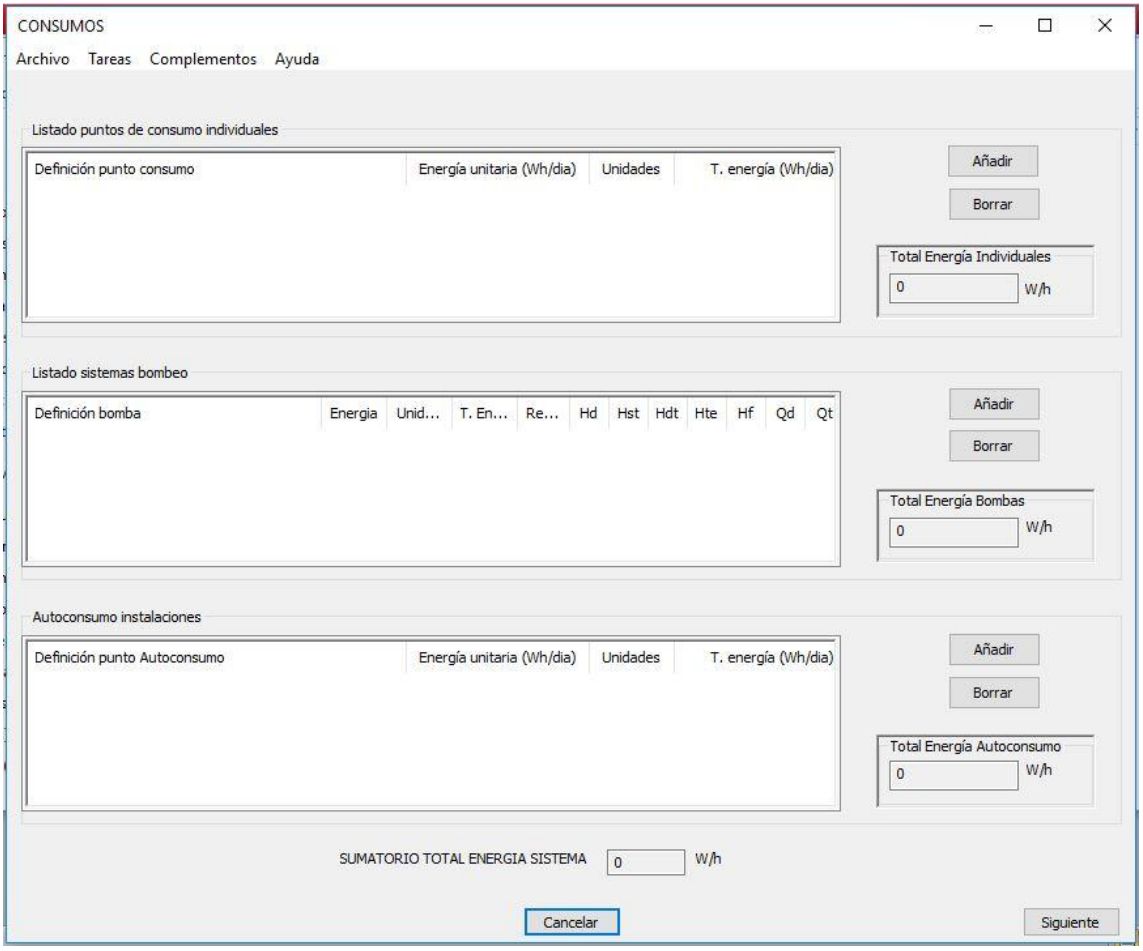

Figura 3: Cuadro dialogo puntos consumo energético

Los consumos energéticos se dividen en tres apartados, de los cuales en cada uno de ellos, la información necesaria quedará reflejada en el cuadro de dialogo correspondientes. Tales puntos de consumo son:

- *Puntos de consumo individuales*: Se insertará todo punto de consumo diferente de una bomba que esté integrado en la instalación o explotación agropecuaria (elevadores de cangilones, motores, autómatas, etc…).

  La introducción de datos se realizar mediante el botón *"Añadir"*, siendo que al pulsarlo se abrirá un cuadro de dialogo que requerirá que el usuario introduzca los datos característicos de cada elemento concreto (descripción de punto de

consumo, unidades de tal punto, energía consumida por unidad de elemento, etc..).

La información de cada elemento, quedará reflejada en su apartado, siendo que el usuario puede eliminar un punto concreto, seleccionándolo de la lista mediante el ratón, y pulsando posteriormente *"Borrar"*, eliminándose tal elemento de manera definitiva.

La aplicación realizará el sumatorio de consumos totales de este tipo de apartado ("*Total consumos individuales*"), reflejando la energía total en casilla bajo botones de inserción y borrado.

- *Sistemas de bombeo*: Se insertaran todos los elementos de bombeo de la aplicación. La metodología de inserción y cálculo será análoga al apartado anterior

- *Puntos de autoconsumo*: Se insertará todo punto de la propia instalación fotovoltaica y equipamiento que tengan autoconsumo (inversores, etc…). La metodología de inserción y cálculo será análoga al apartado anterior

La siguiente pantalla guiara al usuario a través del cálculo de dimensionado de generador fotovoltaico, dividiéndose en tres apartados específicos:

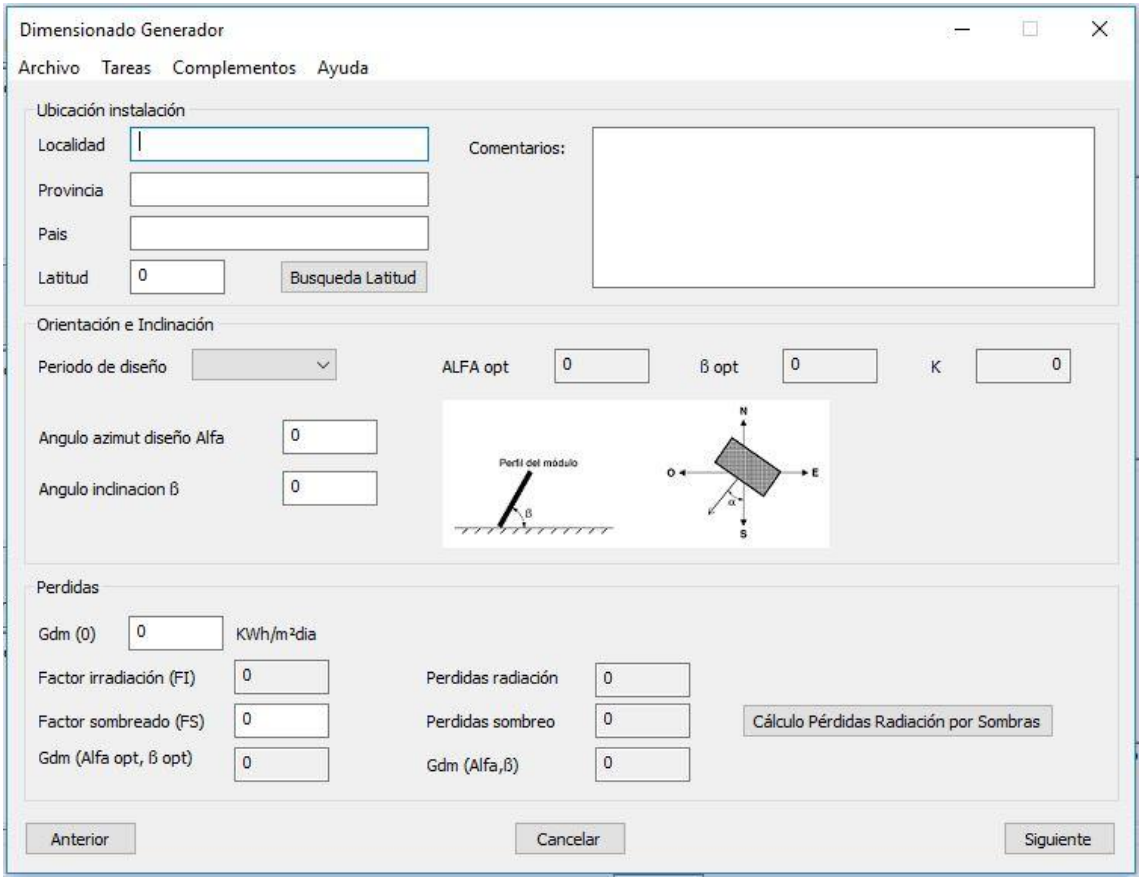

Figura 4: Cuadro dialogo dimensionado generador

- *Ubicación de la instalación:* Se permitirá introducir textualmente la ubicación de la instalación, así como generar los comentarios textuales que puedan ayudar al usuario a identificar la instalaciones, características de esta o cuantos datos considere de relevancia.

  Con respecto a latitud de la instalación, el usuario puede introducir de manera directa un valor numérico, o en su caso lanzar la aplicación GoogleMaps ("*Búsqueda latitud*") para búsqueda y localización de la latitud (que una vez localizada deberá ser introducida de manera manual).

- *Orientación e inclinación*: El usuario seleccionará el periodo más desfavorable de la instalación en el cuadro desplegable "*periodo de diseño*", pudiendo optar entre junio, diciembre o anual en caso de ser un sistema homogéneo todo el año. Además se deberá introducir el ángulo azimut de la instalación y la inclinación de los panales (inclinación total, debiéndose sumar en su caso inclinación de estructura de apoyo + cubierta).

- *Perdidas:* Las perdidas por orientación se calcularán de manera automática, siendo que las pérdidas por sombreo, se podrán introducir de manera directa, o en su caso utilizar "*cálculo perdidas radiación por sombra*" para realizar de manera automatizada el cálculo (ver apartado pérdidas por sombras).

La continuación del aplicativo, mediante el paso por "*Siguiente*", conllevará el lanzamiento del cuadro de dialogo "*Dimensionado Sistema*", en el que el usuario deberá seguir introduciendo los valores requeridos por orden de tabulación.

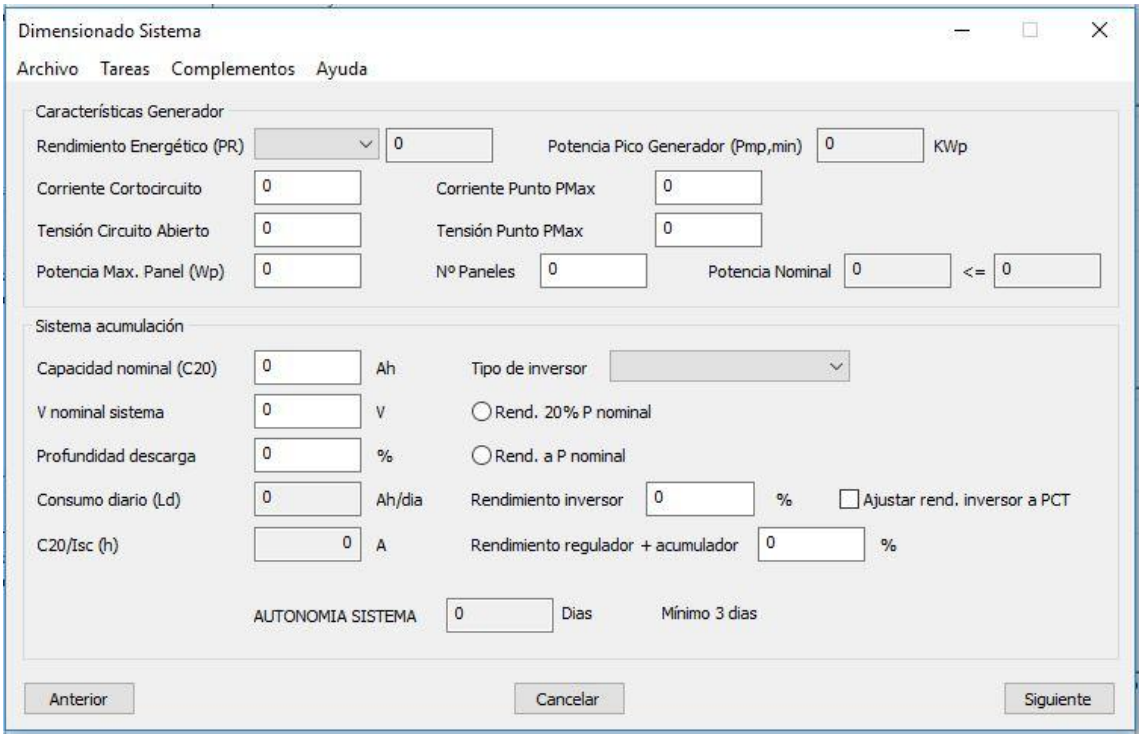

Figura 5: Cuadro dialogo dimensionado generador

El usuario introducirá los datos del generador o paneles requeridos, en el que se describirán su valores característicos de intensidades y tensiones de funcionamiento según catalogo (consultables en bases de datos del programa), así como la potencia de cada panel y el número de paneles que configuraran la instalación (el propio programa indicará si se cumple la norma técnica o si por el contrario se supera la potencia pico del generador).

Del mismo modo y posteriormente se completaran los datos de los acumuladores, sistema, rendimientos de inversores, sistema de regulación–acumulador, etc…, indicando el aplicativo si se cumplen los requisitos previstos de autonomía.

La continuación requerirá del usuario seleccionar el tipo de estructura que portará los paneles, pudiendo ser esta de tipo seguidor (no implementando su cálculo en la presente versión del programa), o estructura fija, sobre la que se podrá calcular la sobrecarga que produce en la cubierta en la que se instala, así como su respuesta a acciones externas como viento.

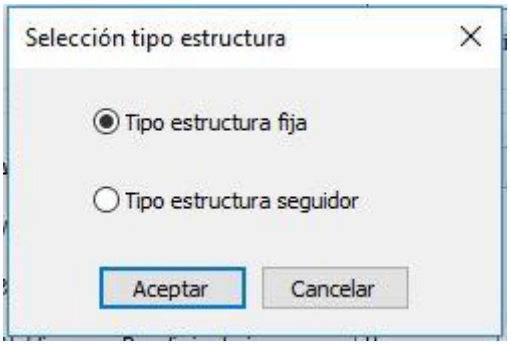

Figura 6: Cuadro dialogo tipo estructura

El cálculo de las cargas sobre cubierta, se realizarán en función de las fuerzas actuantes, resultando estas mayoritariamente el viento, dependiente este de la zona geográfica (ver zona en diagrama "*ver zonas*"), la altura de la instalación, el grado de aspereza del entorno y el coeficiente eólico (datos a completar por el usuario), y aportando en función de tal información la acción del viento sobre la instalación de cubierta.

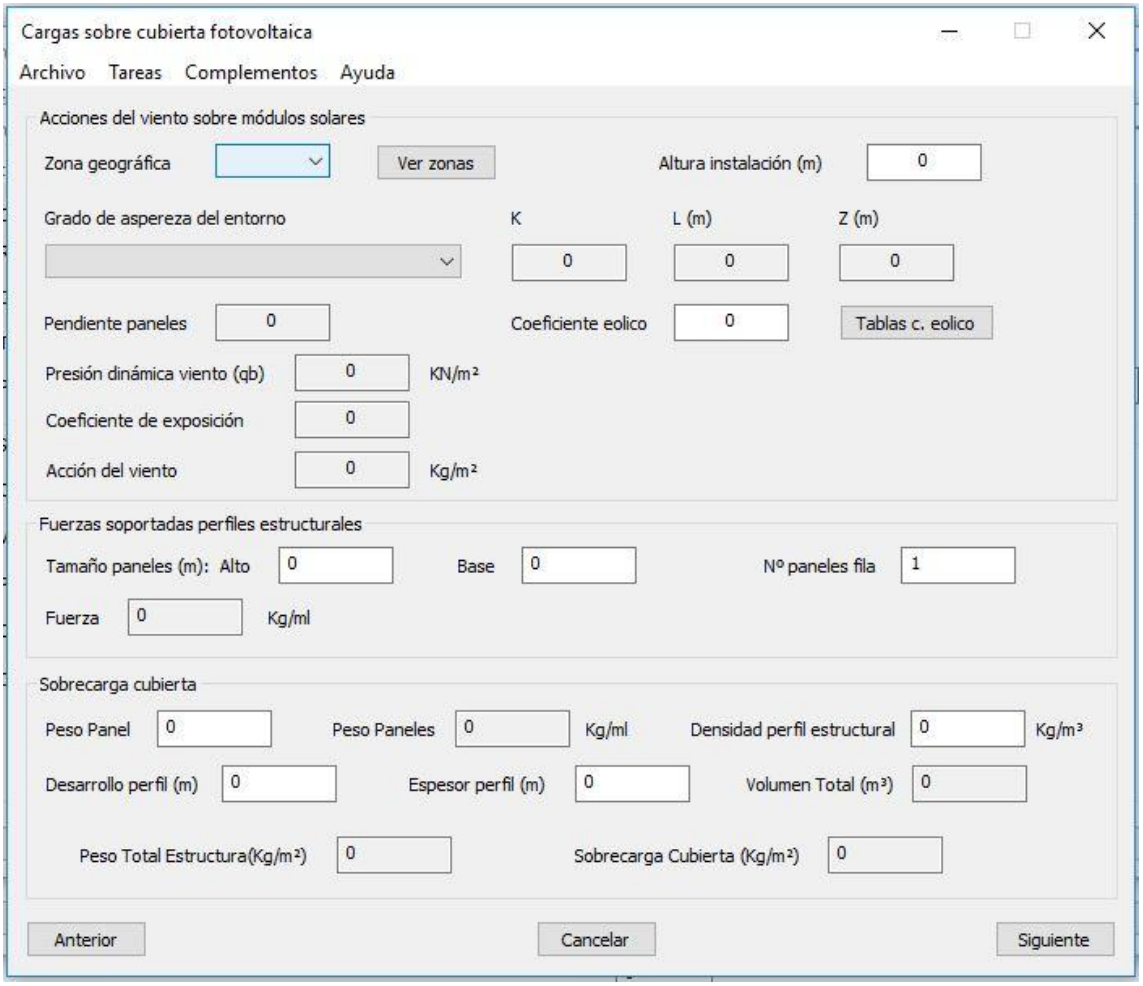

Figura 7: Cuadro dialogo cargas cubierta

Así mismo también se calcularán las fuerzas sobre los perfiles estructurales de los paneles, y la sobrecarga global de la cubierta, en la cual únicamente se indicará la carga extra derivada de la instalación de la cubierta solar, en ningún caso se realizarán cálculos estructurales de la propia cubierta o resistencia mecánica necesaria de ésta.

Finalmente en el proceso de cálculo, se analizará la inversión económica de la instalación, así como la rentabilidad de la instalación, en función de los costes de cada uno de los parámetros intervinientes y costes de los mismos.

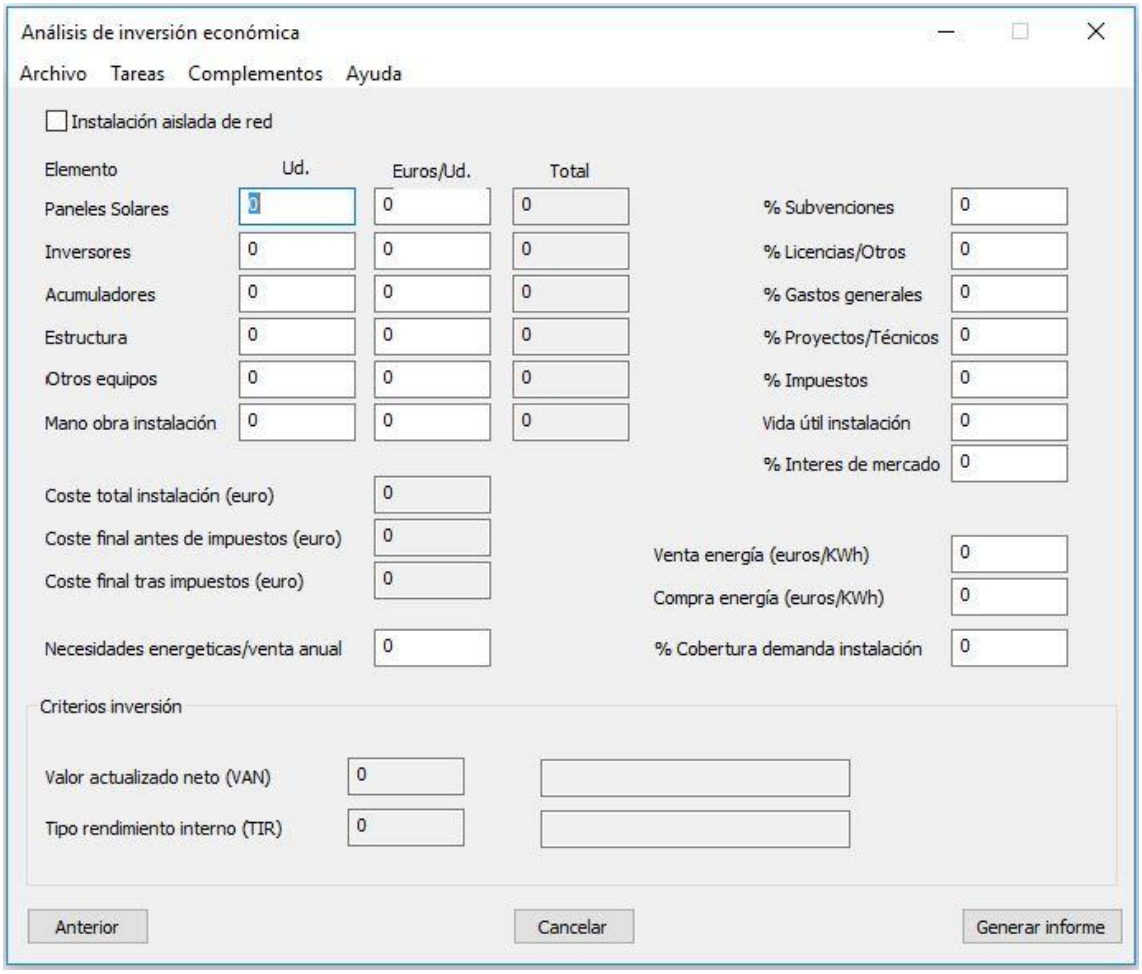

Figura 8: Cuadro dialogo análisis inversión

La aplicación permitirá calcular o selección si la instalación es aislada de red (marcando casilla usuario), en la cual el total de la energía es para autoconsumo, o si en su caso existe venta de energía.

- En el caso de ser una instalación conectada a red, se asume que la totalidad de energía es vendida, siendo por tanto que las necesidades energéticas/venta anual (en KWh), se multiplicarán por el coste de venta a fin de analizar la rentabilidad de la inversión.
- En el caso de una instalación aislada de red, no existirá venta, pero si unas necesidades energéticas que deberán ser satisfechas, permitiendo el programa que se estipule un porcentaje de cobertura de autodemanda. Esto es, existirá parte de la demanda que requerirá compra de energía con un coste y parte de la cobertura de las necesidades que se cubrirán con costes producción de la instalación, lo cual conllevará una reducción del coste energético total de la empresa y por tanto se considerará a efectos de cálculo de rentabilidades.

El programa realizará el análisis de la inversión en función del criterio del VAN y del TIR, informando al usuario sobre si la inversión debería realizarse o no, atendiendo exclusivamente a tales parámetros y rentabilidad de la inversión.

La finalización del cálculo mediante "*Generar informe*", lanzará un documento "*txt*", que el usuario podrá modificar y/o guardar.

Fotovoltaica conectada a red:

La primera pantalla del aplicativo para cálculo de instalaciones conectadas a red, requerirá al usuario la introducción de datos de los puntos de consumo de las instalaciones que compone el riesgo, a efectos de cálculos estimados de requerimientos energéticos. Al no existir aprovechamiento interno de la energía generada, completar esta pantalla no será necesario, aunque si requerido si se desea luego aparezca tal información en el informe final.

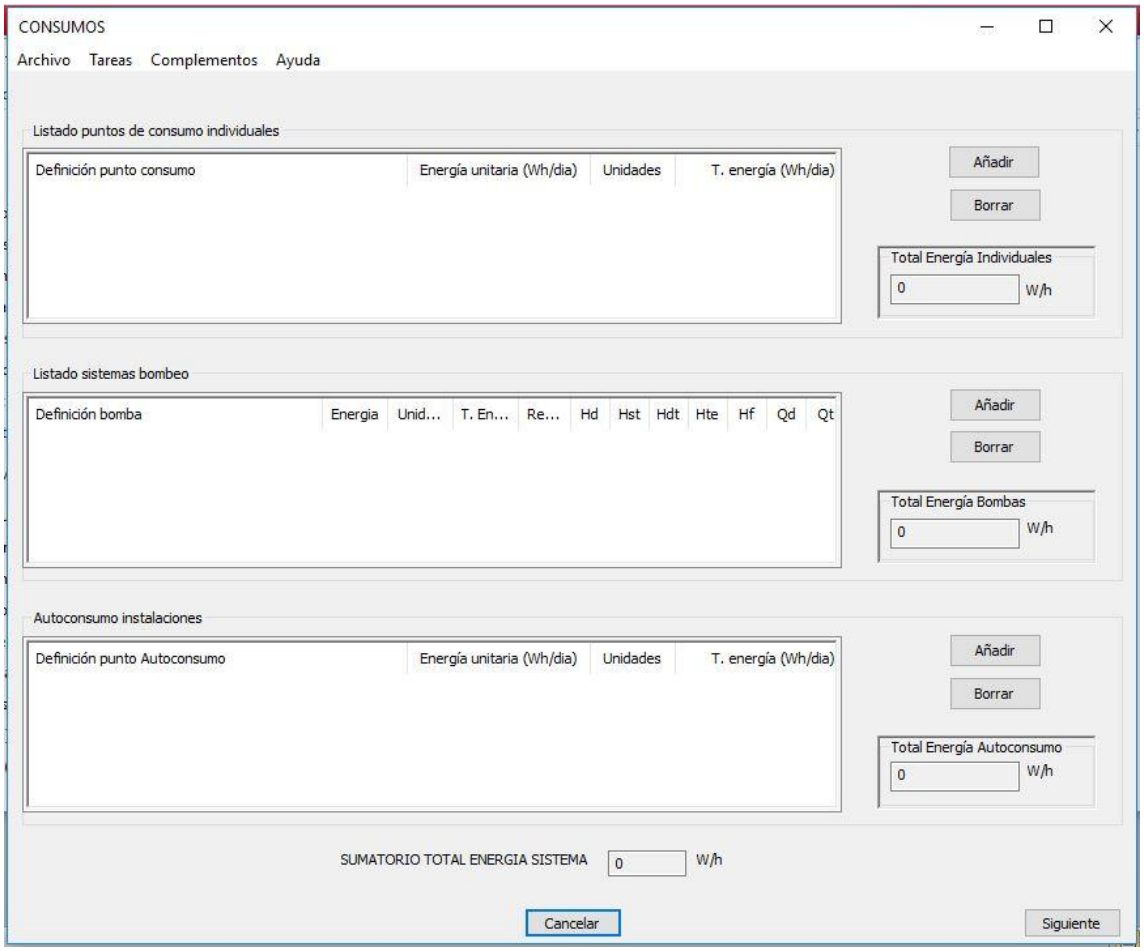

Figura 9: Cuadro dialogo consumos instalación conectada red

Los consumos energéticos se dividen en tres apartados, de los cuales en cada uno de ellos, la información necesaria quedará reflejada en el cuadro de dialogo correspondientes. Tales puntos de consumo son:

- *Puntos de consumo individuales*: Se insertará todo punto de consumo diferente de una bomba que esté integrado en la instalación o explotación agropecuaria (elevadores de cangilones, motores, autómatas, etc…).

  La introducción de datos se realizar mediante el botón *"Añadir"*, siendo que al pulsarlo se abrirá un cuadro de dialogo que requerirá que el usuario introduzca los datos característicos de cada elemento concreto (descripción de punto de consumo, unidades de tal punto, energía consumida por unidad de elemento, etc..).

  La información de cada elemento, quedará reflejada en su apartado, siendo que el usuario puede eliminar un punto concreto, seleccionándolo mediante el ratón de la lista, y pulsando posteriormente *"Borrar"*, eliminándose tal elemento de manera definitiva.

  La aplicación realizará el sumatorio de consumos totales de este tipo de apartado ("*Total consumos individuales*"), reflejando la energía total en casilla bajo botones de inserción y borrado.

- *Sistemas de bombeo*: Se insertaran todos los elementos de bombeo de la aplicación. La metodología de inserción y calculo será análoga al apartado anterior

- *Puntos de autoconsumo*: Se insertara todo punto propio de la instalación fotovoltaica que tengan autoconsumo (inversores, etc…). La metodología de inserción y cálculo será análoga al apartado anterior.

La siguiente pantalla guiara al usuario a través del cálculo de dimensionado de generador fotovoltaico, dividiéndose en tres apartados específicos:

- *Ubicación de la instalación:* Se permitirá introducir textualmente la ubicación de la instalación, así como generar los comentarios textuales que puedan ayudar al usuario a identificar la instalaciones, características de esta o cuantos datos considere de relevancia.

  Con respecto a latitud de la instalación, el usuario puede introducir de manera directa un valor numérico, o en su caso lanzar la aplicación GoogleMaps

("*Búsqueda latitud*") para búsqueda y localización de la latitud (que una vez localizada deberá ser introducida de manera manual).

- *Orientación, inclinación y perdidas*: El usuario introducirá el ángulo azimut de la instalación y la inclinación de los panales (inclinación total, debiéndose sumar en su caso inclinación de estructura de apoyo + cubierta). Así como la inclinación máxima y mínima en función del cálculo de pérdidas de orientación especificado en pliego técnico de condiciones IDAE (disponible en menú despegable Ayuda). De igual manera se fijará el porcentaje de pérdidas permitido según el tipo de instalación (seleccionable en menú desplegable).

  Las pérdidas por orientación se calcularán de manera automática, siendo que las pérdidas por sombreo, se podrán introducir de manera directa, o en su caso utilizar "*cálculo perdidas radiación por sombra*" para realizar de manera automatizada el cálculo (ver apartado perdidas por sombras).

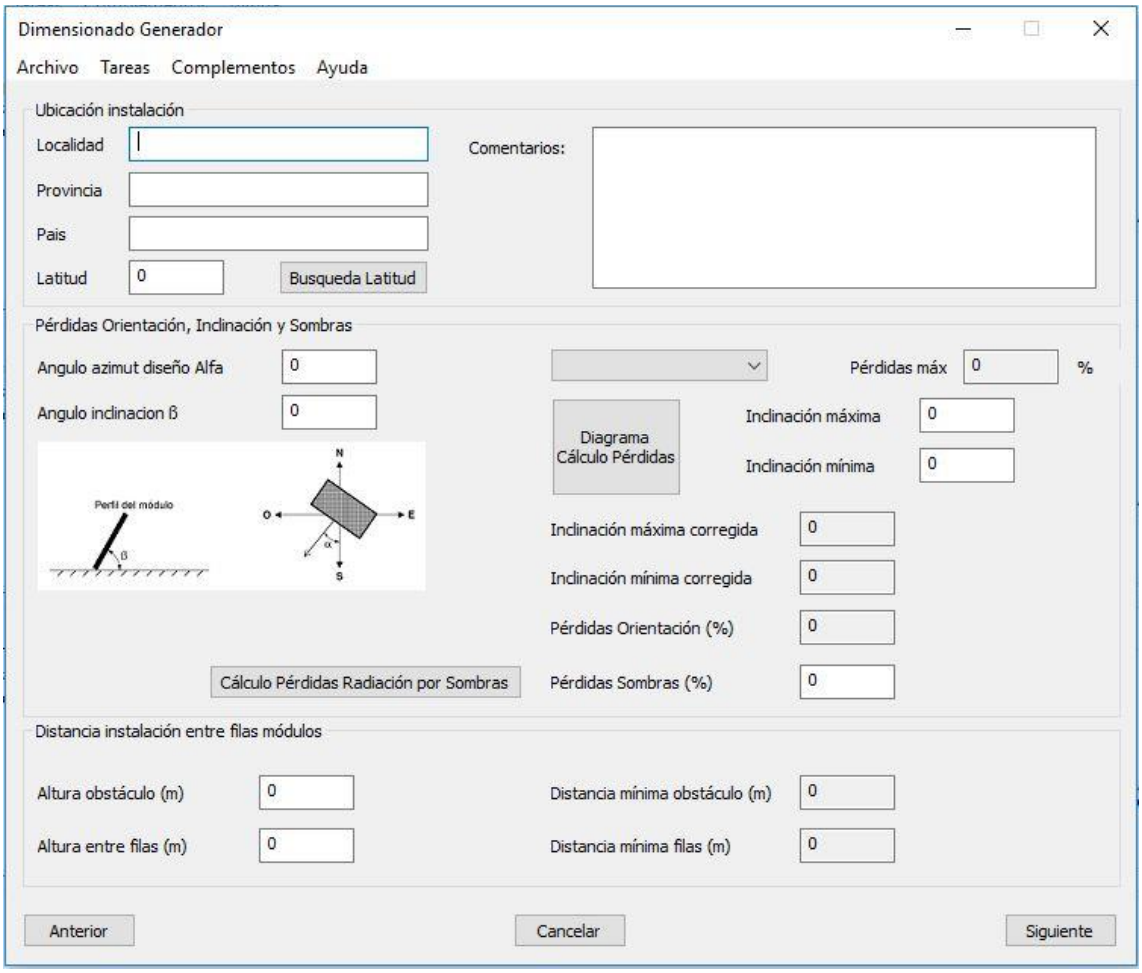

Figura 10: Cuadro dialogo dimensionado generador red

- *Distancias*: Fijando la altura de las líneas de paneles de la instalación con respecto a obstáculos cercanos, así como la altura entre fila (en caso de existir), se indicara al usuario las distancias mínimas para con los obstáculos y la separación entre filas de instalación.

Continuando con el cálculo de la instalación, se llamará a la pantalla correspondiente a cálculo de potencia instalada:

El usuario introducirá los datos del generador o paneles requeridos, en el que se describirán sus valores característicos de intensidades y tensiones de funcionamiento según catalogo (consultables en bases de datos del programa), así como la potencia de cada panel y el número de paneles que configurarán la instalación.

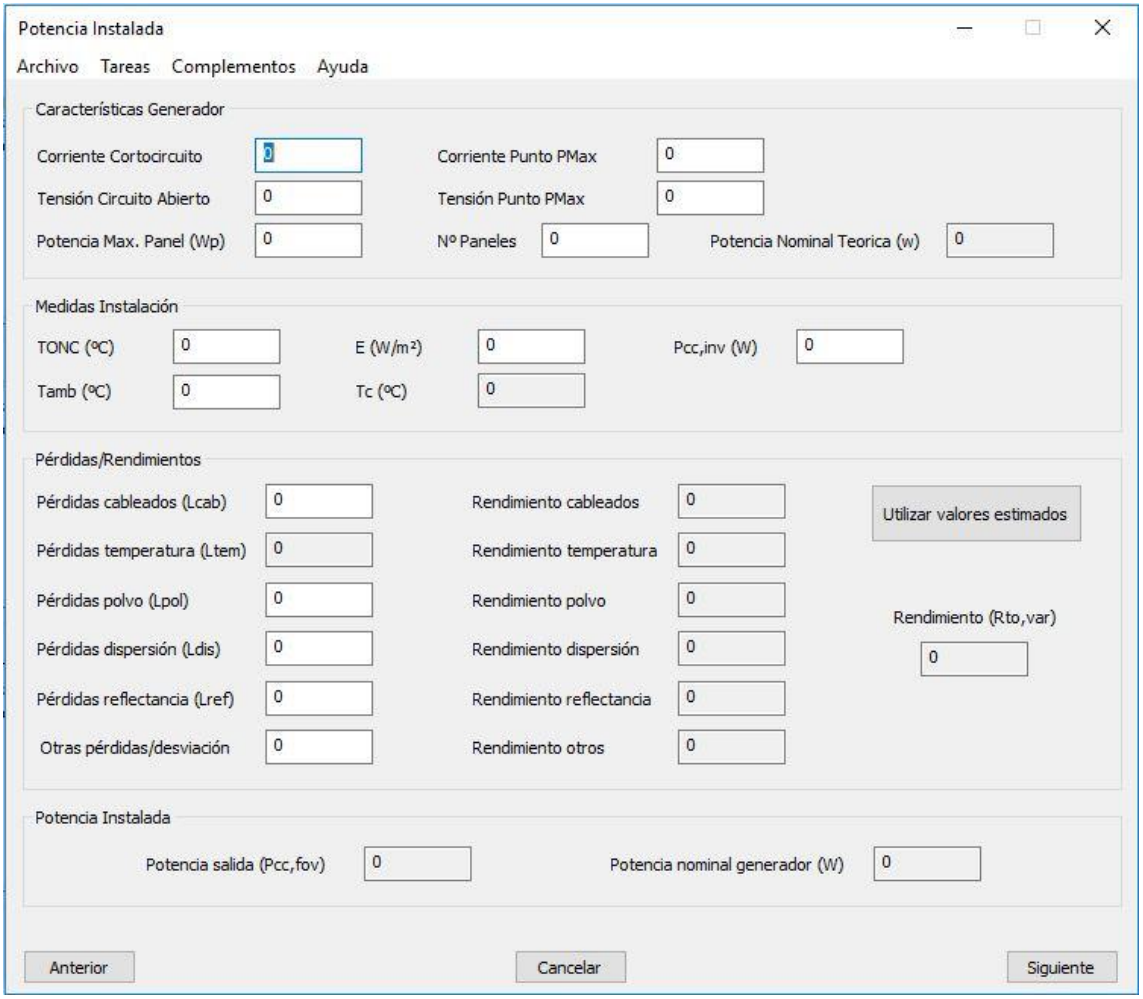

Figura 11: Cuadro dialogo potencia instalada

El sistema requerirá introducir en las casillas editables, datos de la instalación, o medidas de la misma, bien se conozcan in situ o sean obtenidas de catálogos o bases de datos. En caso de conocerse las pérdidas del instalación, por cálculo o medición, se

completarán las pérdidas de cada clase (en caso de desconocerse se pueden establecer perdidas estándar según "*utilizar valores estimados*" que cargará automáticamente pérdidas normales de instalaciones).

Mediante este cálculo el usuario obtendrá la potencia nominal del sistema.

Como último paso de la instalación o cálculo del generador se calculará la potencia anual esperada, requiriéndose la indicación de temperatura y radiación para cada mes del año.

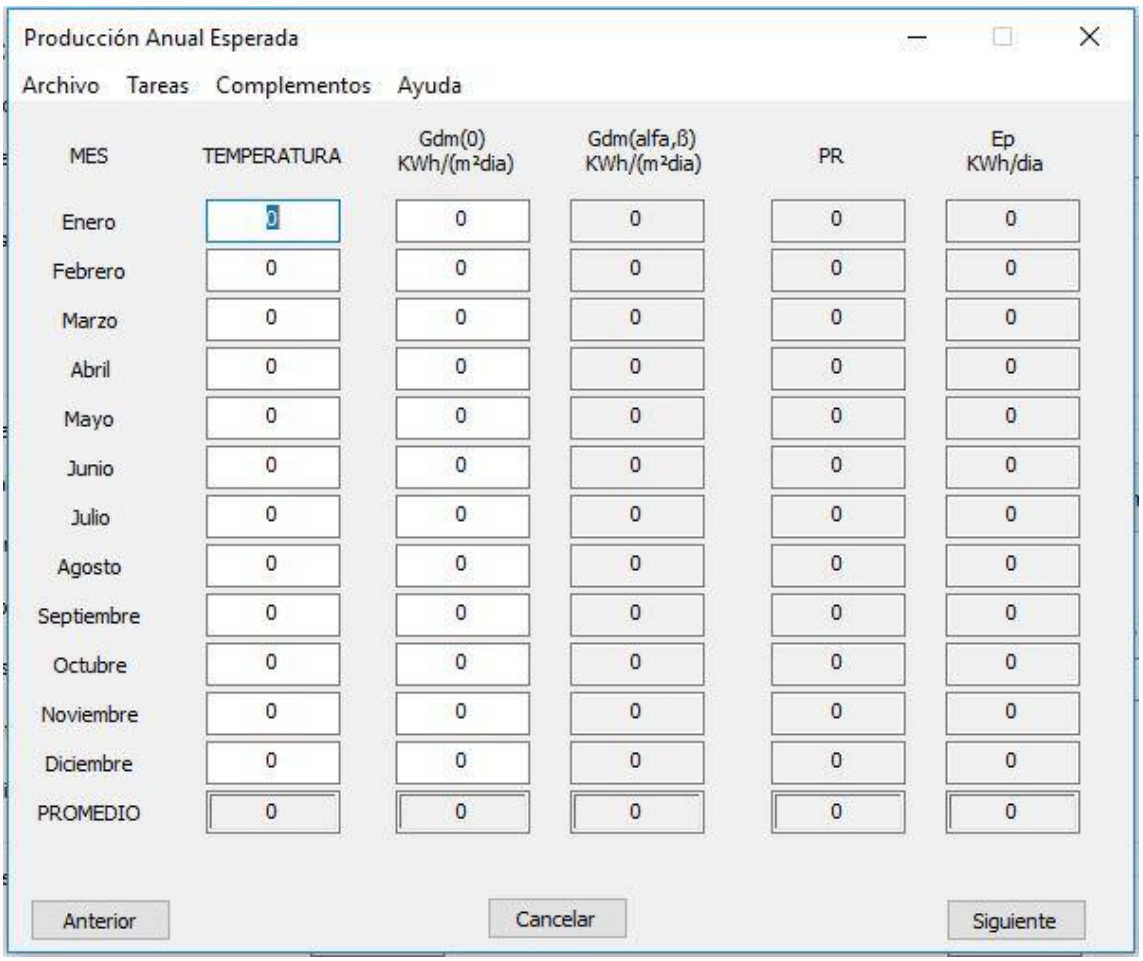

Figura 12: Cuadro dialogo producción esperada

La continuación requerirá del usuario seleccionar el tipo de estructura que portara los paneles, pudiendo ser esta de tipo seguidor (no implementando su cálculo en la presente versión del programa), estructura fija, sobre la que se podrá calcular la sobrecarga que produce en la cubierta en la que se instala, así como su respuesta a acciones externas como viento.

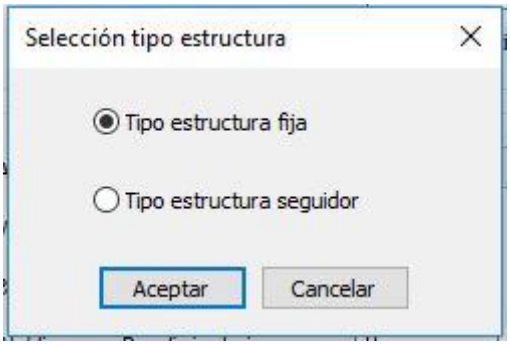

Figura 13: Cuadro dialogo tipo estructura

El cálculo de las cargas sobre cubierta, se realizarán en función de las fuerzas actuantes, resultando estas mayoritariamente el viento, dependiente de este de la zona geográfica (ver zona en diagrama "*ver zonas*"), la altura de la instalación, el grado de aspereza del entorno y el coeficiente eólico (datos a completar por el usuario), y aportando en función de tal información la acción del viento sobre la instalación de cubierta.

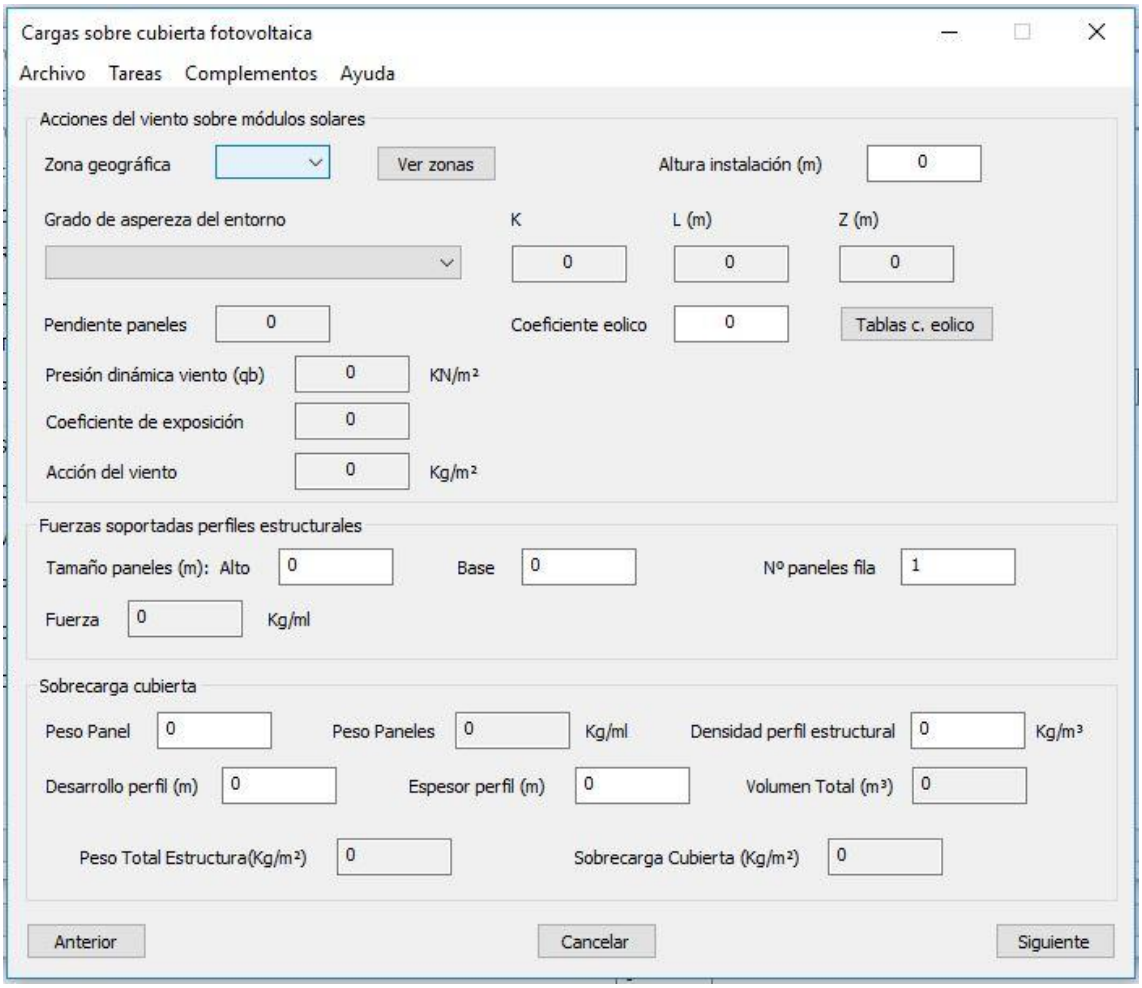

Figura 14: Cuadro dialogo cargas cubierta

Así mismo también se calcularán las fuerzas sobre los perfiles estructurales de los paneles, y la sobrecarga global de la cubierta, en la cual únicamente se indicara la carga extra derivada de la instalación de la cubierta solar, en ningún caso se realizarán cálculos estructurales de la propia cubierta o resistencia mecánica necesaria de ésta.

Finalmente en el proceso de cálculo, se analizará la inversión económica de la instalación, así como la rentabilidad de la instalación, en función delos costes de cada uno de los parámetros intervinientes y costes de los ismos.

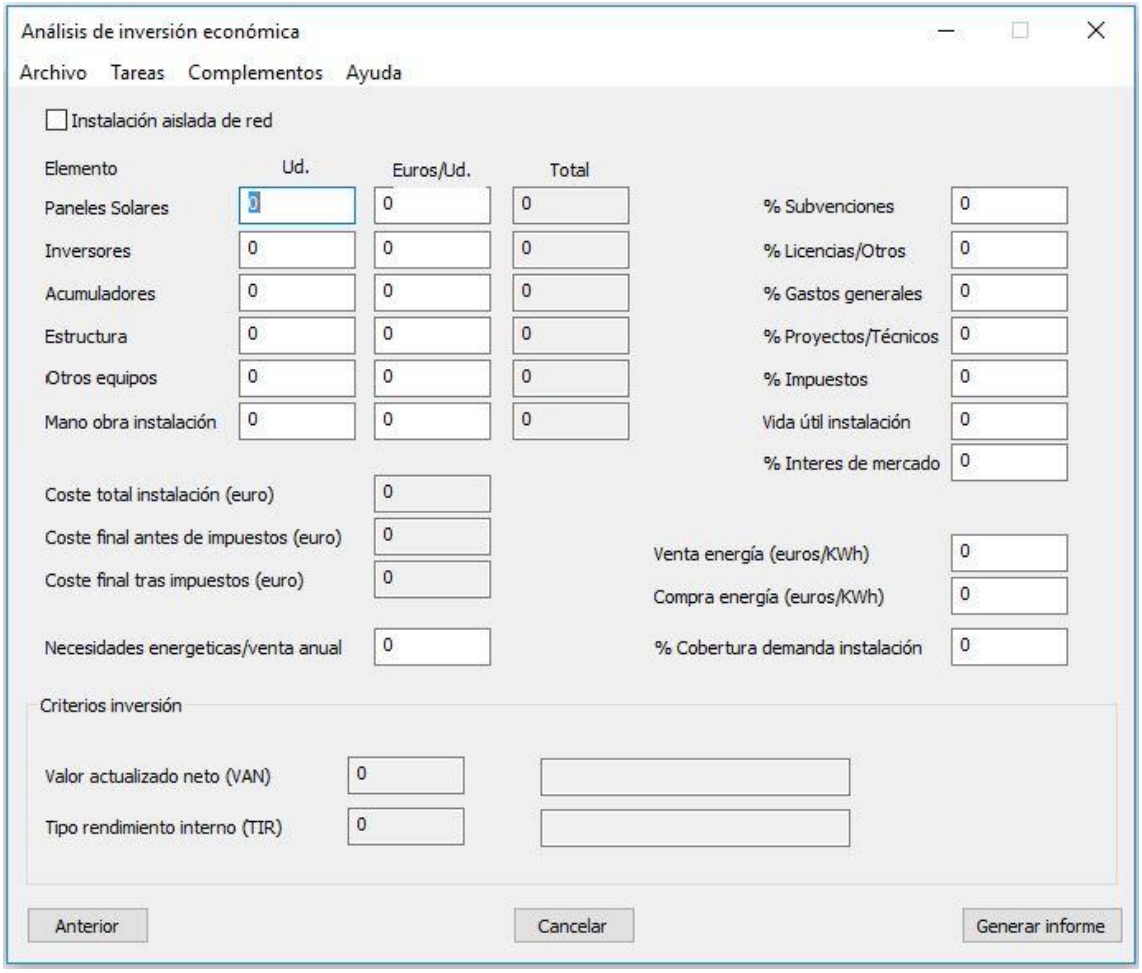

Figura 14: Cuadro dialogo análisis inversión

La aplicación permitirá calcular o selección si la instalación es aislada de red (marcando casilla usuario), en la cual el total de la energía es para autoconsumo, o si en su caso existe venta de energía.

- En el caso de ser una instalación conectada a red, se asume que la totalidad de energía es vendida, siendo por tanto que las necesidades energéticas/venta

anual (en KWh), se multiplicarán por el coste de venta a fin de analizar la rentabilidad de la inversión.

- En el caso de una instalación aislada de red, no existirá venta, pero si unas necesidades energéticas que deberán ser satisfechas, permitiendo el programa que se estipule un porcentaje de cobertura de autodemanda. Estos es, existirá parte de la demanda que requerirá compra de energía con un coste y parte de la cobertura de las necesidades que se cubrirán con costes producción de la instalación, lo cual conllevará una reducción del coste energético total de la empresa y por tanto se considerara a efectos de cálculo de rentabilidades.

El programa realizará el análisis de la inversión en función del criterio del VAN y del TIR, informando al usuario sobre si la inversión debería realizarse o no, atendiendo exclusivamente a tale parámetros y rentabilidad de la inversión.

La finalización del cálculo mediante "Generar informe", lanzara un documento "*txt*", que el usuario podrá modificar y/o guardar.

Base Datos:

Heclia dispone de la posibilidad de generar una base de datos consultables de diverso equipamiento o materiales de normal presencia en instalaciones de cálculo en energías renovables. El usuario podrá agregar y/o generar su propia base de datos, manipulándola mediante la introducción de datos, el borrado de los mismos y el guardado en archivo permanente (que funcionará o recuperará datos en cualquier equipo que mantenga *Heclia* instalado tan solo con el guardado del archivo en la carpeta de instalación).

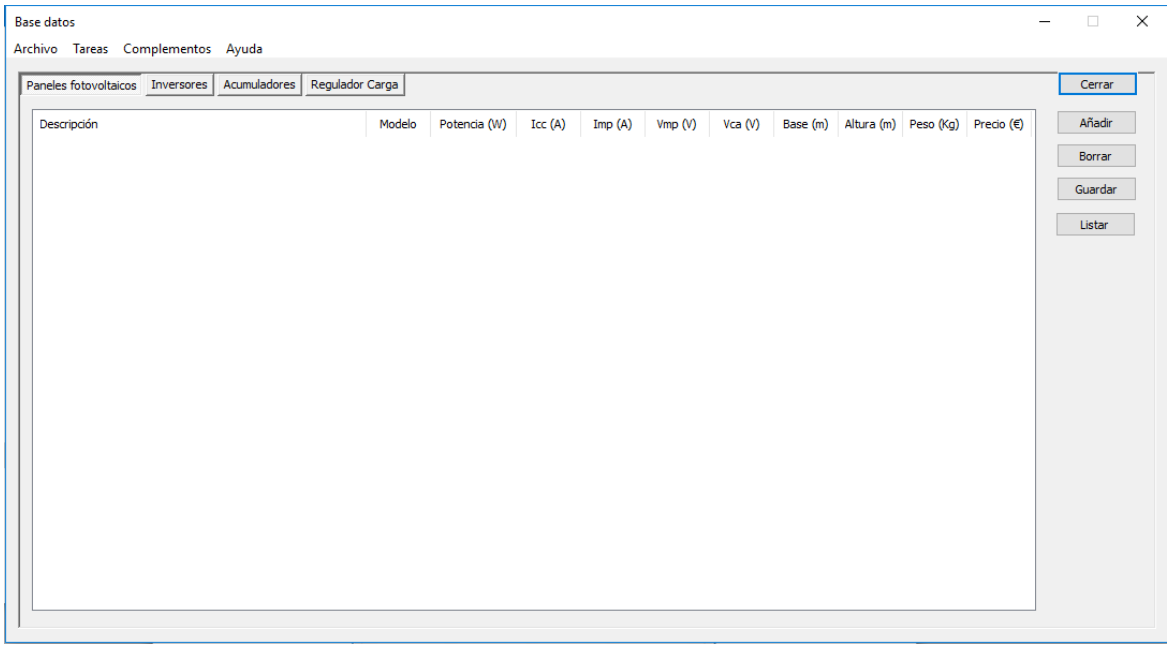

Figura 15: Cuadro dialogo base datos

Las bases de datos de la versión actual, corresponden a paneles solares fotovoltaicos, inversores, acumuladores y reguladores de carga, siendo que los datos representativos y técnicos de cada uno de los elementos, se mostrará en pantalla a modo de lista.

El usuario podrá cambiar entre cada tipo de elemento mediante los pulsadores dispuestos sobre las listas, estando activado aquel que se muestra marcado. En la versión actual de la aplicación, la base de datos es consultiva, no estando disponible la posibilidad de capturar información para que se cargue automáticamente en otras zonas o cálculos de la aplicación.

En cada base de datos las opciones disponibles serán:

- *Cerrar*: Se cierra automáticamente la totalidad de bases de datos, con independencia de la pantalla en la que se encuentre el usuario.

- *Añadir:* La opción añadir mostrará un cuadro de dialogo que solicitará al usuario los datos a introducir en la base de datos para ser almacenados, siendo que en función de la vista o pantalla de equipamientos en la que se encuentre el usuario, el cuadro de introducción de datos estará adaptado a la misma, siendo diferente para cada tipo de aparamenta. No se pueden introducir datos de un elemento, cuya base de datos no esté siendo visualizada en el momento de la introducción de la información.

- *Borrar*: La función borrar sirve para eliminar elementos de la base de datos. No se pueden borrar varios elementos al mismo tiempo, exclusivamente de uno en uno. Para borrar el usuario deberá marcar el elemento a manipular (se selecciona la descripción que varía de color al ser pulsada con el ratón), pulsándose a continuación el botón "*Borrar*", para eliminar tal elemento (el elemento borrado puede volver a ser introducido, pero no recuperado o desecha la acción de borrado).

- *Guardar*: El guardado almacena en soporte físico (carpeta de archivos de Heclia en disco duro), el contenido de la base de datos. Existe un soporte o archivo de guardado por cada una de las bases de datos, no siendo archivos editables o manipulables fuera de Heclia. Cuando se modifique una base de datos, será necesario el guardado para conservarla, siendo que la acción de guardado debe ser realizada antes de salir de la aplicación o cambiar de vista o clase de base de datos.

- *Listar*: Si el usuario lo desea, puede solicitar el listado de la base de datos en la que se encuentre, generando esto un fichero de salida *"*.txt"*, el cual pude ser guardado y/o editado como archivo de texto normal (las modificaciones del mismo luego no serán reconocidas por Heclia).

Calculo perdidas por sombras:

Sistema de cálculo de pérdidas por sombreo sobre superficie de producción solar fotovoltaica, siendo que se podrá acceder a tal modulo, directamente desde el interior de la aplicación seleccionada, o de manera directa desde el menú "*complementos*".

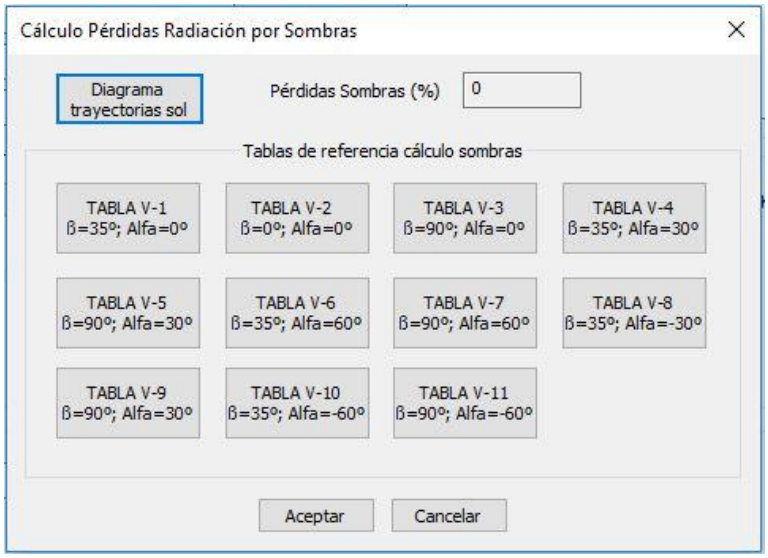

Figura 16: Cuadro dialogo cálculo pérdidas

Activando la función de cálculo de pérdidas de radiación solar por sombras, se podrá aplicar el método de cálculo de las pérdidas que experimenta una superficie debidas a las sombras circundantes. Tales perdidas se expresaran como porcentaje de la radiación solar global que incidiría sobre la mencionada superficie de no existir sombra alguna.

El usuario realizará una definición o cálculo del perfil de obstáculos, atendiendo al diagrama de trayectorias del sol, disponible pulsando sobre "*diagrama trayectorias de sol*", en el cual se representan las hora solares delimitadas e identificadas por una letra y un número, conociéndose el porcentaje de ocultación de los objetos sobre cada una de esas casillas en función de una ángulo de elevación y un azumit.

Una vez se conozca el perfil de sombras, se escogerá de entre las tablas disponibles, la más adecuada o parecida a la realidad de la instalación, en función de la inclinación y ángulo azimut, pulsando el botón correspondiente a tal tabla, que desplegará los datos de pérdida correspondientes a cada casilla del diagrama antes indicado, marcándose o pulsando el valor de cada elemento en función de la letra y número de referencia correspondiente como si se tratara de una tabla de doble entrada.

La pulsación de tal valor lo carga en el porcentaje de pérdidas, dejando la casilla marcada con una "*xxxx*", referencia que indica que se ha sumado tal valor, siendo que una segunda pulsación desmarcaría la casilla, retornado el valor "xxxx" a su valor numérico originario, lo cual indica que no se ha consideradora en el cálculo tal valor.

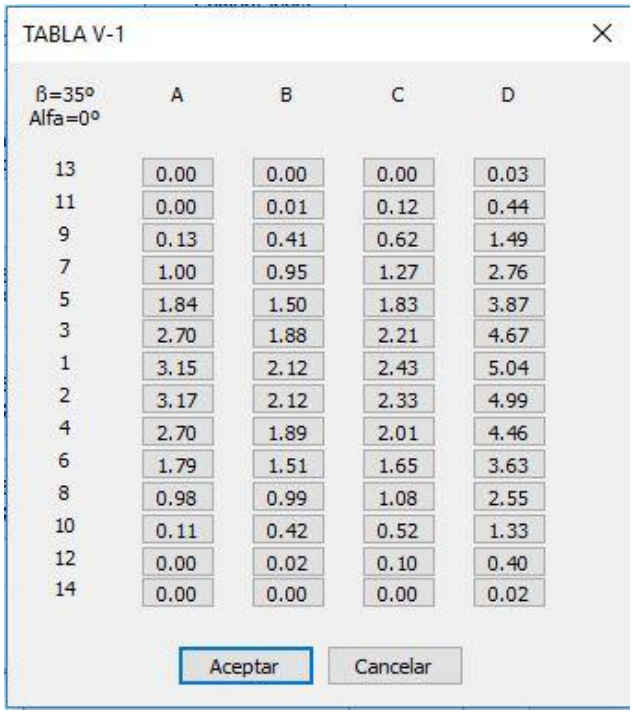

Figura 17: Cuadro dialogo valores pérdidas

Así mismo con respecto a cada valor, puede ser que su contribución o grado de sombra no sea del 100%, motivo por el cual al seleccionarse el valor o casilla que contribuye a las perdidas, el programa abrir un cuadro de dialogo que nos pregunte, que porcentaje de sombra afecta a la casilla o valor seleccionado, a fin de aplicar ese porcentaje sobre el valor total referenciado.

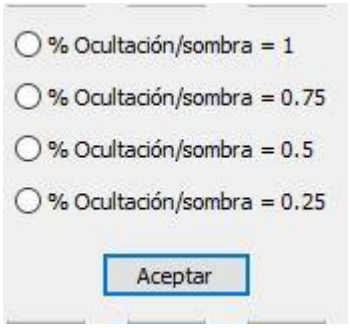

Figura 18: Cuadro dialogo selección porcentaje sombra

Menú:

El menú general disponible desde cada vista, permitirá al usuario acceder a diferentes opciones:

- *Archivo:* Presenta funcionalidades de "*Guardar*" en la que se abre un cuadro de dialogo en el que el usuario puede guardar el trabajo realizado, seleccionando la ubicación del archivo a guardar así como el nombre que desee y la opción "*Salir*", para cerrar la aplicación de manera directa con independencia de la zona de proceso de cálculo en la que se encuentre.

- *Tareas*: Función no implementada en la versión actual.

- *Complementos*: da acción a las funciones:

  - *GoogleMaps:* Abre la función Google Maps en navegador por defecto del equipo, para poder localizar latitudes de lugar ubicación instalación.
  - *Perdidas sistemas fotovoltaicos:* Permite acceder directamente a aplicación de cálculo de sombras incluida en *Heclia*, así como a los diagramas para realizar cálculos de pérdidas por sombreo y orientación correspondientes a pliegos técnicos de cálculo.
  - *Calculadora:* Lanza la calculadora integrada en Windows como apoyo ante posible necesidad de efectuar cálculos matemáticos.

- *Ayuda*: da acción a las funciones:

    - *Documentación:* da acceso en formato pfd a los archivos y pliegos técnicos de condiciones de cálculo de instalaciones en los que se basa la presente aplicación.
    - *Manual de usuario:* Permite acceder en formato pdf al presente manual de usuario.
    - *Versión:* Indicación versión actual de Heclia.

ANEXO II

Descripción código programación

INDICE

1.- Archivos código

En El presente anexo, se realizará una descripción pormenorizada del funcionamiento del código implementado en los archivos programados *".cpp*", recorriéndose los mismos explicando la funcionalidad de cada elemento interpuesto en ellos.

No se analiza o desglosa el funcionamiento de los archivos de cabecera correspondientes a los archivos principales que se explican, dado que los mismos contienen las clases base o derivadas, así como las declaraciones de variables, funciones, etc…, pero propiamente, no incluyen código añadido de actuación.

De igual manera, no se comentan los elementos concretos, herramientas, o funciones que se han definido o explicado en la memoria, o cuando menos no se analizan al detalle, dado que ya han sido ampliamente comentadas. De igual manera se procede con respecto a las funciones o ecuaciones, que son de aplicación y que se han definido en capítulo de base teórica.

Según esto, la base de este anexo se centra, exclusivamente, en la descripción del código puro implementado en el presente proyecto, el recorrido a través de los archivos, llamadas a funciones y en general, el código definido.

Los archivos serán explicados de manera pormenorizada, siguiendo el desarrollo de la aplicación, esto es, se comenzarán describiendo el código de la pantalla principal y secuencialmente el código que va surgiendo en el que sería el orden de ejecución normal del usuario conforme utiliza la aplicación. Esto significa recorrer el código que se ejecuta internamente, conforme se realiza un proyecto o actividad normal con el programa desarrollado.

2.- Pantalla principal Heclia

De manera inicial, tanto en el presente archivo, como en todos los demás que lanzan o hacen llamadas a un cuadro de dialogo, se realiza la gestión o inicialización estándar de este elemento, declarándose los archivos de cabecera a los que se hace referencia en el archivo o los cuales debe reconocer, participar o compartir.

De igual manera se realizará la carga de elementos estándar en la pantalla principal del proyecto, llamándose o cargándose elementos de ayuda, el icono principal de la aplicación y gestiones generalistas similares, tales como llamar al constructor, destructor de la clase, etc…

Tales tareas son comunes y generales para cada uno de los archivos que se describirán posteriormente, siendo por tanto que esta introducción se obviará de aquí en adelante, debiéndose asumir que la invocación al cuadro de dialogo y el resto de elementos usuales, se repetirán para cada clase.

PFCDlg.cpp: archivo de implementación

```
#include "stdafx.h"
#include "PFC.h"
#include "PFCDlg.h"
#include "DlgProxy.h"
#include "afxdialogex.h"
#include "Consumos.h"
#include "ConsumosRed.h"
#include "AbrirExistente.h"
#include "BaseDatosGeneral.h"
#include "PerdSombrasGraf.h"
#include "CalcPerdSombra.h"
#include "PredOrientacionGraf.h"
#include "DGeneradorRed.h"

using namespace std;
#ifdef _DEBUG
#define new DEBUG_NEW
#endif

int SeleccionInstalacion;
// Cuadro de diálogo CAboutDlg utilizado para el comando Acerca de

class CAboutDlg : public CDialogEx
{
public:
	CAboutDlg();

// Datos del cuadro de diálogo
	enum { IDD = IDD_ABOUTBOX };

	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // Compatibilidad con
DDX/DDV

// Implementación
protected:
	DECLARE_MESSAGE_MAP()
```

```
};

CAboutDlg::CAboutDlg() : CDialogEx(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialogEx)
END_MESSAGE_MAP()

// Cuadro de diálogo de CPFCDlg

IMPLEMENT_DYNAMIC(CPFCDlg, CDialogEx);

CPFCDlg::CPFCDlg(CWnd* pParent /*=NULL*/)
	: CDialogEx(CPFCDlg::IDD, pParent)
{
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
	m_pAutoProxy = NULL;
}//IDI_ICONHECLIA

CPFCDlg::~CPFCDlg()
{
	// Si existe un proxy de automatización para este cuadro de diálogo,
	//  volver a establecer su puntero a dicho cuadro de diálogo como NULL,
	para que sepa
	//  que el cuadro de diálogo se ha eliminado.
	if (m_pAutoProxy != NULL)
		m_pAutoProxy->m_pDialog = NULL;
}

void CPFCDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
}
```

Se declararán los mensajes que actúan o que son invocados en el archivo presente, siendo que cada una de estas llamadas, relacionan el identificador específico de cada herramienta o control de la plantilla o cuadro de dialogo, con la acción a ejecutar, siendo el código o declaración:

Evento a ejecutar (acción que se ejecuta o que desencadena el evento, véase `ON_BN_CLICKED`, para pulsaciones del botón izquierdo del ratón), identificación del botón, campo de texto, o recurso al que hace menciona la acción (`IDC_ABRIREXISTENTE`, identificador del elemento sobre el que se está actuando) y función a la que se hace llamada o se invoca en el evento lanzado (`CPFCDlg::OnBnClickedAbrirexistente`, se llama a la función concreta invocada, dándose un salto del programa a la misma).

```
BEGIN_MESSAGE_MAP(CPFCDlg, CDialogEx)
	ON_WM_SYSCOMMAND()
	ON_WM_CLOSE()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
```

```
	ON_BN_CLICKED(IDC_ABRIREXISTENTE, &CPFCDlg::OnBnClickedAbrirexistente)
	ON_COMMAND(ID_ARCHIVO_GUARDAR, &CPFCDlg::OnArchivoGuardar)
	ON_BN_CLICKED(IDC_RED, &CPFCDlg::OnBnClickedRed)
	ON_BN_CLICKED(IDC_MIXTA, &CPFCDlg::OnBnClickedMixta)
	ON_BN_CLICKED(IDC_AEROGENERACION3, &CPFCDlg::OnBnClickedAerogeneracion3)
	ON_BN_CLICKED(IDC_BASEDATOS, &CPFCDlg::OnBnClickedBasedatos)
	ON_BN_CLICKED(IDC_FOTOAISLADA, &CPFCDlg::OnBnClickedFotoaislada)
	ON_COMMAND(ID_CALCSOMBRAS, &CPFCDlg::OnCalcsombras)
	ON_COMMAND(ID_DIAGORIENTACIO, &CPFCDlg::OnDiagorientacio)
	ON_COMMAND(ID_DIAGSOMBRAS, &CPFCDlg::OnDiagsombras)
	ON_COMMAND(ID_GOOGLE, &CPFCDlg::OnGoogle)
	ON_COMMAND(ID_COMPLEMENTOS_CALCULADORA,&CPFCDlg::OnComplementosCalculador)
	ON_COMMAND(ID_APP_EXIT, &CPFCDlg::OnAppExit)
	ON_COMMAND(ID_AYUDA_VERSI32797, &CPFCDlg::OnAyudaVersi32797)
	ON_COMMAND(ID_DOCUMENTAISLADA, &CPFCDlg::OnDocumentaislada)
	ON_COMMAND(ID_DOCUMENTRED, &CPFCDlg::OnDocumentred)
	ON_COMMAND(ID_AYUDA_MANUALUSUARIO, &CPFCDlg::OnAyudaManualusuario)
END_MESSAGE_MAP()
```

Se describen secuencialmente todas las funciones a las que se hacen llamadas, o que como se ha indicado previamente, son invocadas por eventos, bien internos del proceso del programa, o por gestiones y llamadas del usuario durante su interacción para con el programa, en una primera sección se exponen las llamadas o funciones embebidas en el propio código y que gestiona la vista de recurso, posicionando la ventana, icono, interponiendo funciones básicas de minimización, botones de apertura/cierre, etc…

```
// Controladores de mensaje de CPFCDlg

BOOL CPFCDlg::OnInitDialog()
{
	CDialogEx::OnInitDialog();

	// Agregar el elemento de menú "Acerca de..." al menú del sistema.

	// IDM_ABOUTBOX debe estar en el intervalo de comandos del sistema.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		BOOL bNameValid;
		CString strAboutMenu;
		bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
		ASSERT(bNameValid);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}

	// Establecer el icono para este cuadro de diálogo.  El marco de trabajo
	realiza esta operación
	//  automáticamente cuando la ventana principal de la aplicación no es un
	cuadro de diálogo
	SetIcon(m_hIcon, TRUE);			// Establecer icono grande
```

```
	SetIcon(m_hIcon, FALSE);		// Establecer icono pequeño

	// TODO: agregar aquí inicialización adicional

	return TRUE;  // Devuelve TRUE  a menos que establezca el foco en un
	control
}

void CPFCDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialogEx::OnSysCommand(nID, lParam);
	}
}

// Si agrega un botón Minimizar al cuadro de diálogo, necesitará el siguiente
código
//  para dibujar el icono.  Para aplicaciones MFC que utilicen el modelo de
documentos y vistas,
//  esta operación la realiza automáticamente el marco de trabajo.

void CPFCDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // Contexto de dispositivo para dibujo
SendMessage(WM_ICONERASEBKGND,reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

		// Centrar icono en el rectángulo de cliente
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Dibujar el icono
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialogEx::OnPaint();
	}
}

// El sistema llama a esta función para obtener el cursor que se muestra mientras
el usuario arrastra
//  la ventana minimizada.
HCURSOR CPFCDlg::OnQueryDragIcon()
{
	return static_cast<HCURSOR>(m_hIcon);
}
```

```
// Los servidores de automatización no se deben cerrar cuando un usuario cierra
la interfaz de usuario
//  si un controlador conserva uno de sus objetos.
//  Estos controladores de mensaje garantizan que si el servidor proxy aún está
en uso,
//  la interfaz de usuario (UI) se oculta pero el cuadro de diálogo que permanece
a su alrededor
//  se descarta.

void CPFCDlg::OnClose()
{
	if (CanExit())
		CDialogEx::OnClose();
}

void CPFCDlg::OnOK()
{
	if (CanExit())
		CDialogEx::OnOK();
}

void CPFCDlg::OnCancel()
{
	if (CanExit())
		CDialogEx::OnCancel();
}

BOOL CPFCDlg::CanExit()
{
	// Si el objeto del servidor proxy aún está presente, el controlador de
	//  automatización conserva esta aplicación.  Dejar
	//  el cuadro de diálogo, pero ocultar su interfaz de usuario.
	if (m_pAutoProxy != NULL)
	{
		ShowWindow(SW_HIDE);
		return FALSE;
	}

	return TRUE;
}
```

Ante la pulsación del usuario del botón correspondiente a la selección de instalaciones fotovoltaicas aisladas de red, se hace llamada a la presente función, asignándose automáticamente un valor que identificara la selección realizada, y la asociará con los archivos que se vayan a guardar en el disco duro, de tal manera, que el programa reconozca en todo momento, que los datos introducidos pertenecen a este tipo de instalación, y no otra. Posteriormente se hace una llamada a un cuadro de dialogo modal, el cual corresponderá, en el presente caso, a la plantilla o cuadro de dialogo que gestionará el cálculo de instalaciones fotovoltaicas aisladas de red (ver aparatado correspondiente)

```
void CPFCDlg::OnBnClickedFotoaislada()
	// TODO: Agregue aquí su código de controlador de notificación de control
{
	IntalAisla = true;
	InstalRed = false;
```

```
	SeleccionInstalacion = 1; //valor de selección para primera posición de
seriación

	Consumos DlgConsumos(NULL);	//Llama al constructor del dialogo

	if (DlgConsumos.DoModal() == IDOK)

	{

	}
}
```

El usuario puede seleccionar la apertura de un archivo de programa almacenado en disco duro, esto es un proyecto existente y que previamente ha manipulado o creado. El guardado de un archivo, permite conservar de manera permanente un proyecto o los datos del mismo, recuperándose en el momento que el usuario quiera modificar o consultar el archivo. La presente función hace una llamada al cuadro de diálogo "Abrir", en el que se ofrecerá una vista estándar de este tipo de cuadro formato Windows, existiendo una vista de las diferentes carpetas del disco duro del equipo, y pudiéndose seleccionar de las mismas el archivo a abrir en función de las terminaciones que reconoce el sistema implementado.

```
void CPFCDlg::OnBnClickedAbrirexistente()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	this->UpdateData();
		CFileDialog DlgAbrirExistente(TRUE, _T(".sfa"), NULL,
	OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,_TEXT("Ficheros de proyecto
	(*.sfa)|*.sfa |Ficheros de texto (*.txt)|*.txt| Todos los ficheros
	(*.*)|*.*||"), NULL);

	if (DlgAbrirExistente.DoModal()==IDOK)

	{
```

De manera inicial se recuperará el nombre del archivo seleccionado por el usuario, abriéndose en modo lectura y preparándose para el cargado de los datos serializados en el mismo.

```
		//Visualizar el nombre del fichero en la caja de texto
		Registro = DlgAbrirExistente.GetFileName();
		UpdateData(true);
		RegistroAux = Registro;

		//abrir el fichero para leer
		CFile FicheroAbierto;
		FicheroAbierto.Open(Registro, CFile::modeRead);//abro fichero
```

La deserialización del archivo y recuperación de sus datos, comenzará recuperando el primer valor de la serialización, que siempre corresponderá, con el tipo de instalación con la que el usuario ha ejecutado el proyecto y que por tanto reconocerá el modelo y parámetros de la misma

```
            CArchive ar(&FicheroAbierto, CArchive::load);//abro fichero
            serializable

                  ar >> SeleccionInstalacion;

                  ar.Close();
                  FicheroAbierto.Close();//cierro ficheros
                  this->UpdateData(FALSE);
```

Una vez recuperado el tipo de información o instalación que se desea abrir, se saltará a la función interna de deserialización, ejecutándose en la misma todo el proceso de extracción de datos (función expuesta en el presente archivo de código en secciones posteriores).

```
                  ExtraerDatos();//salto a función que distribuye los datos en
                  función del tipo de instalación que había generado
                  AbiertoAachivo = true;
      }
}
```

A la inversa que el proceso de guardado, el usuario puede seleccionar del menú de usuario la función de guardado, siendo esta inversa a la apertura, de manera que la primera extraía los datos de una archivo de disco, mientras que la presente, introduce o serializa la información en el archivo especifico de la aplicación. A tal respecto, se hará una llamada a la caja de dialogo estándar de "*Guardar como*", la cual será invocada de manera análoga a la de apertura de archivos, pasando los mismos parámetros y requerimientos del archivo a manipular, siendo la única diferencia el primer argumento pasado a la clase del cuadro, que en el caso de apertura era TRUE y en el de guardado es FALSE.

```
void CPFCDlg::OnArchivoGuardar()
{
      // TODO: Agregue aquí su código de controlador de notificación de control
      this->UpdateData();
      CFileDialog DlgGuardarCambios(FALSE, NULL,_T(".sfa") , OFN_HIDEREADONLY |
      OFN_OVERWRITEPROMPT ,_TEXT("Ficheros de proyecto (*.sfa)|*.sfa |Ficheros
      de texto (*.txt)|*.txt| Todos los ficheros (*.*)|*.*||"), NULL);

      //visualiza caja de dialogo Guardar
      if (DlgGuardarCambios.DoModal() == IDOK)

      {
```

Como en el caso de la aperura de archivos, el código consistirá en la capura del nombre del archivo seleccionado en el cuadro de dialogo o en su caso impuesto por el usuario.

```
            //Visualizar el nombre del fichero en la caja de texto
            m_TextoEdit2 = DlgGuardarCambios.GetFileName();
            UpdateData(true);
```

Con el nombre del archivo, se realizará un salto a la función "*ExtraerTemporales()*", en la cual se procesará el nombre para guardar de forma paralela dos archivos, el de puntos de consumo, y el de datos generales del resto de la aplicación, guardándose con similar nombre pero uno con terminación .sfa y el otro .dat.

```
ExtraerTemporales(); //extrae datos de archivos temporales para
seriar en archivo asociado
```

El proceso de gestión del guardado, asociado a su cuadro de dialogo especifico, será análogo al de apertura, abriendo un modo de escritura, creando el archivo para almacenaje y guardando secuencialmente en el mismo, la totalidad de datos generados por la aplicación.

```
//abrir el fichero para escribir
CFile FicheroGuardado;
//estructura para almacenar el estado del fichero
CFileStatus EstadoFi;
UINT nModoDeAceso = CFile::modeWrite;  //Acceso para escribir
//GetStatus devuelve true si el fichero existe, o false si no
existe

//guardar datos generales aplicación
FicheroGuardado.Open(m_TextoEdit2, CFile::modeCreate
CFile::modeWrite);
CArchive ar(&FicheroGuardado, CArchive::store);

if (InstalRed)
{
//sección DisSistemaRed
ar<< SeleccionInstalacion << LocalidadInfosalida<<
ProvinciaInfosalida << PaisInfosalida << LatitudInfosalida <<
ComentariosInfosalida <<AlfaDisenoInfosalida <<
BetaDisenoInfosalida <<TipoPerdInfosalida << InclиMaxInfosalida <<
InclMinInfosalida << IncMaxCorrInfosalida <<IncMinCorrInfosalida <<
PerdidasOrientacionInfosalida <<PerPerdidasInfosalida <<
PerdSombrasInfosalidaInfosalida << HObsataculoInfosalida <<
HFilasInfosalida <<DObstaculoInfosalida << DFilasInfosalida <<
KInfosalida;
//sección DisGeneradorRed
ar<<ICortoInfosalida << IPMaxInfosalida << VAbiertoInfosalida <<
VPMaxInfosalida <<PPanelInfosalida << NPanelesInfosalida <<
PTeoricaInfosalida << ToncInfosalida << EInfosalida <<
PInvInfosalida << TAmbInfosalida << TcInfosalida << LCabInfosalida
<< LTemInfosalida << LPolInfosalida << LDisInfosalida <<
LRefInfosalida << OtrosInfosalida << RCabInfosalida <<
RTemInfosalida << RPolInfosalida <<RDisInfosalida << RRefInfosalida
<< ROtrosInfosalida << RTotalInfosalida << PFovInfosalida <<
PNominalInfosalida;
//sección PEsperadaRed
ar<<TEneroInfosalida << TFebreroInfosalida << TMarzoInfosalida <<
TAbrilInfosalida << TMayoInfosalida << TJunioInfosalida <<
TJulioInfosalida << TAgostoInfosalida <<TSeptiembreInfosalida <<
TOctubreInfosalida << TNoviembreInfosalida <<TDiciembreInfosalida
<< TPromedioInfosalida <<G0EneroInfosalida << G0FebreroInfosalida
<< G0MarzoInfosalida << G0AbrilInfosalida <<G0MayoInfosalida <<
G0JunioInfosalida <<G0JulioInfosalida << G0AgostoInfosalida
<<G0SeptiembreInfosalida << G0OctubreInfosalida <<
G0NoviembreInfosalida <<G0DiciembreInfosalida<<G0PromedioInfosalida
<<GEneroInfosalida << GFebreroInfosalida << GMarzoInfosalida <<
GAbrilInfosalida <<GMayoInfosalida << GJunioInfosalida <<
GJulioInfosalida << GAgostoInfosalida <<GSeptiembreInfosalida <<
GOctubreInfosalida << GNoviembreInfosalida <<GDiciembreInfosalida
<< GPromedioInfosalida <<PREneroInfosalida << PRFebreroInfosalida
```

```
<< PRMarzoInfosalida << PRAbrilInfosalida <<PRMayoInfosalida <<
PRJunioInfosalida << PRJulioInfosalida << PRAgostoInfosalida
<<PRSeptiembreInfosalida << PROctubreInfosalida
<<PRNoviembreInfosalida <<PRDiciembreInfosalida <<
PRPromedioInfosalida <<EEneroInfosalida << EFebreroInfosalida <<
EMarzoInfosalida << EAbrilInfosalida <<EMayoInfosalida <<
EJunioInfosalida << EJulioInfosalida << EAgostoInfosalida
<<ESeptiembreInfosalida <<EOctubreInfosalida <<
ENoviembreInfosalida <<EDiciembreInfosalida << EPromedioInfosalida;
//sección cargas cubierta
ar<<ZonaInfosalida << AsperezaInfosalida <<PresDinamicaInfosalida<<
AlturaInstalInfosalida << KInfosalida << LInfosalida <<
ZInfosalida << PendientePInfosalida << CoefEolicoInfosalida <<
PresDinInfosalida << CExpInfosalida << AccVientoInfosalida
<< LargoInfosalida << AnchoInfosalida << PFilaInfosalida <<
FuerzaInfosalida << PesoPanelInfosalida << PesoPanelesMlInfosalida
<< DensidadInfosalida << DesarrolloPerfilInfosalida <<
EspesorPerfilInfosalida <<VolumenTotalInfosalida <<
PesoTotalInfosalida << SobrecargaInfosalida;
//sección inversion
ar<< UdPanelesInfoSalida<< UdInversoresInfoSalida<<
UdAcumuladoresInfoSalida<< UdEstructuraInfoSalida<<
UdOtrosInfoSalida<< UdManoInfoSalida<<EurosPanelesInfoSalida<<
EurosInversoresInfoSalida<< EurosAcumuladoresInfoSalida<<
EurosEstrcuturaInfoSalida<< EurosOtrosInfoSalida<<
 EurosManoInfoSalida<<TotalPanelesInfoSalida<<
TotalInversoresInfoSalida<< TotalAcumuladoresInfoSalida<<
TotalEstructuraInfoSalida<<TotalOtrosInfoSalida
<<TotalManoInfoSalida<< CosteInstalInfoSalida<<
SubvencionesInfoSalida<< LicenciasInfoSalida<<
GGeneralesInfoSalida<<ProyectosInfoSalida<<
CosteAImpuestosInfoSalida<< ImpuestosInfoSalida<<
CosteTrasImpInfoSalida<< NecesidadesInfoSalida<<
VidaUtilInfoSalida<< InteresMercadoInfoSalida<<
VentaEnergInfoSalida<< CompraEnergInfoSalida<<
CoberturaDemandaInfoSalida<<VANInfoSalida<< TIRInfoSalida<<
OkVanInfoSalida<< OkTirInfoSalida;
}

if (IntalAisla)
{
//Seccion DisSistema
ar << SeleccionInstalacion << LocalidadInfosalida <<
ProvinciaInfosalida << PaisInfosalida << ComentariosInfosalida <<
LatitudInfosalida << AlfaOptInfosalida << BetaOptInfosalida <<
KInfosalida << SelecPeriodoInfosalida <<AlfaDisenoInfosalida <<
BetaDisenoInfosalida << Gdm0Infosalida << FIrradiacionInfosalida <<
FSombreoInfosalida <<GdmMaxInfosalida << PerdidasRadInfosalida <<
PerdidasSomInfosalida << GdmDisInfosalida;
//Seccion DGenerador
ar << PRInfosalida << RendEnergInfosalida << PmpInfosalida <<
ICortoInfosalida << IPMaxInfosalida <<
VAbiertoInfosalida << VPMaxInfosalida << PotMaxPanelInfosalida <<
NPanelesInfosalida << PNominalInfosalida<< PMaxInfosalida <<
GdmDisAuxInfosalida << C20Infosalida << VnominalSisInfosalida <<
ProfDescargaInfosalida << ConsDiarioInfosalida << IscInfosalida <<
TipoInvInfosalida << Rend20Infosalida << RendNomInfosalida <<
RendInversorInfosalida << OpcionNormaInfosalida <<
RendAcumInfosalida << AutonomiaInfosalida <<
SumTotalInfosalida << GdmDisInfosalida;
```

```
		//sección cargas cubierta
		ar << ZonaInfosalida << AsperezaInfosalida <<
		PresDinamicaInfosalida << AlturaInstalInfosalida << KInfosalida <<
		LInfosalida <<ZInfosalida << PendientePInfosalida <<
		CoefEolicoInfosalida << PresDinInfosalida << CExpInfosalida <<
		AccVientoInfosalida<< LargoInfosalida << AnchoInfosalida <<
		PFilaInfosalida << FuerzaInfosalida << PesoPanelInfosalida <<
		PesoPanelesMlInfosalida << DensidadInfosalida
		<<DesarrolloPerfilInfosalida << EspesorPerfilInfosalida <<
		VolumenTotalInfosalida << PesoTotalInfosalida <<
		SobrecargaInfosalida;
		//sección inversion
		ar << UdPanelesInfoSalida << UdInversoresInfoSalida <<
		UdAcumuladoresInfoSalida << UdEstructuraInfoSalida <<
		UdOtrosInfoSalida << UdManoInfoSalida <<
		EurosPanelesInfoSalida << EurosInversoresInfoSalida <<
		EurosAcumuladoresInfoSalida << EurosEstrcuturaInfoSalida <<
		EurosOtrosInfoSalida <<EurosManoInfoSalida <<
		TotalPanelesInfoSalida << TotalInversoresInfoSalida <<
		TotalAcumuladoresInfoSalida <<TotalEstructuraInfoSalida <<
		TotalOtrosInfoSalida << TotalManoInfoSalida <<
		CosteInstalInfoSalida << SubvencionesInfoSalida <<
		LicenciasInfoSalida << GGeneralesInfoSalida << ProyectosInfoSalida
		<< CosteAImpuestosInfoSalida << ImpuestosInfoSalida <<
		CosteTrasImpInfoSalida <<NecesidadesInfoSalida <<
		VidaUtilInfoSalida << InteresMercadoInfoSalida <<
		VentaEnergInfoSalida << CompraEnergInfoSalida <<
		CoberturaDemandaInfoSalida <<VANInfoSalida << TIRInfoSalida <<
		OkVanInfoSalida << OkTirInfoSalida;

		}
		ar.Close();

		//escribir el texto en el fichero
		FicheroGuardado.Close();

	}
	}
```

La presente función, responde al evento del usuario formado por pulsación del botón correspondiente a instalaciones fotovoltaicas conectadas a red, mediante el cual se realiza una llamada al cuadro de diálogo modal que lanza la rama de la aplicación correspondiente a esta terminación.

```
void CPFCDlg::OnBnClickedRed()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	IntalAisla = false;
	InstalRed = true;
	SeleccionInstalacion = 2; //valor de selección para primera posición de
seriación

	ConsumosRed DlgConsumosRed(NULL);    //Llama al constructor del dialogo

	if (DlgConsumosRed.DoModal() == IDOK)

	{
	}
}
```

En la presente aplicación, no se han activado o implementado las funciones correspondientes a instalaciones eólicas, o mixtas solares/eólicas, si bien se instalan sus botones de lanzamiento de pantalla de usuario, resultando que en caso de actuarse sobre ellos, se hace una llamada a una caja de dialogo de texto, en el que se ofrece información al operador, en ambos casos indicando, textualmente que la herramienta esta desactivada.

Las cajas de texto estándar, se componen de tres elementos o entradas, correspondiendo la primera a la información que será reflejada en el cuadro, el segundo al texto de título de la ventana y la tercera al tipo de icono o imagen que será mostrada, bien sea de advertencia, información, prohibición, etc…

```
void CPFCDlg::OnBnClickedMixta()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	MessageBox(_TEXT("Modulo no instalado"), _TEXT("Calculo instalaciones
	mixtas"), MB_ICONINFORMATION);
}


void CPFCDlg::OnBnClickedAerogeneracion3()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	MessageBox(_TEXT("Modulo no instalado"), _TEXT("Calculo instalaciones
	aerogeneradores"), MB_ICONINFORMATION);
}
```

La actuación del botón de base de datos, mediante la pulsación del mismo, lanzará tal ramal de la aplicación, abriendo las cajas de dialogo y controles correspondientes a la base de datos desarrollada para poder almacenar información con referencia a materiales y diversos elementos de uso normal en instalaciones que serán generadas por la aplicación.

```
void CPFCDlg::OnBnClickedBasedatos()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	BaseDatosGeneral DlgBaseDatosGeneral(NULL);	//Llama al constructor del
dialogo

	if (DlgBaseDatosGeneral.DoModal() == IDOK)

	{
	}
}
```

Existirán otro tipo de elementos que serán lanzados, bien desde el menú, o en su caso desde los botones o herramientas dispuestas al uso, de forma que cuando el usuario actúe sobre ellos, al igual que en el caso anterior, se lanzaran cuadros de dialogo modales, como puede ser el caso del cálculo de pérdidas por sombreo, invocando a su función correspondiente.

```
void CPFCDlg::OnCalcsombras()
{
```

```
	// TODO: Agregue aquí su código de controlador de comandos
	CalcPerdSombra DlgCalcPerdSombra(NULL); //Llama aplicación calculo
perdidas sombras

	if (DlgCalcPerdSombra.DoModal() == IDOK)
	{

	}
}

void CPFCDlg::OnDiagorientacio()
{
	// TODO: Agregue aquí su código de controlador de comandos
	PredOrientacionGraf DlgPredOrientacionGraf(NULL); ////Diagrama orientación

	if (DlgPredOrientacionGraf.DoModal() == IDOK)
	{

	}
}

void CPFCDlg::OnDiagsombras()
{
	// TODO: Agregue aquí su código de controlador de comandos
	PerdSombrasGraf DlgPerdSombrasGraf(NULL); //Diagrama sombras

	if (DlgPerdSombrasGraf.DoModal() == IDOK)
	{
	}
}
```

El programa permite implementar llamadas a otros programas desde la propia base del desarrollado, pudiéndose invocar navegadores, elementos de apoyo de sistema (calculadora), o cualquier otro elemento que se ha considerado de interés y apoyo al desarrollo de la aplicación.

```
void CPFCDlg::OnGoogle()
{
	// TODO: Agregue aquí su código de controlador de comandos
	ShellExecute(NULL, _T("open"), _T("https://www.google.es/maps/"), NULL,
NULL, SW_SHOWNORMAL);
}

void CPFCDlg::OnComplementosCalculadora()
{
	// TODO: Agregue aquí su código de controlador de comandos
	system("calc.exe");

	return;
}
```

La función de salida, normal en toda aplicación, lanzará una advertencia en forma de caja de texto o mensaje (para avisar el usuario de la posible pérdida de datos), siendo que la cancelación del cierre, dejara sin efecto la función saliendo de ella, mientras que la aceptación de la acción, conllevara la invocación de la función *DestrroyWindow*, que cerrará toda ventana de la aplicación.

```
void CPFCDlg::OnAppExit()
{
	// TODO: Agregue aquí su código de controlador de comandos
	int mens = MessageBox(_TEXT("Va salir de la aplicación, si no ha guardado
los cambios los perdera\n¿desea salir?"),
		_TEXT("Salir de la Aplicación"), MB_ICONQUESTION|MB_OKCANCEL);

	switch (mens)
	{
	case IDCANCEL:
		// TODO: add code
		break;
	case IDOK:
		// TODO: add code
		DestroyWindow();
		SALIDA = true;
	//	DestroyWindow(); //cierra toda la aplicación
		break;

	}

}
```

La presente función viene invocada desde la sección o función `OnBnClickedAbrirexistente()`, estando encaminada exclusivamente a la extracción de la información contenida en un archivo permanente del disco duro. En función del tipo de instalación de la que se trate (fotovoltaica conectada a red o aislada), se extraerán y cargarán en variables globales (legibles y manipulables en cualquier sección del programa), correspondientes a cada tipo de instalación, a fin de que éstas queden reflejadas en cada zona necesaria del programa, cuando el usuario navegue por la interface gráfica de la aplicación.

```
void CPFCDlg::ExtraerDatos()
{
	if (SeleccionInstalacion == 1)   //tipo instalacion aislada de red
	{
		CFile FicheroAbierto;
		FicheroAbierto.Open(Registro, CFile::modeRead);
		CArchive ar(&FicheroAbierto, CArchive::load);

		ar >> SeleccionInstalacion >> LocalidadRetorno >> ProvinciaRetorno
		>> PaisRetorno >> ComentariosRetorno >>LatitudRetorno >>
		AlfaOptRetorno >> BetaOptRetorno >> KRetorno >> SelecPeriodoRetorno
		>>AlfaDisenoRetorno >> BetaDisenoRetorno >> Gdm0Retorno >>
		FIrradiacionRetorno >> FSombreoRetorno >>GdmMaxRetorno >>
		PerdidasRadRetorno >> PerdidasSomRetorno >> GdmDisRetorno;

		ar >> PRRetorno >> RendEnergRetorno >> PmpRetorno >> ICortoRetorno
		>> IPMaxRetorno >> VAbiertoRetorno >> VPMaxRetorno >>
		PotMaxPanelRetorno >> NPanelesRetorno >> PNominalRetorno >>
		PMaxRetorno >>GdmDisAuxRetorno >> C20Retorno >> VnominalSisRetorno
		>> ProfDescargaRetorno >> ConsDiarioRetorno >> IscRetorno
		>>TipoInvRetorno >> Rend20Retorno >> RendNomRetorno >>
		RendInversorRetorno >> OpcionNormaRetorno >> RendAcumRetorno >>
		AutonomiaRetorno >> SumTotalRetorno >> GdmDisRetorno;
```

```
            ar>>ZonaRetorno>> AsperezaRetorno>> PresDinamicaRetorno>>
            AlturaInstalRetorno>> KRetorno>> LRetorno>>ZRetorno>>
            PendientePRetorno>> CoefEolicoRetorno>> PresDinRetorno>>
            CExpRetorno>> AccVientoRetorno>>LargoRetorno>> AnchoRetorno>>
            PFilaRetorno>> FuerzaRetorno>> PesoPanelRetorno>>
            PesoPanelesMlRetorno>> DensidadRetorno>> DesarrolloPerfilRetorno>>
            EspesorPerfilRetorno>> VolumenTotalRetorno>> PesoTotalRetorno>>
            SobrecargaRetorno;

            ar >> UdPanelesRetorno >> UdInversoresRetorno >>
            UdAcumuladoresRetorno >> UdEstructuraRetorno >> UdOtrosRetorno >>
            UdManoRetorno >> EurosPanelesRetorno >> EurosInversoresRetorno >>
            EurosAcumuladoresRetorno >> EurosEstrcuturaRetorno
            >> EurosOtrosRetorno >> EurosManoRetorno >> TotalPanelesRetorno >>
            TotalInversoresRetorno >> TotalAcumuladoresRetorno
            >> TotalEstructuraRetorno >> TotalOtrosRetorno >> TotalManoRetorno
            >> CosteInstalRetorno >> SubvencionesRetorno
            >> LicenciasRetorno >> GGeneralesRetorno >> ProyectosRetorno >>
            CosteAImpuestosRetorno >> ImpuestosRetorno >> CosteTrasImpRetorno
            >> NecesidadesRetorno >> VidaUtilRetorno >> InteresMercadoRetorno
            >> VentaEnergRetorno >> CompraEnergRetorno >>
            CoberturaDemandaRetorno>> VANRetorno >> TIRRetorno >> OkVanRetorno
            >> OkTirRetorno;

            ar.Close();

            FicheroAbierto.Close();
            this->UpdateData(FALSE);
      }

      if (SeleccionInstalacion == 2)   //tipo instalacion conectada a red
      {
            CFile FicheroAbierto;
            FicheroAbierto.Open(Registro, CFile::modeRead);
            CArchive ar(&FicheroAbierto, CArchive::load);

            ar >>SeleccionInstalacion >> LocalidadRetorno >> ProvinciaRetorno
            >> PaisRetorno >> LatitudRetorno >> ComentariosRetorno >>
            AlfaDisenoRetorno >> BetaDisenoRetorno >> TipoPerdRetorno >>
            IncliMaxRetorno >> InclMinRetorno >> IncMaxCorrRetorno >>
            IncMinCorrRetorno >> PerdidasOrientacionRetorno >>
            PerPerdidasRetorno >> PerdSombrasRetornoRetorno >>
            HObsataculoRetorno >> HFilasRetorno >> DObstaculoRetorno >>
            DFilasRetorno >> KRetorno;

            ar>> ICortoRetorno>> IPMaxRetorno>> VAbiertoRetorno>> VPMaxRetorno
            >> PPanelRetorno>> NPanelesRetorno>> PTeoricaRetorno>>
            ToncRetorno>> ERetorno>> PInvRetorno>> TAmbRetorno>> TcRetorno >>
            LCabRetorno>> LTemRetorno>>LPolRetorno>> LDisRetorno>> LRefRetorno
            >> OtrosRetorno>> RCabRetorno>> RTemRetorno>> RPolRetorno
            >> RDisRetorno>> RRefRetorno>> OtrosRetorno>> RTotalRetorno >>
            PFovRetorno>> PNominalRetorno;

            ar>> TEneroRetorno>> TFebreroRetorno>> TMarzoRetorno>>
            TAbrilRetorno>> TMayoRetorno>> TJunioRetorno>>
            TJulioRetorno>> TAgostoRetorno>>TSeptiembreRetorno>>
            TOctubreRetorno>> TNoviembreRetorno>> TDiciembreRetorno>>
            TPromedioRetorno>> G0EneroRetorno>> G0FebreroRetorno>>
            G0MarzoRetorno>> G0AbrilRetorno >> G0MayoRetorno
```

```
            >> G0JunioRetorno>> G0JulioRetorno>> G0AgostoRetorno>>
            G0SeptiembreRetorno>> G0OctubreRetorno>>
            G0NoviembreRetorno>> G0DiciembreRetorno>> G0PromedioRetorno>>
            GEneroRetorno>> GFebreroRetorno>> GMarzoRetorno>>
            GAbrilRetorno>> GMayoRetorno>> GJunioRetorno>> GJulioRetorno>>
            GAgostoRetorno>> GSeptiembreRetorno>>
            GOctubreRetorno>> GNoviembreRetorno>> GDiciembreRetorno>>
            GPromedioRetorno>> PREneroRetorno>> PRFebreroRetorno>>
            PRMarzoRetorno>> PRAbrilRetorno>> PRMayoRetorno>> PRJunioRetorno>>
            PRJulioRetorno>> PRAgostoRetorno>> PRSeptiembreRetorno>>
            PROctubreRetorno>> PRNoviembreRetorno>> PRDiciembreRetorno>>
            PRPromedioRetorno>> EEneroRetorno>> EFebreroRetorno>>
            EMarzoRetorno>> EAbrilRetorno>> EMayoRetorno>> EJunioRetorno>>
            EJulioRetorno>> EAgostoRetorno>> ESeptiembreRetorno>>
            EOctubreRetorno>> ENoviembreRetorno>> EDiciembreRetorno>>
            EPromedioRetorno;

            ar >> ZonaRetorno >> AsperezaRetorno >> PresDinamicaRetorno >>
            AlturaInstalRetorno >> KRetorno >> LRetorno >>
            ZRetorno >> PendientePRetorno >> CoefEolicoRetorno >>
            PresDinRetorno >> CExpRetorno >> AccVientoRetorno >>
            LargoRetorno >> AnchoRetorno >> PFilaRetorno >> FuerzaRetorno >>
            PesoPanelRetorno >> PesoPanelesMlRetorno
            >> DensidadRetorno >> DesarrolloPerfilRetorno >>
            EspesorPerfilRetorno >> VolumenTotalRetorno >> PesoTotalRetorno >>
            SobrecargaRetorno;

            ar>> UdPanelesRetorno>> UdInversoresRetorno>>
            UdAcumuladoresRetorno>> UdEstructuraRetorno>>UdOtrosRetorno>>
            UdManoRetorno >> EurosPanelesRetorno>> EurosInversoresRetorno
            >>EurosAcumuladoresRetorno>> EurosEstrcuturaRetorno
            >> EurosOtrosRetorno>>EurosManoRetorno>> TotalPanelesRetorno >>
            TotalInversoresRetorno>> TotalAcumuladoresRetorno
            >> TotalEstructuraRetorno>> TotalOtrosRetorno >> TotalManoRetorno>>
            CosteInstalRetorno>> SubvencionesRetorno
            >> LicenciasRetorno>> GGeneralesRetorno>> ProyectosRetorno >>
            CosteAImpuestosRetorno>> ImpuestosRetorno>> CosteTrasImpRetorno
            >> NecesidadesRetorno>> VidaUtilRetorno>>InteresMercadoRetorno >>
            VentaEnergRetorno>> CompraEnergRetorno>> CoberturaDemandaRetorno
            >> VANRetorno>> TIRRetorno >> OkVanRetorno>> OkTirRetorno;

            ar.Close();

            FicheroAbierto.Close();
            this->UpdateData(FALSE);
      }
}
```

La función “ExtraerTeporales”, es llamada cuando se intenta guardar un archivo serializado desde la caja de dialogo “guardar como”, siendo que capturado previamente el nombre de archivo a manipular, este será procesado a fin de añadirle la terminación correcta.

Tal labor se realiza, dado que los datos son guardados en dos archivos asociados, uno que forma la información de los puntos de consumo de la instalación, y otro que contiene toda la información del resto de pantallas del programa. Según esto, en la primera fase de guardado, se capturará el nombre con el que se guardarán los datos

generales, realizándose en esta función el renombrado del archivo asociado a tales datos formado por las bases de consumo, de manera que se denominan con el mismo nombre, pero con diferente terminación, el archivo de datos de consumo y el de datos generales de la aplicación.

```
void CPFCDlg::ExtraerTemporales()
{
	//modifico el archivo temporal que contiene los puntos de consumo, para
	poder asociarlo
	// al nombre de archivo de guardado y posteriormente poder deserializarlo
	fácilmente
	CString term = _T(".dat");
	CString Asociado = m_TextoEdit2+term;
	CFile::Rename(_T("TempCons"), Asociado); //modifica nombre de archivo
	temporal

}
```

La página principal del programa, al igual que las funciones anteriores, alberga cualquier acción del menú general desplegable, dando esta contenido a cajas de dialogo de información e versión, lanzamiento de cualquier documentación relevante (manuales de instrucciones, etc…) y en general cualquier tipo de acción que se ha relacionado, o se quiera relacionar con acciones de menú.

```
void CPFCDlg::OnAyudaVersi32797()
{
	// TODO: Agregue aquí su código de controlador de comandos
	//llama cuadro dialogo versión producto
	CAboutDlg dlgAbout;
	dlgAbout.DoModal();
}

void CPFCDlg::OnDocumentaislada()
{
	// TODO: Agregue aquí su código de controlador de comandos
	ShellExecute(NULL, _T("open"), _T("PInstAisladas.pdf"), NULL, NULL,
SW_SHOWNORMAL);
}


void CPFCDlg::OnDocumentred()
{
	// TODO: Agregue aquí su código de controlador de comandos
	ShellExecute(NULL, _T("open"), _T("PInstRed.pdf"), NULL, NULL,
SW_SHOWNORMAL);
}


void CPFCDlg::OnAyudaManualusuario()
{
	// TODO: Agregue aquí su código de controlador de comandos
	ShellExecute(NULL, _T("open"), _T("manual_usuario_Heclia.pdf"), NULL,
NULL, SW_SHOWNORMAL);
}
```

3.- Instalaciones fotovoltaicas aisladas de red

La clase y control que gestiona la parte de la aplicación correspondiente a introducción de puntos de consumo, se insertará en la presente página de código o archivo cpp, de manera que cuando es llamada a través del cuadro de diálogo principal, se lanza el conjunto de instrucciones que la forman y que dan cabida a sus respectivos controles de acción (cuadros de lista, cuadros de edición, etc…).

De igual manera a esta sección del código, el fragmento o página que da cabida a las mismas instrucciones, pero derivadas de la clase que alberga los puntos de consumo de instalaciones conectadas a red, se desarrollarán de manera idéntica (por lo cual no se explican en el presente anexo), siendo que exclusivamente variará la clase de la que se derivan. Se podría haber ejecutado una clase común para ambas configuraciones, pero se ha elegido separarlas, a fin de que no puedan existir confusiones o interacciones entre un tipo de instalación y otra.

Se obvia en la descripción del código, los fragmentos iniciales, correspondientes a inicialización de clases, inicialización de variables tipo "*extern*", declaración de funciones, declaración de comando o herramientas… pasándose directamente a la declaración de funciones que se manejan en esta clase.

Cuando el usuario selección la inserción de algún elemento o punto de consumo nuevo, este se puede insertar en tres cuadro de lista diferentes, en función del seleccionado y tipo de objeto elegido (puntos de consumo generales, bombas o puntos de autoconsumo), siendo que cada pulsación para añadir un elemento en su cuadro correspondiente, hará una llamada a un segundo cuadro de dialogo, que albergará las herramientas de intercambio oportunas.

```
void Consumos::OnBnClickedAnadirpunt()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	Puntual DlgPuntual(NULL);	//Llama al constructor del dialogo para
introducir puntos consumo

	if (DlgPuntual.DoModal() == IDOK)

	{

	}

	if (dato)
	{
```

Una vez devueltos los datos del cuadro de dialogo de inserción, estos se procesarán, gestionarán e insertarán cada una de sus listas correspondientes. No se explica el funcionamiento interno de este control de lista, dado que se ha explicado previamente en la memoria del presente proyecto.

```
		//Añadir nombre referencia consumo puntual a la vista
		UpdateData(TRUE);

		//añadir fila al control de lista
		int nItem;

		nItem = m_DatosPuntuales.InsertItem(0, definicionPuntualAux);

		//dar formato a los archivos numéricos para salir en pantalla como
		texto
		CString t;
		CString y;
		CString u;
		t.Format(_T("%f"), energiaPuntualAux);
		y.Format(_T("%d"), unidadesPuntualAux);
		u.Format(_T("%f"), totalPuntualAux);

		// insertar en cada columna de una misma línea cada uno de los
		datos de los elementos
		m_DatosPuntuales.SetItemText(nItem, 0, definicionPuntualAux);
		m_DatosPuntuales.SetItemText(nItem, 1, t);
		m_DatosPuntuales.SetItemText(nItem, 2, y);
		m_DatosPuntuales.SetItemText(nItem, 3, u);
```

Además de la introducción en la lista, se ha implementado un contador, de forma que se compute, en un cuadro de edición, fuera de las herramientas de listado, el cómputo global de la energía correspondiente a ese capítulo, siendo que tal acción, permitirá sumar cada elemento introducido, o restar cada elemento retirado.

```
		//contador salida energía sección puntual
		m_TPunt = m_TPunt + totalPuntualAux;
```

La llamada a la presente función, trasladara el flujo de ejecución a una función específica para totalizar la suma de todos los cuadros de lista (suma de potencias de consumo).

```
		ActualizarPuntual();

		//actualizar variables salida
		UpdateData(TRUE);
	}
}

// zona lanzamiento de cuadro de dialogo de añadir datos bombas

void Consumos::OnBnClickedAnadirbomba()
{
	// TODO: Agregue aquí su código de controlador de notificación de control


	bombas DlgBombas(NULL);    //Llama al constructor del dialogo para
introducir puntos consumo

	if (DlgBombas.DoModal() == IDOK)

	{
```

```
	}

	if (dato)
	{

		//Añadir nombre referencia consumo puntual a la vista

		UpdateData(TRUE);

		//añadir fila al control de lista
		int nItemBombas;

		nItemBombas = m_DatosBombas.InsertItem(0, DefinicionBombaAux);

		//dar formato a los archivos numéricos para salir en pantalla como
		texto
		CString q, w, e, r, t, y, u, i, o, p, a, s;

		q.Format(_T("%f"), EmbAux);
		w.Format(_T("%d"), UdBombasAux);
		e.Format(_T("%f"), TotalBombasAux);
		r.Format(_T("%f"), RendimientoAux);
		t.Format(_T("%f"), HdAux);
		y.Format(_T("%f"), HstAux);
		u.Format(_T("%f"), HdtAux);
		i.Format(_T("%f"), HteAux);
		o.Format(_T("%f"), HfAux);
		p.Format(_T("%f"), QdAux);
		a.Format(_T("%f"), QtAux);
		s.Format(_T("%f"), QapAux);

		// insertar en cada columna de una misma linea cada uno de los
		datos de los elementos
		m_DatosBombas.SetItemText(nItemBombas, 0, DefinicionBombaAux);
		m_DatosBombas.SetItemText(nItemBombas, 1, q);
		m_DatosBombas.SetItemText(nItemBombas, 2, w);
		m_DatosBombas.SetItemText(nItemBombas, 3, e);
		m_DatosBombas.SetItemText(nItemBombas, 4, r);
		m_DatosBombas.SetItemText(nItemBombas, 5, t);
		m_DatosBombas.SetItemText(nItemBombas, 6, y);
		m_DatosBombas.SetItemText(nItemBombas, 7, u);
		m_DatosBombas.SetItemText(nItemBombas, 8, i);
		m_DatosBombas.SetItemText(nItemBombas, 9, o);
		m_DatosBombas.SetItemText(nItemBombas, 10, p);
		m_DatosBombas.SetItemText(nItemBombas, 11, a);
		m_DatosBombas.SetItemText(nItemBombas, 12, s);

		//salida contadores energía bombas
		m_TBom = m_TBom + TotalBombasAux;
		ActualizarBombas();

		//actualizar variables salida
		UpdateData(TRUE);
	}
}

// Zona de diálogo de añadir datos de autoconsumos

void Consumos::OnBnClickedAnadirautocons()
{
```

```
	// TODO: Agregue aquí su código de controlador de notificación de control

	Autoconsumos DlgAutoConsumos(NULL);	//Llama al constructor del dialogo
	para introducir puntos consumo

	if (DlgAutoConsumos.DoModal() == IDOK)

	{

	}

	if (dato)
	{

		//Añadir nombre referencia consumo puntual a la vista

		UpdateData(TRUE);

		//añadir fila al control de lista
		int nItem;

		nItem = m_DatosAutoconsumo.InsertItem(0, DefinicionAutoconsumoAux);

		//dar formato a los archivos numéricos para salir en pantalla como
		texto
		CString t;
		CString y;
		CString u;
		t.Format(_T("%f"), EnergiaAutoconsumoAux);
		y.Format(_T("%d"), UdAutoconsumoAux);
		u.Format(_T("%f"), TotalAutoconsumoAux);

		// insertar en cada columna de una misma linea cada uno de los
		datos de los elementos
		m_DatosAutoconsumo.SetItemText(nItem, 0, DefinicionAutoconsumoAux);
		m_DatosAutoconsumo.SetItemText(nItem, 1, t);
		m_DatosAutoconsumo.SetItemText(nItem, 2, y);
		m_DatosAutoconsumo.SetItemText(nItem, 3, u);

		//salida contadores autoconsumos
		m_TAuto = m_TAuto + TotalAutoconsumoAux;
		ActualizarAutoconsumos();

		//actualizar variables salida
		UpdateData(TRUE);
	}
}
```

La aplicación al igual que se inserta un punto de consumo, en función de su clase, en cada una de sus herramientas o zonas adecuadas, debe permitir realizar la acción inversa, mediante el borrado del elemento en cuestión.

```
// controles de borrado de cada elemento de sus respectivas listas

void Consumos::OnBnClickedBorrarpunt()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
```

Al marcarse en la lista un elemento, y pulsar el botón de borrado, se seleccionará tal elemento (o más bien su contenido de referencia), mediante la función *DeleteItem*, asociada a la variable miembro que caracteriza el control, eliminando la misma del mapa del control referido.

```
	// Obtenemr el elemento seleccionao de la lista

	// borrar el elemento de la lista
	m_DatosPuntuales.DeleteItem(Elemento);
```

El proceso de conteo, para restar el consumo del elemento borrado, será de manera completa análogo al de inserción de un elemento previamente descrito, siendo el valor del elemento obviamente negativo.

```
	//restar unidades de los cuadros de energía parcial y total
	m_TPunt = m_TPunt - EnergTPuntual;
	ActualizarPuntual();
```

Se pone la variable del contador a 0 una vez eliminado el elemento, o más bien al salir o retornar de la función de conteo, dado que en caso en caso contrario, la variable de descuento quedaría cargada con el ultimo valor seleccionado del elemento borrado, en cuyo caso, cualquier otro paso por esta función, aun cuando finalmente no eliminase el elemento, seguiría borrando de manera errónea la cantidad preseleccionada.

```
	//pone valor de borrado a 0 para que en caso de no seleccionar otro
	elemento no pueda restar por error
	EnergTPuntual = 0;

	//actualizar variables salida
	UpdateData(TRUE);
}


void Consumos::OnBnClickedBorrarbomba()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	// borrar el elemento de la lista
	m_DatosBombas.DeleteItem(Elemento);

	//restar unidades de los cuadros de energía parcial y total
	m_TBom = m_TBom - EnergTBombas;
	ActualizarBombas();

	//pone valor de borrado a 0 para que en caso de no seleccionar otro
	elemento no pueda restar por error
	EnergTBombas = 0;
	//actualizar variables salida
	UpdateData(TRUE);
}

void Consumos::OnBnClickedBorrarautocons()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	// borrar el elemento de la lista
	m_DatosAutoconsumo.DeleteItem(Elemento);

	//restar unidades de los cuadros de energía parcial y total
```

```
	m_TAuto = m_TAuto - EnergTAutoconsumo;
	ActualizarAutoconsumos();

	//pone valor de borrado a 0 para que en caso de no seleccionar otro
	elemento no pueda restar por error
	EnergTAutoconsumo = 0;

	//actualizar variables salida
	UpdateData(TRUE);
}
```

Los contadores de totalización, al igual que se hacía específicamente para cada elemento de los tres disponibles, suman o restan todos los consumo, en un cuadro de dialogo. Esto es, se ha explicado previamente una acumulación o contador para cada uno de los puntos de consumo introducidos en los controles de listas, realizándose la misma función, en global para todos los elementos en conjunto, a fin de saber en una interface el cómputo total de consumos del sistema.

```
// sistema de visualización de indicadores de totalización de consumos de cada
grupo y total
//ejecuta cuentas cada vez que se modifican las vistas de ListBox que es cuando
se añaden o quitan elementos.

void Consumos::ActualizarPuntual()
{

	m_SumTotal = m_TPunt + m_TBom + m_TAuto;
	SumaTotalAux = m_SumTotal;
	UpdateData(FALSE);

	return;
}

void Consumos::ActualizarBombas()
{

	m_SumTotal = m_TPunt + m_TBom + m_TAuto;
	SumaTotalAux = m_SumTotal;
	UpdateData(FALSE);

	return;
}

void Consumos::ActualizarAutoconsumos()
{
	m_TAuto;
	m_SumTotal = m_TPunt + m_TBom + m_TAuto;
	SumaTotalAux = m_SumTotal;
	UpdateData(FALSE);

	return;
}
```

Un vez el usuario haya terminado de introducir puntos de consumo energético en la primera pantalla de la aplicación, pasará a la siguiente pulsando el botón especifico al uso, circunstancia que entre otras cosas, lanzara el cuadro de diálogo modal

correspondiente a ésta, pero además de manera interna se implementa código para realizar otro tipo de acciones.

```
// Punto de cambio o lanzamiento de siguiente pantalla del programa
void Consumos::OnBnClickedSigdiseno()
{
```

En función del tipo de instalación que se esté ejecutando, se generará un archivo de guardado temporal de la información que se ha registrado en la pantalla, a fin de que esta se presente disponible para el guardado en caso de no terminarse el desarrollo del proyecto, para recuperarla en caso de volver o retroceder en a las pantallas previas, y en general conservar de manera no volátil los datos introducidos.

```
    // TODO: Agregue aquí su código de controlador de notificación de control
    Aislada = true;
    ConectadaRed = false;

    //genera un archivo de guardado temporal de los datos de consumo
    this->UpdateData();
```

Se recuperara el número de elementos disponibles encada control de lista, mediante la función específica "*GetItemCount()*".

```
    ItemPuntual = m_DatosPuntuales.GetItemCount();
    ItemBomba = m_DatosBombas.GetItemCount();
    ItemAuto = m_DatosAutoconsumo.GetItemCount();
```

Se genera un archivo de escritura, no explicándose esta labor dado que las acciones de guardado y gestión de la información se han explicado en la memoria del proyecto.

```
    CString TempConsum = _T("TempCons");

    //abrir el fichero para escribir
    CFile ArchTemp;
    //estructura para almacenar el estado del fichero
    ArchTemp.Open(TempConsum, CFile::modeCreate | CFile::modeWrite);

    //bucle de carga y seriación de cada una de las lineas de la base de datos
    hasta que se terminen
    CArchive ar(&ArchTemp, CArchive::store);

    ar << ItemPuntual << ItemBomba << ItemAuto;

    ItemPuntual = ItemPuntual - 1; //resta uno para recuperar datos del valor
    0;
    ItemBomba = ItemBomba - 1;
    ItemAuto = ItemAuto - 1;
```

Se realizan bucles, en los cuales, hasta que no se hayan terminado los elementos registrados en la lista, y de los cuales hemos comprobado el número, se recuperan secuencialmente los datos de cada uno de los valores que generan el mapa de la lista (dato principal y todos los asociados al mismo). A tal respecto se usa la función *GetItemText*, asociada a la variable miembro de la lista, y se cargan en variables auxiliares, las cuales se serializaran, en el interior del archivo de disco en cada ciclo del

bucle. El archivo no se cerrara hasta que se hayan conectados todos lo bucles de todos los cuadros de lista.

```
while (ItemPuntual > -1)   // mientras haya objetos en la lista
{
	//Recuperación de los valores de los elementos de lista en formato
	texto
	DefinicionPunt = m_DatosPuntuales.GetItemText(ItemPuntual, 0);
	EnergPunt = m_DatosPuntuales.GetItemText(ItemPuntual, 1);
	UDPunt = m_DatosPuntuales.GetItemText(ItemPuntual, 2);
	EnerT = m_DatosPuntuales.GetItemText(ItemPuntual, 3);

	ItemPuntual = ItemPuntual - 1;

	ar << DefinicionPunt << EnergPunt << UDPunt << EnerT;
}

while (ItemBomba > -1)   // mientras haya objetos en la lista

{
	//Recuperación de los valores de los elementos de lista en formato
	texto
	DefBom = m_DatosBombas.GetItemText(ItemBomba, 0);
	EnerBom = m_DatosBombas.GetItemText(ItemBomba, 1);
	UDBom = m_DatosBombas.GetItemText(ItemBomba, 2);
	ETBom = m_DatosBombas.GetItemText(ItemBomba, 3);
	RendBom = m_DatosBombas.GetItemText(ItemBomba, 4);
	HdBom = m_DatosBombas.GetItemText(ItemBomba, 5);
	HstBom = m_DatosBombas.GetItemText(ItemBomba, 6);
	HdtBom = m_DatosBombas.GetItemText(ItemBomba, 7);
	HteBom = m_DatosBombas.GetItemText(ItemBomba, 8);
	HfBom = m_DatosBombas.GetItemText(ItemBomba, 9);
	QdBom = m_DatosBombas.GetItemText(ItemBomba, 10);
	QtBom = m_DatosBombas.GetItemText(ItemBomba, 11);
	QapBom = m_DatosBombas.GetItemText(ItemBomba, 12);

	ItemBomba = ItemBomba - 1;

	ar << DefBom << EnerBom << UDBom << ETBom << RendBom << HdBom <<
	HstBom << HdtBom << HteBom << HfBom <<QdBom << QtBom << QapBom;
}

while (ItemAuto > -1)   // mientras haya objetos en la lista
{
	//Recuperación de los valores de los elementos de lista en formato
	texto
	DefinicionAuto = m_DatosAutoconsumo.GetItemText(ItemAuto, 0);
	EnergAuto = m_DatosAutoconsumo.GetItemText(ItemAuto, 1);
	UDAuto = m_DatosAutoconsumo.GetItemText(ItemAuto, 2);
	EnerTAuto = m_DatosAutoconsumo.GetItemText(ItemAuto, 3);

	ItemAuto = ItemAuto - 1;

	ar << DefinicionAuto << EnergAuto << UDAuto << EnerTAuto;
}

//seriacion datos consumos
ar << m_TPunt << m_TBom << m_TAuto << m_SumTotal;
```

```
	//cierre seriación y archivo datos
	ar.Close();
	ArchTemp.Close();
```

De manera añadida, se externaliza una función, que genera el mismo archivo guardado como datos, en un archivo de texto, por si el usuario desea obtenerlo, imprimirlo o manipularlo

```
	//llama a la función de generación de archivo de salida
	ArchivoSalida();
```

Se llama a la siguiente pantalla del programa invocada por el usuario mediante cuadro de dialogo modal.

```
	DGenerador DlgDGenerador(NULL);   //Llama al constructor del dialogo para
	lanzar la pantalla

	if (DlgDGenerador.DoModal() == IDOK)

	{

	}

}
```

La función *OnInitDialog*, inicializa las bases generales y funcionamiento de los controles de lista, habiéndose explicado esta inicialización por defecto al llamarse al constructor de la clase en secciones correspondientes de explicación de controles de memoria de proyecto.

```
BOOL Consumos::OnInitDialog()
{
	CDialogEx::OnInitDialog();

	// TODO:  Agregue aquí la inicialización adicional

	//se inician las columnas del control de lista de datos puntuales
	m_DatosPuntuales.InsertColumn(0, _T("Definición punto consumo"),
	LVCFMT_LEFT, 290);
	m_DatosPuntuales.InsertColumn(1, _T("Energía unitaria (Wh/dia)"),
	LVCFMT_CENTER, 140);
	m_DatosPuntuales.InsertColumn(2, _T("Unidades"), LVCFMT_CENTER, 65);
	m_DatosPuntuales.InsertColumn(3, _T("T. energía (Wh/dia)"), LVCFMT_CENTER,
	150);

	//se inician las columnas del control de lista de datos puntuales
	m_DatosBombas.InsertColumn(0, _T("Definición bomba"), LVCFMT_LEFT, 225);
	m_DatosBombas.InsertColumn(1, _T("Energia"), LVCFMT_CENTER, 50);
	m_DatosBombas.InsertColumn(2, _T("Unidades"), LVCFMT_CENTER, 50);
	m_DatosBombas.InsertColumn(3, _T("T. Energia"), LVCFMT_CENTER, 50);
	m_DatosBombas.InsertColumn(4, _T("Rend."), LVCFMT_CENTER, 40);
	m_DatosBombas.InsertColumn(5, _T("Hd"), LVCFMT_CENTER, 30);
	m_DatosBombas.InsertColumn(6, _T("Hst"), LVCFMT_CENTER, 30);
	m_DatosBombas.InsertColumn(7, _T("Hdt"), LVCFMT_CENTER, 30);
	m_DatosBombas.InsertColumn(8, _T("Hte"), LVCFMT_CENTER, 30);
	m_DatosBombas.InsertColumn(9, _T("Hf"), LVCFMT_CENTER, 30);
	m_DatosBombas.InsertColumn(10, _T("Qd"), LVCFMT_CENTER, 30);
```

```
	m_DatosBombas.InsertColumn(11, _T("Qt"), LVCFMT_CENTER, 30);
	m_DatosBombas.InsertColumn(12, _T("Qap"), LVCFMT_CENTER, 30);

	//se inician las columnas del control de lista de datos autoconsumo
	m_DatosAutoconsumo.InsertColumn(0, _T("Definición punto Autoconsumo"),
	LVCFMT_LEFT, 290);
	m_DatosAutoconsumo.InsertColumn(1, _T("Energía unitaria (Wh/dia)"),
	LVCFMT_CENTER, 140);
	m_DatosAutoconsumo.InsertColumn(2, _T("Unidades"), LVCFMT_CENTER, 65);
	m_DatosAutoconsumo.InsertColumn(3, _T("T. energía (Wh/dia)"),
	LVCFMT_CENTER, 150);

	//genera un archivo de salida en blanco
	ofstream GeneracionSalidaConsumos;
	GeneracionSalidaConsumos.open("SalidaConsumosAisladaRed.txt", ios::out);
	GeneracionSalidaConsumos.close();
```

Se añade a la función de inicio de los controles, la opción de que no se esté ejecutando por primera vez la pantalla, esto es, en un primer ciclo de ejecución, la inicialización pondrá nombres a las columnas y gestionará los controles, pero no escribirá dato alguno, sin embargo, si no es una nueva generación de proyecto, sino un archivo preexistente, o que se ha abierto de un archivo de disco duro, llamará a la función *RecuperacionInfo()*, para que los datos contenidos en el archivo, sean directamente cargado al invocarse al constructor de la clase y por extensión a la inicialización de los controles.

```
	if (AbiertoAachivo)
	{
		// llamada a función que deserializa todo el conjunto almacenado
		RecuperacionInfo();
		UpdateData(FALSE);
	}
	return TRUE;  // return TRUE unless you set the focus to a control
}


void Consumos::OnNMClickVistapuntual(NMHDR *pNMHDR, LRESULT *pResult)
{
	//Funcion para capturar la pulsación dl ratón en la lista de autoconsumos
	LPNMITEMACTIVATE pNMItemActivate =
reinterpret_cast<LPNMITEMACTIVATE>(pNMHDR);
	// TODO: Agregue aquí su código de controlador de notificación de control

	NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;

	//obtención del número de elemento o referencia lista
	Elemento = m_DatosPuntuales.GetHotItem();

	//Recuperación de los valores de los elementos de lista en formato texto
	Nombre = m_DatosPuntuales.GetItemText(Elemento, 0);
	EUnitaria = m_DatosPuntuales.GetItemText(Elemento, 1);
	Unid = m_DatosPuntuales.GetItemText(Elemento, 2);
	EnTotal = m_DatosPuntuales.GetItemText(Elemento, 3);

	//conversión de los valores de texto a valores numéricos para su uso
	EnergUPuntual = _wtof(EUnitaria);
	Ud = _wtoi(Unid);
	EnergTPuntual = _wtof(EnTotal);
```

```
	*pResult = 0;
}

void Consumos::OnNMClickVistabombas(NMHDR *pNMHDR, LRESULT *pResult)
{
	//Funcion para capturar la pulsación dl ratón en la lista de autoconsumos
	LPNMITEMACTIVATE pNMItemActivate =
reinterpret_cast<LPNMITEMACTIVATE>(pNMHDR);
	// TODO: Agregue aquí su código de controlador de notificación de control

	NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;

	//obtención del número de elemento o referencia lista
	Elemento = m_DatosBombas.GetHotItem();

	//Recuperación de los valores de los elementos de lista en formato texto
	Nombre = m_DatosBombas.GetItemText(Elemento, 0);
	EUnitaria = m_DatosBombas.GetItemText(Elemento, 1);
	Unid = m_DatosBombas.GetItemText(Elemento, 2);
	EnTotal = m_DatosBombas.GetItemText(Elemento, 3);
	Rendimiento = m_DatosBombas.GetItemText(Elemento, 4);
	Hd = m_DatosBombas.GetItemText(Elemento, 5);
	Hst = m_DatosBombas.GetItemText(Elemento, 6);
	Hdt = m_DatosBombas.GetItemText(Elemento, 7);
	Hte = m_DatosBombas.GetItemText(Elemento, 8);
	Hf = m_DatosBombas.GetItemText(Elemento, 9);
	Qd = m_DatosBombas.GetItemText(Elemento, 10);
	Qt = m_DatosBombas.GetItemText(Elemento, 11);
	Qap = m_DatosBombas.GetItemText(Elemento, 12);

	//conversión de los valores de texto a valores numéricos para su uso
	EnergUBombas = _wtof(EUnitaria);
	Ud = _wtoi(Unid);
	EnergTBombas = _wtof(EnTotal);
	Rendimientoconv = _wtof(Rendimiento);
	Hdconv = _wtof(Hd);
	Hstconv = _wtof(Hst);
	Hdtconv = _wtof(Hdt);
	Hteconv = _wtof(Hte);
	Hfconv = _wtof(Hf);
	Qdconv = _wtof(Qd);
	Qtconv = _wtof(Qt);
	Qapconv = _wtof(Qap);

	*pResult = 0;
}

void Consumos::OnNMClickVistaautoconsumo(NMHDR *pNMHDR, LRESULT *pResult)
{
	//Funcion para capturar la pulsación dl ratón en la lista de autoconsumos
	LPNMITEMACTIVATE pNMItemActivate =
reinterpret_cast<LPNMITEMACTIVATE>(pNMHDR);
	// TODO: Agregue aquí su código de controlador de notificación de control

	NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;

	//obtención del número de elemento o referencia lista
	Elemento = m_DatosAutoconsumo.GetHotItem();
```

```
	//Recuperación de los valores de los elementos de lista en formato texto
	Nombre = m_DatosAutoconsumo.GetItemText(Elemento, 0);
	EUnitaria = m_DatosAutoconsumo.GetItemText(Elemento, 1);
	Unid = m_DatosAutoconsumo.GetItemText(Elemento, 2);
	EnTotal = m_DatosAutoconsumo.GetItemText(Elemento, 3);

	//conversión de los valores de texto a valores numéricos para su uso
	EnergUAutoconsumo = _wtof(EUnitaria);
	Ud = _wtoi(Unid);
	EnergTAutoconsumo = _wtof(EnTotal);

	*pResult = 0;
}
```

La siguiente función implementada, e invocada cuando el usuario cambia de vista del programa, genera un archivo de texto de salida con toda la información introducida en los cuadros o controles que marcan los puntos de consumo (con independencia de su clase). No se entra a explicar las acciones realizadas en la generación de este tipo de archivo de salida, dado que han sido explicadas en la memoria del presente proyecto.

```
//Genera archivo TXT de los puntos de consumo
void Consumos::ArchivoSalida()
{
	//obtención del número de elementos o referencia lista elementos
	ElementoPuntual = m_DatosPuntuales.GetItemCount();
	ElementoPuntual = ElementoPuntual - 1; //resta uno para recuperar datos
	del valor 0;
	ElementoBomba = m_DatosBombas.GetItemCount();
		ElementoBomba = ElementoBomba - 1; //resta uno para recuperar datos
	del valor 0;
	ElementoAuto = m_DatosAutoconsumo.GetItemCount();
	ElementoAuto = ElementoAuto - 1; //resta uno para recuperar datos del
	valor 0;

	//crea el archivo TXT en el que se va a generar el archivo
	ofstream GeneracionSalidaConsumos;
	GeneracionSalidaConsumos.open("SalidaConsumosAisladaRed.txt", ios::out |
	ios::app); //archivo que escribe secuencial

	//salidas de texto de puntos consumo
	GeneracionSalidaConsumos << "PUNTOS DE CONSUMO DE INSTALACION FOTOVOLTAICA
	AISLADA DE RED\n";
	GeneracionSalidaConsumos << "\n";
	while (ElementoPuntual > -1)   // mientras haya objetos en la lista
	{
		//Recuperación de los valores de los elementos de lista en formato
		texto
		NombreSalida = m_DatosPuntuales.GetItemText(ElementoPuntual, 0);
		EUnitariaSalida = m_DatosPuntuales.GetItemText(ElementoPuntual, 1);
		UnidSalida = m_DatosPuntuales.GetItemText(ElementoPuntual, 2);
		EnTotalSalida = m_DatosPuntuales.GetItemText(ElementoPuntual, 3);
		ElementoPuntual = ElementoPuntual - 1;

		//cargar en archivo texto
		int Definicion = NombreSalida.GetLength() * sizeof(TCHAR);
		int Energia = EUnitariaSalida.GetLength() * sizeof(TCHAR);
		int unidades = UnidSalida.GetLength() * sizeof(TCHAR);
```

```
		int Totales = EnTotalSalida.GetLength() * sizeof(TCHAR);

		GeneracionSalidaConsumos << "Definición:" << "
		" << "Energía unitaria" << "      " << "Unidades" << "      " <<
		"Total Energía" << "\n";
		GeneracionSalidaConsumos.write((LPCSTR)(LPCTSTR)NombreSalida,
		Definicion);
		GeneracionSalidaConsumos << "           ";
		GeneracionSalidaConsumos.write((LPCSTR)(LPCTSTR)EUnitariaSalida,
		Energia);
		GeneracionSalidaConsumos << "           ";
		GeneracionSalidaConsumos.write((LPCSTR)(LPCTSTR)UnidSalida,
		unidades);
		GeneracionSalidaConsumos << "           ";
		GeneracionSalidaConsumos.write((LPCSTR)(LPCTSTR)EnTotalSalida,
		Totales);
		GeneracionSalidaConsumos << "\n";
		GeneracionSalidaConsumos << "\n";
	}

	GeneracionSalidaConsumos <<
"##############################################################################
\n";
	GeneracionSalidaConsumos << "\n";

	//salidas de texto de puntos consumo bombas
	GeneracionSalidaConsumos << "SISTEMAS BOMBEO DE INSTALACION FOTOVOLTAICA
	AISLADA DE RED\n";
	GeneracionSalidaConsumos << "\n";
	while (ElementoBomba > -1)   // mientras haya objetos en la lista
	{
		//Recuperación de los valores de los elementos de lista en formato
		texto
		NombreSalidaBomba = m_DatosBombas.GetItemText(ElementoBomba, 0);
		EUnitariaSalidaBomba = m_DatosBombas.GetItemText(ElementoBomba, 1);
		UnidSalidaBomba = m_DatosBombas.GetItemText(ElementoBomba, 2);
		EnTotalSalidaBomba = m_DatosBombas.GetItemText(ElementoBomba, 3);
		RendimientoSalidaBomba = m_DatosBombas.GetItemText(ElementoBomba,
		4);
		HdSalidaBomba = m_DatosBombas.GetItemText(ElementoBomba, 5);
		HstSalidaBomba = m_DatosBombas.GetItemText(ElementoBomba, 6);
		HdtSalidaBomba = m_DatosBombas.GetItemText(ElementoBomba, 7);
		HteSalidaBomba = m_DatosBombas.GetItemText(ElementoBomba, 8);
		HfSalidaBomba = m_DatosBombas.GetItemText(ElementoBomba, 9);
		QdSalidaBomba = m_DatosBombas.GetItemText(ElementoBomba, 10);
		QtSalidaBomba = m_DatosBombas.GetItemText(ElementoBomba, 11);
		QapSalidaBomba = m_DatosBombas.GetItemText(ElementoBomba, 12);
		ElementoBomba = ElementoBomba - 1;

		//cargar en archivo texto
		int DefinicionBomba = NombreSalidaBomba.GetLength() *
		sizeof(TCHAR);
		int EnergiaBomba = EUnitariaSalidaBomba.GetLength() *
		sizeof(TCHAR);
		int unidadesBomba = UnidSalidaBomba.GetLength() * sizeof(TCHAR);
		int TotalesBomba = EnTotalSalidaBomba.GetLength() * sizeof(TCHAR);
		int RendBomba = RendimientoSalidaBomba.GetLength() * sizeof(TCHAR);
		int HdBomba = HdSalidaBomba.GetLength() * sizeof(TCHAR);
		int HstBomba = HstSalidaBomba.GetLength() * sizeof(TCHAR);
		int HdtBomba = HdtSalidaBomba.GetLength() * sizeof(TCHAR);
```

```
            int HteBomba = HteSalidaBomba.GetLength() * sizeof(TCHAR);
            int HfBomba = HfSalidaBomba.GetLength() * sizeof(TCHAR);
            int QdBomba = QdSalidaBomba.GetLength() * sizeof(TCHAR);
            int QtBomba = QtSalidaBomba.GetLength() * sizeof(TCHAR);
            int QapBomba = QapSalidaBomba.GetLength() * sizeof(TCHAR);

GeneracionSalidaConsumos << "Definición:" << "                              " <<
"Energía unitaria" << "      " << "Unidades" << "      " << "Total Energía" << "
" << "Rendimiento" << "      " << "H Deposito" << "      " << "Nivel est." << "
" << "Nivel din." << "      " << "H. eqiv." << "      " << "H fricc." << "      "
<< "Q req." << "      " << "Q prueb." << "      " << "Q medio" << "\n";
GeneracionSalidaConsumos.write((LPCSTR)(LPCTSTR)NombreSalidaBomba,
DefinicionBomba);
GeneracionSalidaConsumos << "           ";
GeneracionSalidaConsumos.write((LPCSTR)(LPCTSTR)EUnitariaSalidaBomba,
EnergiaBomba);
GeneracionSalidaConsumos << "           ";
GeneracionSalidaConsumos.write((LPCSTR)(LPCTSTR)UnidSalidaBomba, unidadesBomba);
GeneracionSalidaConsumos << "           ";
GeneracionSalidaConsumos.write((LPCSTR)(LPCTSTR)EnTotalSalidaBomba,
TotalesBomba);
GeneracionSalidaConsumos << "           ";
GeneracionSalidaConsumos.write((LPCSTR)(LPCTSTR)RendimientoSalidaBomba,
RendBomba);
GeneracionSalidaConsumos << "           ";
GeneracionSalidaConsumos.write((LPCSTR)(LPCTSTR)HdSalidaBomba, HdBomba);
GeneracionSalidaConsumos << "           ";
GeneracionSalidaConsumos.write((LPCSTR)(LPCTSTR)HstSalidaBomba, HstBomba);
GeneracionSalidaConsumos << "           ";
GeneracionSalidaConsumos.write((LPCSTR)(LPCTSTR)HdtSalidaBomba, HdtBomba);
GeneracionSalidaConsumos << "           ";
GeneracionSalidaConsumos.write((LPCSTR)(LPCTSTR)HteSalidaBomba, HteBomba);
GeneracionSalidaConsumos << "           ";
GeneracionSalidaConsumos.write((LPCSTR)(LPCTSTR)HfSalidaBomba, HfBomba);
GeneracionSalidaConsumos << "           ";
GeneracionSalidaConsumos.write((LPCSTR)(LPCTSTR)QdSalidaBomba, QdBomba);
GeneracionSalidaConsumos << "           ";
GeneracionSalidaConsumos.write((LPCSTR)(LPCTSTR)QtSalidaBomba, QtBomba);
GeneracionSalidaConsumos << "           ";
GeneracionSalidaConsumos.write((LPCSTR)(LPCTSTR)QapSalidaBomba, QapBomba);
GeneracionSalidaConsumos << "\n";
GeneracionSalidaConsumos << "\n";
      }

      GeneracionSalidaConsumos <<
"##############################################################################
\n";
      GeneracionSalidaConsumos << "\n";

      //salidas de texto de puntos consumo
      GeneracionSalidaConsumos << "PUNTOS DE AUTOCONSUMO DE INSTALACION
      FOTOVOLTAICA AISLADA DE RED\n";
      GeneracionSalidaConsumos << "\n";
      while (ElementoAuto > -1)   // mientras haya objetos en la lista
      {
            //Recuperación de los valores de los elementos de lista en formato
            texto
            NombreSalidaAuto = m_DatosAutoconsumo.GetItemText(ElementoAuto, 0);
            EUnitariaSalidaAuto = m_DatosAutoconsumo.GetItemText(ElementoAuto,
            1);
```

```
                UnidSalidaAuto = m_DatosAutoconsumo.GetItemText(ElementoAuto, 2);
                EnTotalSalidaAuto = m_DatosAutoconsumo.GetItemText(ElementoAuto,
                3);
                ElementoAuto = ElementoAuto - 1;

                //cargar en archivo texto
                int DefinicionAuto = NombreSalidaAuto.GetLength() * sizeof(TCHAR);
                int EnergiaAuto = EUnitariaSalidaAuto.GetLength() * sizeof(TCHAR);
                int unidadesAuto = UnidSalidaAuto.GetLength() * sizeof(TCHAR);
                int TotalesAuto = EnTotalSalidaAuto.GetLength() * sizeof(TCHAR);

                GeneracionSalidaConsumos << "Definición:" << "
" << "Energía unitaria" << "      " << "Unidades" << "      " << "Total Energía"
<< "\n";
GeneracionSalidaConsumos.write((LPCSTR)(LPCTSTR)NombreSalidaAuto,
DefinicionAuto);
GeneracionSalidaConsumos << "           ";
GeneracionSalidaConsumos.write((LPCSTR)(LPCTSTR)EUnitariaSalidaAuto,
EnergiaAuto);
GeneracionSalidaConsumos << "           ";
GeneracionSalidaConsumos.write((LPCSTR)(LPCTSTR)UnidSalidaAuto, unidadesAuto);
GeneracionSalidaConsumos << "           ";
GeneracionSalidaConsumos.write((LPCSTR)(LPCTSTR)EnTotalSalidaAuto, TotalesAuto);
GeneracionSalidaConsumos << "\n";
GeneracionSalidaConsumos << "\n";
        }

GeneracionSalidaConsumos <<
"################################################################################
\n";
        GeneracionSalidaConsumos << "\n";

        // salidas de sumatorios generales
        GeneracionSalidaConsumos << "DEMANDA ENERGETICA TOTAL SISTEMA EN DISEÑO EN
        INSTALACION AISLADA DE RED\n";
        GeneracionSalidaConsumos << "\n";
        GeneracionSalidaConsumos << "Energía total punto de consumo:" << "  " <<
        m_TPunt << "\n";
        GeneracionSalidaConsumos << "Energía total punto de bombeo:" << "  " <<
        m_TBom << "\n";
        GeneracionSalidaConsumos << "Energía total punto de autoconsumo:" << "  "
        << m_TAuto << "\n";
        GeneracionSalidaConsumos << "Energía total punto de sistema:" << "  " <<
        m_SumTotal << "\n";

        // cierre de archico de salida
        GeneracionSalidaConsumos.close();
}
```

Se ha implementado un función de recuperación, que básicamente, accede al archivo temporal creado previamente cuando el usuario cierra la aplicación o cambia de vista, extrayendo o deserializando los datos contenidos en tal archivo, para poder darles paso a través dela función d inicialización de los controles de lista y mostrar en todo momento los datos de dichas listas, aun cuando el usuario los haya abandonado. Según esto los datos de estas listas se convertirán en permanentes.

La acción a realizar, es idéntica al proceso temporal de guardado de datos descrito mediante la función *OnBnClickedSigdiseno()*, pero en vez de archivando, extrayendo.

```
void Consumos::RecuperacionInfo()
{
	//realiza un bucle de desseriacion de los datos guardados para cada linea
	y tipo de salida de lista
	//hasta que se hayan recorrido todos los elementos guardados
	//lee archivo de datos guardados de la base de datos
	CString ListadosConsumos = RegistroAux + _T(".dat");
	CFile RecCons;
	RecCons.Open(ListadosConsumos, CFile::modeRead);
	CArchive ar(&RecCons, CArchive::load);

	//recuperación del número de ítems de cada lista
	ar >> ItemPuntual >> ItemBomba >> ItemAuto;

	ItemPuntual = ItemPuntual - 1; //resta uno para recuperar datos del valor
	0;
	ItemBomba = ItemBomba - 1;
	ItemAuto = ItemAuto - 1;

	while (ItemPuntual > -1)   // mientras haya objetos en la lista
	{
		//Recuperación de los valores de los elementos de lista en formato
		texto
		ar >> DefinicionPunt >> EnergPunt >> UDPunt >> EnerT;

		int nItem;
		nItem = m_DatosPuntuales.InsertItem(0, DefinicionPunt);

		m_DatosPuntuales.SetItemText(nItem, 0, DefinicionPunt);
		m_DatosPuntuales.SetItemText(nItem, 1, EnergPunt);
		m_DatosPuntuales.SetItemText(nItem, 2, UDPunt);
		m_DatosPuntuales.SetItemText(nItem, 3, EnerT);

		ItemPuntual = ItemPuntual - 1;
		UpdateData(FALSE);

	}

	while (ItemBomba > -1)   // mientras haya objetos en la lista
	{
		//Recuperación de los valores de los elementos de lista en formato
		texto
		ar >> DefBom >> EnerBom >> UDBom >> ETBom >> RendBom >> HdBom >>
		HstBom >> HdtBom >> HteBom >> HfBom >>QdBom >> QtBom >> QapBom;

		int nItem;
		nItem = m_DatosBombas.InsertItem(0, DefBom);

		m_DatosBombas.SetItemText(nItem, 0, DefBom);
		m_DatosBombas.SetItemText(nItem, 1, EnerBom);
		m_DatosBombas.SetItemText(nItem, 2, UDBom);
		m_DatosBombas.SetItemText(nItem, 3, ETBom);
		m_DatosBombas.SetItemText(nItem, 4, RendBom);
		m_DatosBombas.SetItemText(nItem, 5, HdBom);
		m_DatosBombas.SetItemText(nItem, 6, HstBom);
```

```
                m_DatosBombas.SetItemText(nItem, 7, HdtBom);
                m_DatosBombas.SetItemText(nItem, 8, HteBom);
                m_DatosBombas.SetItemText(nItem, 9, HfBom);
                m_DatosBombas.SetItemText(nItem, 10, QdBom);
                m_DatosBombas.SetItemText(nItem, 11, QtBom);
                m_DatosBombas.SetItemText(nItem, 12, QapBom);

                ItemBomba = ItemBomba - 1;
                UpdateData(FALSE);

        }

        while (ItemAuto > -1)   // mientras haya objetos en la lista
        {
                //Recuperación de los valores de los elementos de lista en formato
                texto
                ar >> DefinicionAuto >> EnergAuto >> UDAuto >> EnerTAuto;

                int nItem;
                nItem = m_DatosAutoconsumo.InsertItem(0, DefinicionAuto);

                m_DatosAutoconsumo.SetItemText(nItem, 0, DefinicionAuto);
                m_DatosAutoconsumo.SetItemText(nItem, 1, EnergAuto);
                m_DatosAutoconsumo.SetItemText(nItem, 2, UDAuto);
                m_DatosAutoconsumo.SetItemText(nItem, 3, EnerTAuto);

                ItemAuto = ItemAuto - 1;
                UpdateData(FALSE);
        }

        ar >> m_TPunt >> m_TBom >> m_TAuto >> m_SumTotal;

        //cierre seriación y archivo datos
        ar.Close();
        RecCons.Close();
        UpdateData(FALSE);

}
```

3.1.- Insercion consumos en controles de lista

Las siguientes funciones se insertan en las clases creadas para introducir información en los cuadros de lista de la primera pantalla de la aplicación, tal como se había dicho, al añadir un punto de consumo, en función de que clase es, se lanza un cuadro de diálogo para recibir la información desde teclado facilitada por el usuario.

Obviando los elementos genéricos de las clases y cabeceras de archivos, repetitivos y explicados ampliamente a lo largo del presente anexo, las funciones realizarán acciones de inserción de datos, y operaciones matemáticas asociadas a los cálculos básicos y previos con los que se operara sobre los datos disponibles.

```
// Puntual.cpp: archivo de implementación

void Puntual::OnBnClickedOk()
{
        // TODO: Agregue aquí su código de controlador de notificación de control
```

Se capturarán las variables de los cuadros de edición del cuadro y se operará con los valores, a fin de mostrar las operaciones en casillas o en su caso generar los valores que se pasaran directamente a la aplicación. En el presente caso únicamente el producto de la potencia de los elementos insertados, por el número de elemento de una categoría.

```
	UpdateData(TRUE);  //intecambio de datos de control a variable

	m_totalPuntual = m_unidadesPuntual*m_energiaPuntual;
```

Si el cuadro se ha dejado en blanco y se acepta, se avisará al usuario que no ha realizado inserción alguna mediante una caja de texto, en caso contrario si se cancela la función y no se efectuará acción alguna más allá de retornar al pantalla base.

```
	if (m_definicionPuntual.IsEmpty() == TRUE)
	{
		//indica esta vacio el registro de puntos de consumo y no se peude
		añadir
	MessageBox(_TEXT("No ha definido ningun punto de consumo"),
	_TEXT("Consumos Puntuales"), MB_ICONWARNING);


		UpdateData(FALSE);
	}

	if (m_definicionPuntual.IsEmpty() == FALSE)
	{
		CDialogEx::OnCancel(); // finalizar el cuadro dialogo introducir
		datos puntuales

	}

	//asigna valor a las variables externas de intercambio
	definicionPuntualAux=m_definicionPuntual;
	energiaPuntualAux=m_energiaPuntual;
	unidadesPuntualAux=m_unidadesPuntual;
	totalPuntualAux=m_totalPuntual;

	dato = true;
}

// bombas.cpp: archivo de implementación
// Controladores de mensajes de bombas

void bombas::OnBnClickedOk()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	UpdateData();

	if (m_DefinicionBomba.IsEmpty() == TRUE)

	{
		//indica esta vacio el registro de sistemas de bombeo y no se peude
		añadir
		MessageBox(_TEXT("No ha definido ningun sistema de bombeo"),
		_TEXT("Sistemas Bombeo"), MB_ICONWARNING);

		UpdateData(FALSE);
```

```
	}

	if (m_DefinicionBomba.IsEmpty() == FALSE)
	{
		CDialogEx::OnCancel(); // finalizar el cuadro dialogo introducir
		datos bombas
	}

	// operaciones matematicas del sistema de bombeo

	m_Qap = m_Qd / 24;  // calculo caudal medio aparente
	//calculo altura equivalente de bombeo
	m_Hte = m_Hd + m_Hst + (((m_Hdt - m_Hst) / m_Qt)*m_Qap) + m_Hf;
	//calculo energia electrcioa consumida por unidad de bombeo
	m_Emb = (2.725*m_Qd*m_Hte)/(m_Rendimiento/100);
	m_TotalBombas = m_UdBombas*m_Emb; //consumo total bombas de la smiam clase

	UpdateData();

	if (m_Hf>=0.1*m_Hte)
	{
		MessageBox(_TEXT("Perdidas por altura de fricción superiores al 10%
		de energia hifraulica util"), _TEXT("Sistemas Bombeo"),
		MB_ICONERROR);
		CDialogEx::OnCancel(); //Finalizar cuadro dialogo introducir datos
		bombas
	}

	//datos para pasar a pantalla consumos
	DefinicionBombaAux = m_DefinicionBomba;
	TotalBombasAux = m_TotalBombas;
	QdAux = m_Qd;
	RendimientoAux = m_Rendimiento;
	HdAux = m_Hd;
	HstAux = m_Hst;
	HdtAux = m_Hdt;
	QtAux = m_Qt;
	HfAux = m_Hf;
	EmbAux = m_Emb;
	HteAux = m_Hte;
	UdBombasAux = m_UdBombas;
	QapAux = m_Qap;

	dato = true;
}

void bombas::OnBnClickedCancel()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	CDialogEx::OnCancel();

	dato = false;
}

// Autoconsumo.cpp: archivo de implementación
// Controladores de mensajes de Autoconsumos
void Autoconsumos::OnBnClickedOk()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
```

```
	UpdateData();

	if (m_DefinicionAutoconsumo.IsEmpty() == TRUE)

	{
		//indica esta vacio el registro de sistemas de autoconsumo y no se
		peude añadir
		MessageBox(_TEXT("No ha definido ningun sistema con autoconsumo"),
		_TEXT("Sistemas Autoconsumo"), MB_ICONWARNING);

		UpdateData(FALSE);
	}

	if (m_DefinicionAutoconsumo.IsEmpty() == FALSE)
	{
		CDialogEx::OnCancel(); // finalizar el cuadro dialogo introducir d
	atos Autoconsumo
	}

	// operaciones matematicas del sistema de Autoconsumo
	m_TotalAutoconsumo = m_UdAutoconsumo*m_EnergiaAutoconsumo; //consumo total
	punto con autoconsumo de la misms clase

	UpdateData();

	//carga intercambio variables entre clases
	DefinicionAutoconsumoAux = m_DefinicionAutoconsumo;
	TotalAutoconsumoAux = m_TotalAutoconsumo;
	EnergiaAutoconsumoAux =m_EnergiaAutoconsumo;
	UdAutoconsumoAux = m_UdAutoconsumo;

	dato = true;

}

void Autoconsumos::OnBnClickedCancel()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	CDialogEx::OnCancel();
	dato = false;
}
```

3.2.- Dimensionado de generador

La clase de cálculo de generador de instalaciones aisladas, desarrolla las siguientes funciones en relación a la estructura desarrollada en su cuadro de dialogo especifico, y los elementos o componentes insertados en éste. De manera inicial se ejecutarán las funciones al igual que en cualquier otro archivo.

Se instala un botón, que mediante la función *ShellExecute*, lazan el navegador de internet por defectos del equipo informático del usuario, accediendo a GoogleMaps a fin de poder consultar coordenadas o latitud de orientación del lugar de instalación de los panales del proyecto.

```
void DGenerador::OnBnClickedBlatitud()
```

```
{
	// TODO: Agregue aquí su código de controlador de notificación de control


	ShellExecute(NULL, _T("open"), _T("https://www.google.es/maps/"), NULL,
	NULL, SW_SHOWNORMAL);

}
```

La función siguiente controla un cuadro desplegable, en el cual el usuario puede seleccionar una opción, para que en función del registro elegido, mediante elementos de selección "if", se carguen las variables de ángulos de inclinación y azimut especificados.

```
void DGenerador::OnCbnSelchangePeriodo()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	// añadir codigo cuadro desplegable para seleccinar el periodo de diseño
	//Cuando la seleción del cuadro cambia, se activa esta función

		m_Localidad;
		m_Provincia;
		m_Pais;
		m_Comentarios;
		m_Latitud;

		UpdateData(TRUE);

		if (m_SelecPeriodo == _T("Diciembre"))
		{
			m_AlfaOpt = 0;
			m_BetaOpt = m_Latitud + 10;
			m_K = 1.7;
		}

		if (m_SelecPeriodo == _T("Julio"))
		{
			m_AlfaOpt = 0;
			m_BetaOpt = m_Latitud - 20;
			m_K = 1;
		}

		if (m_SelecPeriodo == _T("Anual"))
		{
			m_AlfaOpt = 0;
			m_BetaOpt = m_Latitud - 10;
			m_K = 1.15;
		}

		UpdateData(FALSE);

}
```

En la presente aplicación se usará de manera recurrente, el evento *OnEnKillFocus*, mediante la cual, cuando el usuario este enfocando o utilizando un control determinado, véase un cuadro de edición para insertar algún dato, y cambien a otro elemento, perdiendo el foco del control previo, se lanzarán las acciones

correspondientes a la función implementada. En la función actual, esto se utilizara para capturar los elementos de los cuadros de edición previos, y efectuar las operaciones matemáticas correspondientes al cálculo del factor de irradiación.

```
void DGenerador::OnEnKillfocusGdm()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	// Cuando se añade G(0), se cargan variables no añadidas hasta el momento
	y se ejecuta matematica para salida de resto de casillas de apartado
	perdidas
	// se activa cunado se deja de tener enfocado el registro

	m_AlfaDiseno;
	m_BetaDiseno;
	m_Gdm0;
	UpdateData(TRUE);

	BetaDisenoAux = m_BetaDiseno;
	// salida FI
	double B = pow((m_BetaDiseno - m_BetaOpt), 2);
	const double Z = 1.2;
	const double X = pow(10.0, -4);
	double A = pow(m_AlfaDiseno, 2);
	const double C = pow(10.0, -5);
```

En función del ángulo de inclinación de diseño, se impondrá un factor de irradiación u otro, atendiendo a las formulas correspondientes en función del pliego técnico correspondiente (ver base teórica de memoria de proyecto).

```
if ((15 < m_BetaDiseno) && (m_BetaDiseno < 90))
	{
		m_FIrradiacion = 1 - ((Z*X*B) + 3.5*C*A);
	}

	else if (m_BetaDiseno <= 15)
	{
		m_FIrradiacion = 1 - (Z*X*B);
	}

	//salida perdidas Irradiacion
	m_PerdidasRad = 1 - m_FIrradiacion;

	//salida Gdm opt
	m_GdmMax = m_K * m_Gdm0;

	UpdateData(FALSE);
}
```

La función siguiente, captura el factor de sombreo introducido manualmente por el usuario en un cuadro de edición, realizando las operaciones matemáticas para cálculo el GDM de diseño relacionado con un valor adimensional K (impuesto en función de las condiciones de diseños seleccionadas en cuadro desplegable inicial, y los factores de irradiación y sombreo calculados, o impuestos.

```
void DGenerador::OnEnKillfocusFs()
{
```

```
        // TODO: Agregue aquí su código de controlador de notificación de control
        // registros casillas tras eliminar el foco de seleccionar Sombreo

        m_FSombreo;
        UpdateData(TRUE);

        // salida perdidas sombreo
        m_PerdidasSom = 1 - m_FSombreo;

        //salida de Gdm de diseño segun angulos elegidos
        m_GdmDis = m_Gdm0*m_K*m_FIrradiacion*m_FSombreo;
        GAux = m_GdmDis;
        UpdateData(FALSE);
}
```

Si el usuario actúa sobre el botón de paso a la siguiente pantalla del desarrollo del proyecto, todas las variables registradas en la página actual se guardaran (para poder guardarla en disco duro o generar informes de salida), lanzándose posteriormente el cuadro de dialogo modal de la siguiente pantalla.

```
void DGenerador::OnBnClickedSigdgen()
{
        // TODO: Agregue aquí su código de controlador de notificación de control
        //carga variables generar informes salida
        LocalidadInfosalida = m_Localidad, ProvinciaInfosalida = m_Provincia,
        PaisInfosalida = m_Pais, ComentariosInfosalida = m_Comentarios;
        LatitudInfosalida = m_Latitud;
        AlfaOptInfosalida = m_AlfaOpt;
        BetaOptInfosalida = m_BetaOpt, KInfosalida = m_K;
        SelecPeriodoInfosalida = m_SelecPeriodo;
        AlfaDisenoInfosalida = m_AlfaDiseno, BetaDisenoInfosalida = m_BetaDiseno,
        Gdm0Infosalida = m_Gdm0;
        FIrradiacionInfosalida = m_FIrradiacion, FSombreoInfosalida = m_FSombreo;
        GdmMaxInfosalida = m_GdmMax, PerdidasRadInfosalida = m_PerdidasRad,
        PerdidasSomInfosalida = m_PerdidasSom;
        GdmDisInfosalida = m_GdmDis;

        DisSistema DlgDisSistema(NULL); //Llama al constructor del dialogo para
        lanzar la pantalla

        if (DlgDisSistema.DoModal() == IDOK)
        {

        }
}
```

Al igual que en el caso de seguir adelante a la siguiente pantalla, el usuario puede optar por dar marcha atrás en el proceso de diseño y volver a pantallas previas, siendo que en este caso se guardara la información introducida, para que esté disponible en el retorno , y no sea necesario volver a introducirla.

```
void DGenerador::OnBnClickedAnteriordgen()
{
        // TODO: Agregue aquí su código de controlador de notificación de control
        UpdateData(TRUE);

        LocalidadRetorno=m_Localidad, ProvinciaRetorno = m_Provincia, PaisRetorno
        = m_Pais,ComentariosRetorno =m_Comentarios;
```

```
	LatitudRetorno = m_Latitud;
	AlfaOptRetorno = m_AlfaOpt;
	BetaOptRetorno = m_BetaOpt, KRetorno = m_K;
	SelecPeriodoRetorno = m_SelecPeriodo;
	AlfaDisenoRetorno = m_AlfaDiseno, BetaDisenoRetorno = m_BetaDiseno,
	Gdm0Retorno = m_Gdm0;
	FIrradiacionRetorno = m_FIrradiacion, FSombreoRetorno = m_FSombreo;
	GdmMaxRetorno = m_GdmMax, PerdidasRadRetorno = m_PerdidasRad,
	PerdidasSomRetorno = m_PerdidasSom;
	GdmDisRetorno = m_GdmDis;

	VueltaDGenerador = true;

	//genera un archivo de salida en blanco
	ofstream GeneracionSalidaConsumos;
	GeneracionSalidaConsumos.open("SalidaConsumosAisladaRed.txt", ios::out);
	GeneracionSalidaConsumos.close();

	EndDialog(NULL);
}
```

Previamente se ha visto la zona de la aplicación o código, donde se captura el factor de sombreo introducido manualmente por el usuario, pero existe instalado una herramienta tipo botón, mediante la cual se puede lanzar la automatización del cálculo de las perdidas por sombreo (explicado en puntos posteriores), según lo cual se abriría un cuadro de diálogo especifico a tal uso.

```
void DGenerador::OnBnClickedPersombras()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	// acceso a la plantilla de calculo de perdidas por sombras

	CalcPerdSombra DlgCalcPerdSombra(NULL); //Llama al constructor del dialogo
	para lanzar la pantalla

	if (DlgCalcPerdSombra.DoModal() == IDOK)
	{
```

El retorno del cuadro de diálogo de cálculo de pérdida de sombreo, cargaría en el cuadro de edición de la presente pantalla, el valor generado en dicha sección del programa, a fin de que se muestre, y efectúen cuenta con él, al igual que si hubiese sido insertado manualmente.

```
		//cuando se cierra dialogo con Ok, establece variable para sacarla
		por pantalla
		m_PerdidasSom = PerdSombrasAux/100;
		m_FSombreo = 1 - m_PerdidasSom;
		UpdateData(FALSE);
```

Se produce un salto a la función especificada, para que las operaciones matemáticas se realicen una vez se haya calculado automáticamente el factor de sombreo. Dichas operaciones no se duplican, sino que únicamente son declaradas en una de ellas, siendo así manipuladas, bien directamente cuando se recorre el código de tal función, o bien mediante el presente salto al usarse un cálculo de pérdidas automatizado.

```
                OnEnKillfocusFs();
        }
}
```

En el cuadro de dialogo, cuando se inicia, se hace una llamada por defecto a la función *OnInitDialog()*, siendo que se impone una condición a cumplir, el análisis o verificación, si se llama al cuadro de diálogo después de haber abierto un proyecto existente en disco (deserializado), y si se entra en el cuadro no por primera vez, sino en retorno tras haber retrocedido pantallas en la aplicación, en cuyo caso, se cargaran los datos disponibles y anteriormente introducidos por el usuario, en los diversos apartado de la interface.

```
BOOL DGenerador::OnInitDialog()
{
        CDialogEx::OnInitDialog();

        // TODO:  Agregue aquí la inicialización adicional
        if (VueltaDGenerador | AbiertoAachivo)
        {

                m_Localidad=LocalidadRetorno,  m_Provincia=ProvinciaRetorno,
                m_Pais=PaisRetorno,  m_Comentarios=ComentariosRetorno;
                m_Latitud=LatitudRetorno;
                m_AlfaOpt=AlfaOptRetorno;
                m_BetaOpt=BetaOptRetorno,  m_K=KRetorno;
                m_SelecPeriodo=SelecPeriodoRetorno;
                m_AlfaDiseno=AlfaDisenoRetorno,   m_BetaDiseno=BetaDisenoRetorno,
                m_Gdm0=Gdm0Retorno;
                m_FIrradiacion=FIrradiacionRetorno,   m_FSombreo=FSombreoRetorno;
                m_GdmMax=GdmMaxRetorno,   m_PerdidasRad=PerdidasRadRetorno,
                m_PerdidasSom=PerdidasSomRetorno;
                m_GdmDis=GdmDisRetorno;

               VueltaDGenerador = false;
               UpdateData(FALSE);
        }

        return TRUE;  // return TRUE unless you set the focus to a control
        // EXCEPCIÓN: las páginas de propiedades OCX deben devolver FALSE
}
```

3.3.- Diseño sistema

Las funciones implementadas en este archivo, gestionarán el cálculo del generador fotovoltaico a estudio, así como el cumplimiento o adaptación del mismo a la norma o pliego técnico. Sin entrar a analizar elementos comunes y generales, se explicarán las diferentes funciones insertadas.

Se utilizan funciones para control de cuadros de lista, siendo una función específica en la cual, cuando se cambia la selección que está marcada en el cuadro (*OnCbnSelchangePR()*), se entra en un comparador "if", asignándose valor a un variable miembro en función de la selección que ha realizado el usuario.

```
void DisSistema::OnCbnSelchangePr()
```

```
{

	UpdateData(TRUE);

	if (m_PR == _T("Inversor"))
	{
		m_RendEnerg = 0.7;
	}

	if (m_PR == _T("Inv/bateria"))
	{
		m_RendEnerg = 0.6;
	}

	if (m_PR == _T("Otros"))
	{
		MessageBox(_TEXT("Funcion no definida; Debe justificar PR"),
		_TEXT("Rendimiento Energetico de la Instalación"),
		MB_ICONINFORMATION);
	}
```

Así mismo además de asignar valor, se realizan las operaciones necesarias para mostrar o calcular los elementos de los cuadros de edición o interface activados hasta ese punto de la vista, capturando sede variables globales variables necesarias de pantallas anteriores, como los consumos del sistema.

```
	m_SumTotal = SumaTotalAux;  // se intercambian estos datos de otros
	modulos
	m_GdmDis = GAux;     // se intercambian estos datos de otros modulos

	double m_SumTotalAux = m_SumTotal / 1000;
	m_Pmp = (m_SumTotalAux*Gcem) / (m_GdmDis*m_RendEnerg);

	m_PMax = m_Pmp*1.2;
	UpdateData(FALSE);

}
```

Como en el resto de secciones, en algún punto, al abandonarse una de las casillas de introducción de datos, se lanza una función, que captura datos, los opera y en su caso ofrecer los avisos necesarios. En la presente función se calculan los datos del generador en diseño, y se verifica que su potencia total se ajusta a los requisitos de potencia máxima.

```
void DisSistema::OnEnKillfocusNpaneles()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	m_ICorto;
	m_IPMax;
	m_VAbierto;
	m_VPMax;
	m_PotMaxPanel;
	m_NPaneles;
	UpdateData(TRUE);

	m_PNominal = (m_PotMaxPanel*m_NPaneles)/1000;
```

```
	UpdateData(FALSE);

	if (m_PNominal > m_PMax)
	{
		MessageBox(_TEXT("ERROR: Potencia nominal generador superior a
		máximo permitido"), _TEXT("Maxima potencia nominal"),
		MB_ICONERROR);
	}
}
```

De igual manera al generador, se operara con los datos disponibles de los acumuladores, debiéndose cumplir los requisitos específicos del pliego técnico correspondiente, y calculándose estos en función de la base teórica del mismo.

```
void DisSistema::OnEnKillfocusProfdescarg()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	m_C20;
	m_VnominalSis;
	m_ProfDescarga;

	UpdateData(TRUE);

	m_ConsDiario = m_SumTotal/m_VnominalSis;
	m_Isc = m_C20 / 20.28;

	UpdateData(FALSE);

	if (m_Isc >= 25)
	{
		MessageBox(_TEXT("ERROR: Valos C20/Isc debe ser < 25"),
		_TEXT("Requisito obligatorio"), MB_ICONERROR);
	}

}
```

Las siguientes funciones, realizarán la selección del rendimiento del inversor, cargado en las funciones miembro correspondientes, en función de una serie de opciones que pueda gestionar el usuario. Según esto se impone funcionamiento de botones de elección, y se encadenan con cuadro de lista desplegable (no se explican dado que ambos tipos de controles se han comentado en memoria), y en función de las comparativas de la selección realizadas por el usuario, se asigna un valor de rendimiento de inversor, a la variable de salida que se mostrará en la interface de usuario.

```
void DisSistema::OnBnClickedRadio1()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	m_TipoInv;
	m_Rend20 = true;
	m_RendNom = false;

	UpdateData(TRUE);

	if (m_TipoInv == _T("Onda senoidal Pnom <= 500VA"))
	{
		RendInversorAux = 85;
```

```
	}

	if (m_TipoInv == _T("Onda senoidal Pnom>500VA"))
	{
		RendInversorAux = 90;

	}

	if (m_TipoInv == _T("Onda no senoidal"))
	{
		RendInversorAux = 90;
	}
}

void DisSistema::OnBnClickedRadio2()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	m_TipoInv;
	m_Rend20 = false;
	m_RendNom = true;

	UpdateData(TRUE);

	CString UU = m_TipoInv;
	if (m_TipoInv == _T("Onda senoidal Pnom<=500VA"))
	{
		RendInversorAux = 75;
	}

	if (m_TipoInv == _T("Onda senoidal Pnom>500VA"))
	{
		RendInversorAux = 85;

	}

	if (m_TipoInv == _T("Onda no senoidal"))
	{
		RendInversorAux = 85;
	}
}


void DisSistema::OnBnClickedAjuste()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (OpcionNormaAux == false)

	{

		if ((m_Rend20 == true) && (m_RendNom == false))
		{

			if (m_TipoInv == _T("Onda senoidal Pnom<=500VA"))
			{
				m_RendInversor = 85;
			}

			if (m_TipoInv == _T("Onda senoidal Pnom>500VA"))
```

```
                        {
                                m_RendInversor = 90;

                        }

                        if (m_TipoInv == _T("Onda no senoidal"))
                        {
                                m_RendInversor = 90;
                        }

                        UpdateData(FALSE);
                }

                if ((m_Rend20 == false) && (m_RendNom == true))
                {

                        if (m_TipoInv == _T("Onda senoidal Pnom<=500VA"))
                        {
                                m_RendInversor = 75;
                        }

                        if (m_TipoInv == _T("Onda senoidal Pnom>500VA"))
                        {
                                m_RendInversor = 85;

                        }

                        if (m_TipoInv == _T("Onda no senoidal"))
                        {
                                m_RendInversor = 85;
                        }

                        UpdateData(FALSE);
                }

                if ((m_Rend20 == false) && (m_RendNom == false))

                {
                        MessageBox(_TEXT("Debe seleccinar % de rendimiento
                        inversor"), _TEXT("Rendimiento Inversor"),
                        MB_ICONEXCLAMATION);

                }

                GetDlgItem(IDC_RENDINVERSOR)->SendMessage(EM_SETREADONLY, 1, 0);
                //cambia el estado de lectura de casilla rendimiento inversor
                OpcionNormaAux = TRUE;
                DisSistema::OnEnKillfocusRendinversor();
        }

        else if (OpcionNormaAux == true)
        {
                GetDlgItem(IDC_RENDINVERSOR)->SendMessage(EM_SETREADONLY, 0, 0);
                OpcionNormaAux = FALSE;
                DisSistema::OnEnKillfocusRendinversor();  // si no esta
                seleccionada casilla salta directametne
                //a comparacion con numero introducido por usuario
        }
}
```

```
void DisSistema::OnEnKillfocusRendinversor()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	m_RendInversor;
	UpdateData(TRUE);

	if ((m_Rend20 == true) && (m_RendNom == false))
	{

		if (m_TipoInv == _T("Onda senoidal Pnom<=500VA"))
		{
			if (m_RendInversor < RendInversorAux)
			{
				MessageBox(_TEXT("ERROR: Rendimiento inversor inferior
				a estipilado en norma"), _TEXT("Rendimiento
				Inversor"), MB_ICONERROR);
			}
		}

		if (m_TipoInv == _T("Onda senoidal Pnom>500VA"))
		{
			if (m_RendInversor < RendInversorAux)
			{
				MessageBox(_TEXT("ERROR: Rendimiento inversor inferior
				a estipilado en norma"), _TEXT("Rendimiento
				Inversor"), MB_ICONERROR);
			}

		}

		if (m_TipoInv == _T("Onda no senoidal"))
		{
			if (m_RendInversor < RendInversorAux)
			{
				MessageBox(_TEXT("ERROR: Rendimiento inversor inferior
				a estipilado en norma"), _TEXT("Rendimiento
				Inversor"), MB_ICONERROR);
			}
		}

	}

	if ((m_Rend20 == false) && (m_RendNom == true))
	{

		if (m_TipoInv == _T("Onda senoidal Pnom<=500VA"))
		{
			if (m_RendInversor < RendInversorAux)
			{
				MessageBox(_TEXT("ERROR: Rendimiento inversor inferior
				a estipilado en norma"), _TEXT("Rendimiento
				Inversor"), MB_ICONERROR);
			}
		}

		if (m_TipoInv == _T("Onda senoidal Pnom>500VA"))
		{
			if (m_RendInversor < RendInversorAux)
			{
```

```
                        MessageBox(_TEXT("ERROR: Rendimiento inversor inferior
                        a estipilado en norma"), _TEXT("Rendimiento
                        Inversor"), MB_ICONERROR);
                }

        }

        if (m_TipoInv == _T("Onda no senoidal"))
        {
                if (m_RendInversor < RendInversorAux)
                {
                        MessageBox(_TEXT("ERROR: Rendimiento inversor inferior
                        a estipilado en norma"), _TEXT("Rendimiento
                        Inversor"), MB_ICONERROR);
                }
        }

    }

    if ((m_Rend20 == false) && (m_RendNom == false))

    {
            MessageBox(_TEXT("Debe seleccinar % de rendimiento inversor"),
            _TEXT("Rendimiento Inversor"), MB_ICONEXCLAMATION);

    }

    UpdateData(FALSE);
}
```

Al igual que en la función específica del cálculo del generador, una vez seleccionados todos los parámetros y atendiendo a las ecuaciones correspondientes, se calculará la autonomía de las baterías y se informará al usuario, mediante un cuadro de texto emergente, si se está dentro del rango mínimo especificado y necesario.

```
void DisSistema::OnEnKillfocusRendbat()
{
    // TODO: Agregue aquí su código de controlador de notificación de control

    m_RendAcum;
    UpdateData(TRUE);

    m_Autonomia = ((m_C20*(m_ProfDescarga/100)) /
    m_ConsDiario)*(m_RendInversor/100)*(m_RendAcum/100);

    UpdateData(FALSE);

    if (m_Autonomia < 3)
    {
            MessageBox(_TEXT("ERROR: Autonomia acumulación por debajo del
            mínimo requerido"), _TEXT("Autonomía Sistema Acumulación"),
            MB_ICONERROR);
    }
}
```

Si el usuario actúa sobre el botón de paso a la siguiente pantalla del desarrollo del proyecto, todas las variables registradas en la página actual se guardarán (para poder

guardarla en disco duro o generar informes de salida), lanzándose posteriormente el cuadro de dialogo modal de la siguiente pantalla.

```
void DisSistema::OnBnClickedSigdsis()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	//carga variables para salida de informes
	PRInfosalida = m_PR;
	RendEnergInfosalida = m_RendEnerg, PmpInfosalida = m_Pmp, ICortoInfosalida
	= m_ICorto, IPMaxInfosalida = m_IPMax,VAbiertoInfosalida = m_VAbierto,
	VPMaxInfosalida = m_VPMax, PotMaxPanelInfosalida = m_PotMaxPanel;
	NPanelesInfosalida = m_NPaneles, PNominalInfosalida = m_PNominal,
	PMaxInfosalida = m_PMax;GdmDisAuxInfosalida = m_GdmDis, C20Infosalida =
	m_C20, VnominalSisInfosalida = m_VnominalSis,ProfDescargaInfosalida =
	m_ProfDescarga, ConsDiarioInfosalida = m_ConsDiario, IscInfosalida =m_Isc;
	TipoInvInfosalida = m_TipoInv, Rend20Infosalida = m_Rend20,
	RendNomInfosalida = m_RendNom;
	RendInversorInfosalida = m_RendInversor, OpcionNormaInfosalida =
	m_OpcionNorma, RendAcumInfosalida = m_RendAcum,
	AutonomiaInfosalida = m_Autonomia, SumTotalInfosalida = m_SumTotal,
	GdmDisInfosalida = m_GdmDis;
```

En la llamada al siguiente cuadro de diálogo (que será el destinado al cálculo de cargas sobre cubierta), antes de lanzarlo de manera directa, se mostrara otro cuadro de dialogo modal, con una elección para el usuario, correspondiendo esta al tipo de estructura, si es fija para el cálculo de la cargas en cubierta, y si es tipo seguidor, en cuadro caso y al no implementarse este cálculo, en lugar del saltar a la pantalla de cálculo de cargas, se saltara la misma (al nos e necesaria) y se pasará a la pantalla posterior de cálculo de inversiones del proyecto.

```
	//Lanzar selector tipo estructura de soporte de paneles
	Estructura DlgEstrcutura(NULL);   //Llama al constructor del dialogo para
	lanzar la pantalla

	if (DlgEstrcutura.DoModal() == IDOK)

	{

	}

	if (TipoEstruct == 0)
	{
		// lanzar cuadro de dialogo de cargas estructura
		Cargas DlgCargas(NULL);   //Llama al constructor del dialogo para
		lanzar la pantalla

		if (DlgCargas.DoModal() == IDOK)

		{

		}
	}
	else if (TipoEstruct == 1)
	{
		//Lanzar aviso de que el caluclo del segurio deber realziarse
		independinte
		//Pasar a sigueinte pantalla saltandose la de la estructura
```

```
                MessageBox(_TEXT("Ha seleccionado estructura tipo seguidor.El
                cálculo de cargas sobre este elemento debe calcularse de manera
                independiente"),_TEXT("Cálculo cargas en estructura de paneles"),
                MB_ICONINFORMATION);

                Inversion DlgInversion(NULL); //Llama al constructor del dialogo
                para lanzar la pantalla

                if (DlgInversion.DoModal() == IDOK)
                {

                }
        }

}
```

Al igual que en el caso de seguir adelante a la siguiente pantalla, el usuario puede optar por dar marcha atrás en el proceso de diseño y volver a pantallas previas, siendo que en este caso se guardará la información introducida, para que esté disponible en el retorno , y no sea necesario volver a introducirla.

```
void DisSistema::OnBnClickedAnteriordsis()
{
        // TODO: Agregue aquí su código de controlador de notificación de control
        UpdateData(TRUE);

        PRRetorno=m_PR;
        RendEnergRetorno=m_RendEnerg,PmpRetorno=m_Pmp, ICortoRetorno=m_ICorto,
        IPMaxRetorno=m_IPMax, VAbiertoRetorno=m_VAbierto, VPMaxRetorno=m_VPMax,
        PotMaxPanelRetorno=m_PotMaxPanel;
        NPanelesRetorno=m_NPaneles,PNominalRetorno=m_PNominal, PMaxRetorno=m_PMax;
        GdmDisAuxRetorno = m_GdmDis, C20Retorno =m_C20,
        VnominalSisRetorno=m_VnominalSis,ProfDescargaRetorno = m_ProfDescarga,
        ConsDiarioRetorno=m_ConsDiario, IscRetorno=m_Isc;
        TipoInvRetorno=m_TipoInv,Rend20Retorno=m_Rend20, RendNomRetorno =
        m_RendNom;
        RendInversorRetorno=m_RendInversor,OpcionNormaRetorno =
        m_OpcionNorma,RendAcumRetorno=m_RendAcum, AutonomiaRetorno=m_Autonomia,
        SumTotalRetorno = m_SumTotal, GdmDisRetorno = m_GdmDis;

        VueltaDisSistema = true;
        EndDialog(NULL);
}
```

En el cuadro de dialogo, cuando se inicia, se hace una llamada por defecto a la función OnInitDialog(), siendo que se impone una condiciona cumplir, el análisis o verificación, si se llama al cuadro de diálogo después de haber abierto un proyecto existente en disco (deserializado), y si se entra en el cuadro no por primera vez, sino en retorno tras haber retrocedido pantallas en la aplicación, en cuyo caso, se cargaran los datos disponibles y anteriormente introducidos por el usuario, en los diversos apartado de la interface.

```
BOOL DisSistema::OnInitDialog()
{
        CDialogEx::OnInitDialog();

        // TODO:  Agregue aquí la inicialización adicional
```

```
	if (VueltaDisSistema|AbiertoAachivo)
	{

		m_PR=PRRetorno;
		m_RendEnerg=RendEnergRetorno,   m_Pmp=PmpRetorno,
		m_ICorto=ICortoRetorno,   m_IPMax=IPMaxRetorno,
		m_VAbierto=VAbiertoRetorno,  m_VPMax=VPMaxRetorno,
		m_PotMaxPanel=PotMaxPanelRetorno;
		m_NPaneles=NPanelesRetorno,  m_PNominal=PNominalRetorno,
		m_PMax=PMaxRetorno;
		m_GdmDis=GdmDisAuxRetorno,   m_C20=C20Retorno,
		m_VnominalSis=VnominalSisRetorno,
		m_ProfDescarga=ProfDescargaRetorno,
		m_ConsDiario=ConsDiarioRetorno,   m_Isc=IscRetorno;
		m_TipoInv=TipoInvRetorno,   m_Rend20=Rend20Retorno,
		m_RendNom=RendNomRetorno;
		m_RendInversor=RendInversorRetorno,
		m_OpcionNorma=OpcionNormaRetorno,  m_RendAcum=RendAcumRetorno,
		m_Autonomia=AutonomiaRetorno,  m_SumTotal=SumTotalRetorno,
		m_GdmDis=GdmDisRetorno;

		VueltaDisSistema = false;
		UpdateData(FALSE);
	}

	return TRUE;  // return TRUE unless you set the focus to a control
	// EXCEPCIÓN: las páginas de propiedades OCX deben devolver FALSE
}
```

3.4.- Cargas cubierta

En el presente archivo, se calculan las sobrecargas de cubierta, o iteraciones para con agentes atmosféricos, soportados por instalaciones fotovoltaicas para producción, con estructura fija. Tal tipo de cálculo, será similar y asociado a todos los tipos de instalaciones programadas en el presente sistema, sean estas para autoconsumo, o para producción energética destinada a venta.

Mediante el uso de cuadro desplegable, se seleccionará la zona de ubicación de la instalación, asignándose a una variable miembro el valor matemático asociado a tal zona.

```
void Cargas::OnCbnSelchangeSelzona()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	//Seleccion zona con menu desplegable
	UpdateData(TRUE);

	if (m_Zona == _T("Zona A"))
	{
		m_PresDinamica = 0.42;
	}

	if (m_Zona == _T("Zona B"))
	{
		m_PresDinamica = 0.45;
	}
```

```
	if (m_Zona == _T("Zona C"))
	{
		m_PresDinamica = 0.52;
	}

	m_PresDin = m_PresDinamica;
	UpdateData(FALSE);
}
```

Para consultar la zona de ubicación de la instalación, según mapas distribución de zonas, se crea una clase específica, mediante la cual se implementa un control de imagen (explicado en memoria), siendo que al pulsar el botón de visualización, se lanzará el cuadro de diálogo que lo muestra.

```
void Cargas::OnBnClickedZonas()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	//desplegar grafico para ver que zona greogafica corresponde

	ZonasVientoGraf DlgZonasVientoGraf(NULL); //Llama al constructor del
	dialogo para lanzar la pantalla

	if (DlgZonasVientoGraf.DoModal() == IDOK)
	{

	}
}
```

De manera análoga a la zona geográfica, se seleccionara la aspereza del terreno mediante cuadro desplegable, cargándose los valores específicos a la variables de cálculo, en función de la selección realizada por el usuario.

```
void Cargas::OnCbnSelchangeSelaspereza()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	//Selección rugosidad del terreno
	UpdateData(TRUE);
	m_AlturaInstal;
	m_PendienteP = BetaDisenoAux;

	if (m_Aspereza == _T("I Borde mar o lago con sup. agua min 5Km"))
	{
		m_K = 0.15;
		m_L = 0.003;
		m_Z = 1.00;
	}
	if (m_Aspereza == _T("II Rural llano sin obstáculos ni arbolado"))
	{
		m_K = 0.17;
		m_L = 0.01;
		m_Z = 1.00;
	}
	if (m_Aspereza == _T("III Rural accidentado con algún obstáculo"))
	{
		m_K = 0.19;
		m_L = 0.05;
		m_Z = 2.00;
```

```
	}
	if (m_Aspereza == _T("IV Zona urbana, industrial o forestal"))
	{
		m_K = 0.22;
		m_L = 0.3;
		m_Z = 5.00;
	}
	if (m_Aspereza == _T("V Centro ciudades con edificios en altura"))
	{
		m_K = 0.24;
		m_L = 1.00;
		m_Z = 10.00;
	}
```

Se calculará el coeficiente de exposición eólico en función de los datos cargados de manera previa, siendo que los resultados, se empleaáan a lo largo del archivo de código para los cálculos correspondientes.

```
	//calculos matematicos de coeficiente exposicion
	double max = max(m_AlturaInstal, m_Z);
	double F = m_K*log(max / m_L);
	m_CExp = F*(F + (7 * m_K));

	UpdateData(FALSE);
}
```

Al igual que con el anteriormente referido grafico de zonas eólicas, se implementa una clase análoga para visualizar el grafico del tipo de coeficiente eólico a seleccionar.

```
void Cargas::OnBnClickedCoefgraf()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	//lanza graficos de coeficinete eolico para revisar y elegir

	CoefEolicoGraf DlgCoefEolicoGraf(NULL); //Llama al constructor del dialogo
para lanzar la pantalla

	if (DlgCoefEolicoGraf.DoModal() == IDOK)
	{

	}
}
```

Seleccionado el coeficiente eólico, y pasado a siguiente cuadro de edición, se opera con todos los valores disponibles, para calcular la acción del viento sobre la estructura en función de las ecuaciones ofrecidas por el código técnico aplicable al uso.

```
void Cargas::OnEnKillfocusCeolico()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	//funcion una vez se mete dato en casilla coeficiente eolico y se pasa a
la siguiente

	UpdateData(TRUE);
	m_CoefEolico;
```

```
	m_AccViento = (m_PresDin*m_CExp*m_CoefEolico)*1000/9.81;
	UpdateData(FALSE);
}
```

En función del número de paneles por fila y sus dimensiones, se calcula la fuerza derivada de la acción del viento sobre estos. También se calcula al ir pasando por la interface por las diferentes casilla su opciones, el peso línea de panel (considerado en montaje).

```
void Cargas::OnEnKillfocusPfila()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	UpdateData(TRUE);
	m_Largo;
	m_Ancho;
	m_PFila;

	m_Fuerza = m_AccViento*m_PFila*m_Largo;
	UpdateData(FALSE);
}


void Cargas::OnEnKillfocusPesopanel()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	UpdateData(TRUE);
	m_PesoPanel;
	m_PesoPanelesMl = m_PesoPanel/m_Ancho;
	UpdateData(FALSE);
}
```

Se calcula así mismo en función de las longitudes de los perfiles de la estructura de montaje estándar, el peso del material de las estructuras, en función de su desarrollo lineal y volumen, siendo la sobrecarga sobre cubierta el peso de la estructura más el del panel.

```
void Cargas::OnEnKillfocusEspperfil()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	UpdateData(TRUE);
	m_Densidad;
	m_DesarrolloPerfil;
	m_EspesorPerfil;

	// longitud de perfiles en funcion de orientación panel (si base = ancho 2
	largeros,
	//si base = altura 4 largeros)
	if (m_Ancho <= 1.00)
	{
		longitud = 4 * m_Ancho + 2 * m_Largo;
	}
	else if (m_Ancho > 1.00)
	{
		longitud = 4 * m_Ancho + 4 * m_Largo;
	}

	m_VolumenTotal = longitud*m_DesarrolloPerfil*m_EspesorPerfil;
	PesoPanelesSeccion = 2 * m_PesoPanel*m_PFila;
```

```
	SuperficieCalculo = 2 * m_Ancho*m_Largo*m_PFila;
	m_PesoTotal = (m_VolumenTotal*m_Densidad*m_PFila)/SuperficieCalculo;
	m_Sobrecarga = (PesoPanelesSeccion + m_PesoTotal) / SuperficieCalculo;

	UpdateData(FALSE);
}
```

Si el usuario actúa sobre el botón de paso a la siguiente pantalla del desarrollo del proyecto, todas las variables registradas en la página actual se guardaran (para poder guardarla en disco duro o generar informes de salida), lanzándose posteriormente el cuadro de dialogo modal de la siguiente pantalla.

```
void Cargas::OnBnClickedSigcargas()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	//carga variables para salida informe
	ZonaInfosalida = m_Zona, AsperezaInfosalida = m_Aspereza;
	PresDinamicaInfosalida = m_PresDinamica, AlturaInstalInfosalida =
	m_AlturaInstal, KInfosalida = m_K, LInfosalida = m_L,
	ZInfosalida = m_Z, PendientePInfosalida = m_PendienteP,
	CoefEolicoInfosalida = m_CoefEolico, PresDinInfosalida = m_PresDin,
	CExpInfosalida = m_CExp, AccVientoInfosalida = m_AccViento,
	LargoInfosalida = m_Largo, AnchoInfosalida = m_Ancho;
	PFilaInfosalida = m_PFila;
	FuerzaInfosalida = m_Fuerza, PesoPanelInfosalida = m_PesoPanel,
	PesoPanelesMlInfosalida = m_PesoPanelesMl,
	DensidadInfosalida = m_Densidad, DesarrolloPerfilInfosalida =
	m_DesarrolloPerfil, EspesorPerfilInfosalida = m_EspesorPerfil,
	VolumenTotalInfosalida = m_VolumenTotal, PesoTotalInfosalida =
	m_PesoTotal, SobrecargaInfosalida = m_Sobrecarga;

	Inversion DlgInversion(NULL); //Llama al constructor del dialogo para
	lanzar la pantalla

	if (DlgInversion.DoModal() == IDOK)
	{

	}
}
```

Al igual que en el caso de seguir adelante a la siguiente pantalla, el usuario puede optar por dar marcha atrás en el proceso de diseño y volver a pantallas previas, siendo que en este caso se guardara la información introducida, para que esté disponible en el retorno , y no sea necesario volver a introducirla.

```
void Cargas::OnBnClickedAnteriorcargas()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	UpdateData(TRUE);

	ZonaRetorno=m_Zona, AsperezaRetorno=m_Aspereza;
	PresDinamicaRetorno=m_PresDinamica, AlturaInstalRetorno=m_AlturaInstal,
	KRetorno=m_K, LRetorno=m_L,
	ZRetorno=m_Z,
	PendientePRetorno=m_PendienteP,CoefEolicoRetorno=m_CoefEolico,
	PresDinRetorno=m_PresDin,
	CExpRetorno=m_CExp, AccVientoRetorno=m_AccViento,LargoRetorno = m_Largo,
	AnchoRetorno = m_Ancho;
```

```
        PFilaRetorno=m_PFila;
        FuerzaRetorno=m_Fuerza, PesoPanelRetorno=m_PesoPanel,
        PesoPanelesMlRetorno=m_PesoPanelesMl,
        DensidadRetorno = m_Densidad, DesarrolloPerfilRetorno =
        m_DesarrolloPerfil,EspesorPerfilRetorno = m_EspesorPerfil,
        VolumenTotalRetorno = m_VolumenTotal, PesoTotalRetorno =
        m_PesoTotal,SobrecargaRetorno=m_Sobrecarga;

        VueltaCargas = true;
        EndDialog(NULL);
}
```

En el cuadro de dialogo, cuando se inicia, se hace una llamada por defecto a la función OnInitDialog(), siendo que se impone una condiciona cumplir, el análisis o verificación, si se llama al cuadro de diálogo después de haber abierto un proyecto existente en disco (deserializado), y si se entra en el cuadro no por primera vez, sino en retorno tras haber retrocedido pantallas en la aplicación, en cuyo caso, se cargaran los datos disponibles y anteriormente introducidos por el usuario, en los diversos apartado de la interface.

```
BOOL Cargas::OnInitDialog()
{
        CDialogEx::OnInitDialog();

        // TODO:  Agregue aquí la inicialización adicional
        if (VueltaCargas | AbiertoAachivo)
        {
                m_Zona=ZonaRetorno,  m_Aspereza=AsperezaRetorno;
                m_PresDinamica=PresDinamicaRetorno,
                m_AlturaInstal=AlturaInstalRetorno,   m_K=KRetorno,   m_L=LRetorno,
                m_Z=ZRetorno,   m_PendienteP=PendientePRetorno,
                m_CoefEolico=CoefEolicoRetorno,   m_PresDin=PresDinRetorno,
                m_CExp=CExpRetorno,   m_AccViento=AccVientoRetorno,
                m_Largo=LargoRetorno,   m_Ancho=AnchoRetorno;
                m_PFila=PFilaRetorno;
                m_Fuerza=FuerzaRetorno,   m_PesoPanel=PesoPanelRetorno,
                m_PesoPanelesMl=PesoPanelesMlRetorno,
                m_Densidad=DensidadRetorno,
                m_DesarrolloPerfil=DesarrolloPerfilRetorno,
                m_EspesorPerfil=EspesorPerfilRetorno,
                m_VolumenTotal=VolumenTotalRetorno,   m_PesoTotal=PesoTotalRetorno,
                m_Sobrecarga=SobrecargaRetorno;

                VueltaCargas = false;
                UpdateData(FALSE);
        }
        return TRUE;  // return TRUE unless you set the focus to a control
        // EXCEPCIÓN: las páginas de propiedades OCX deben devolver FALSE
}
```

3.5.- Inversion económica

En el archivo desarrollado a continuación, se implementarán los elementos necesarios para analizar la rentabilidad económica de la instalación proyectada en base a criterio financieros de toma de decisiones, tales como el VAN y el TIR.

Se capturarán de sus respectivas pantallas o cuadro de edición, cada uno de los valores económicos implicados en el cálculo de rentabilidad de la inversión, operando a fin de conocer el coste de cada una de las partidas en función del número de unidades de un tipo de material o elemento, por el coste unitario del mismo. De igual manera se calcular el coste de la instalación en función de su presupuesto de ejecución material, se obtendrán los datos introducidos de vida útil, subvenciones, etc…

```
void Inversion::OnEnKillfocusEurospan()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	UpdateData(TRUE);
	m_TotalPaneles = m_UdPaneles*m_EurosPaneles;
	UpdateData(FALSE);
}

void Inversion::OnEnKillfocusEurosinv()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	UpdateData(TRUE);
	m_TotalInversores = m_UdInversores*m_EurosInversores;
	UpdateData(FALSE);
}


void Inversion::OnEnKillfocusEurosacu()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	UpdateData(TRUE);
	m_TotalAcumuladores = m_UdAcumuladores*m_EurosAcumuladores;
	UpdateData(FALSE);
}

void Inversion::OnEnKillfocusEurosstru()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	UpdateData(TRUE);
	m_TotalEstructura = m_UdEstructura*m_EurosEstrcutura;
	UpdateData(FALSE);
}

void Inversion::OnEnKillfocusEurosotro()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	UpdateData(TRUE);
	m_TotalOtros = m_UdOtros*m_EurosOtros;
	UpdateData(FALSE);
}

void Inversion::OnEnKillfocusEurosmano()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	UpdateData(TRUE);
	m_TotalMano = m_UdMano*m_EurosMano;
	m_CosteInstal = m_TotalPaneles + m_TotalInversores + m_TotalAcumuladores +
		m_TotalEstructura + m_TotalOtros + m_TotalMano;
	UpdateData(FALSE);
}
```

```
void Inversion::OnEnKillfocusPorproy()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	UpdateData(TRUE);
	m_CosteAImpuestos = m_CosteInstal - m_CosteInstal*(m_Subvenciones / 100) +
	m_CosteInstal*(m_Licencias / 100) + m_CosteInstal*(m_GGenerales / 100) +
	m_CosteInstal*(m_Proyectos / 100);
	UpdateData(FALSE);

}

void Inversion::OnEnKillfocusPorimp()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	UpdateData(TRUE);
	m_CosteTrasImp = m_CosteAImpuestos + m_CosteAImpuestos*(m_Impuestos /
	100);
	UpdateData(FALSE);
}
```

Se interpone un cuadro de elección, en el que se indica si las instalaciones conectadas o no a red, y en función de la elección, se tomarán diferentes elecciones, atendiendo al porcentaje de cobertura de la demanda o de la producción del sistema.

```
void Inversion::OnEnKillfocusPorcentajecobertura()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	UpdateData(TRUE);
	CButton* m_TipoInstalacion = (CButton*)GetDlgItem(IDC_TIPOINSTAL);
```

Si la instalación está conectada a red, esto es, no es una instalación aislada, toda la producción se destinara a venta, multiplicándose la producción anual por el precio de venta de la energía.

```
	if (!m_VerificacionInstalacion)
	{
		m_CoberturaDemanda = 0.00;
		Q = m_Necesidades*m_VentaEnerg;
	}
```

En caso contrario, si la instalación es aislada, existirá un coste de compra de la energía, el cual se minorará con el ahorro que supone la cobertura de la demanda energética total de la instalación, que proveer la instalación de generación diseñada por el usuario.

```
	else if (m_VerificacionInstalacion)
	{
		m_CoberturaDemanda;
		Q = (m_Necesidades*m_CompraEnerg) -
		(m_Necesidades*m_CompraEnerg*(1-(m_CoberturaDemanda/100)));
	}
```

Se ha programado un ciclo para el cálculo del VAN, en el cual se aplicará la fórmula matemática correspondiente al mismo (base teórica), desde el momento de la inversión, hasta la amortización o paso de vida útil de la instalación, aceptándose o

rechazando la inversión, en función de si el valor que ofrece el cálculo es superior o no a 0.

```
double iciclo;
for (iciclo = 1; iciclo<=m_VidaUtil; iciclo++)
{
	double factor = (1 + (m_InteresMercado/100));
	double denominador = pow(factor, m_VidaUtil);
	m_VAN = m_VAN + (Q / denominador);

}
m_VAN = m_VAN - m_CosteAImpuestos;

if (m_VAN > 0)
{
	m_OkVan = _T("La inversión debe realizarse");
}
if (m_VAN == 0)
{
	m_OkVan = _T("La inversión es indiferente");
}
if (m_VAN < 0)
{
	m_OkVan = _T("La inversión no es recomendable");
}
UpdateData(FALSE);
```

De igual manera al cálculo del VAN, se programa un ciclo iterativo para el cálculo del TIR, en el cual se busca el valor del interés que hace que el VAN = 0, realizándose o repitiéndose el bucle de iteración, variando el valor de dicho interés, hasta que se ajuste la ecuación.

```
//inicia la iteraccion para calcular el TIR
double r = 0.0001;
van = m_CosteAImpuestos + 1; // primer valor para que deje entrar en el
bucle
while ((-m_CosteAImpuestos+van) > 0.0000)
{
	double factor = (1 + r);
	van = 0;
	for (iciclo = 1; iciclo <= m_VidaUtil; iciclo++)
	{
		double denominador = pow(factor, m_VidaUtil);
		van = van + (Q / denominador);
	}
	r = r + 0.0001;
}

m_TIR = r * 100;
if (m_TIR > m_InteresMercado)
{
	m_OkTir = _T("La inversión debe realizarse");
}
if (m_TIR == m_InteresMercado)
{
	m_OkTir = _T("La inversión es indiferente");
}
if (m_TIR < m_InteresMercado)
```

```
	{
		m_OkTir = _T("La inversión no es recomendable");
	}
	UpdateData(FALSE);
}
```

En el presente caso, los controles de comienzo, marcha atrás en el cuadro de dialogo, y paso adelante a la siguiente pantalla, serán idénticos a los casos o cuadros de dialogo explicados de manera previa, con la salvedad, de que al ser la última pantalla de la aplicación, en vez de pasar a otro cuadro de edición, se generara un archivo de texto de salida (no explicado dado que se analiza su generación en memoria de proyecto).

4.- Instalaciones fotovoltaicas conectadas a red

En el presente apartado, cuando el usuario pulsa el botón de este tipo de instalaciones desde la pantalla principal del programa, se lanza la ejecución del cálculo de la presente clase de instalaciones. Con respecto a la pantalla de inserción de puntos de consumo, así como sus elementos asociados (cuadros de dialogo para inserción de datos), tales elementos serán coincidentes con el apartado de instalaciones aisladas de red, no entrándose por tanto a comentar los mismos.

4.1.- Dimensionado generador red

Se instala un botón, que mediante la función ShellExecute, lazan el navegador de internet por defectos del equipo informático del usuario, accediendo a GoogleMaps a fin de poder consultar coordenadas o latitud de orientación del lugar de instalación de los panales del proyecto.

```
void DGeneradorRed::OnBnClickedBlatitud()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	ShellExecute(NULL, _T("open"), _T("https://www.google.es/maps/"), NULL,
	NULL, SW_SHOWNORMAL);
}
```

La función siguiente controla un cuadro desplegable, en el cual el usuario puede seleccionar una opción de tipo de instaalcion, para que en función del registro elegido, mediante elementos de selección “if”, se carguen las perdidas máximas permitidas para las mismas.

```
void DGeneradorRed::OnCbnSelchangeTipoperd()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	m_Localidad;
	m_Provincia;
	m_Pais;
	m_Comentarios;
	m_Latitud;
	m_AlfaDiseno;
	m_BetaDiseno;
	m_TipoPerd;
```

```
	UpdateData(TRUE);

	BetaDisenoAux = m_BetaDiseno;
	//adquisicion latitud para intercambio informacin entre clases
	LatitudAux = GetDlgItemInt(IDC_Latitud);

	if (m_TipoPerd == _T("Caso general"))
	{
		m_PerPerdidas = 10;
	}

	if (m_TipoPerd == _T("Superposición"))
	{
		m_PerPerdidas = 20;
	}

	if (m_TipoPerd == _T("Integración Arquitectónica"))
	{
		m_PerPerdidas = 40;
	}

	UpdateData(FALSE);
}
```

Se lanza un cuadro de dialogo que muestra el diagrama de cálculo de pérdidas por orientación.

```
void DGeneradorRed::OnBnClickedDiagrama()
{
// TODO: Agregue aquí su código de controlador de notificación de control
	PredOrientacionGraf DlgPredOrientacionGraf(NULL); //Llama al constructor
	del dialogo para lanzar la pantalla

	if (DlgPredOrientacionGraf.DoModal() == IDOK)
	{

	}

}
```

Una vez introducidas todas la variables, se lanza la función, en la que se corrige la inclinación máxima y mínima permitida en función de la latitud de la instalación, dando paso a bucles “if”, en los cuales se comparara el rango de inclinación de diseño de la instalación, calculándose las pérdidas por orientación, y verificando si la instalación está dentro de los márgenes permitidos y requeridos de diseño.

```
void DGeneradorRed::OnEnKillfocusIncmin()
{
// TODO: Agregue aquí su código de controlador de notificación de control

	m_IncliMax;
	m_InclMin;

	UpdateData(TRUE);

	m_IncMaxCorr = m_IncliMax - (41 - m_Latitud);
	m_IncMinCorr = m_InclMin - (41 - m_Latitud);
```

```
	if (m_IncMinCorr < 0)
	{
		m_IncMinCorr = 0;
	}

//Calculos matemmaticos correspondinetes al analisis del angulo maximo y min
	//de inclinacin, asi como verificación porcentaje de pérdidas.

	if (m_BetaDiseno <= 15)
	{
		const double X = pow(10.0, -4);
		double B = pow((m_BetaDiseno - m_Latitud + 10), 2);
		m_PerdidasOrientacion = 100 * (1.2*X*B);

		PerdidasOrientacionAux = m_PerdidasOrientacion;
	}

	else if ((m_BetaDiseno > 15) && (m_BetaDiseno < 90))
	{
		const double X = pow(10.0, -4);
		const double Y = pow(10.0, -5);
		double B = pow((m_BetaDiseno - m_Latitud + 10), 2);
		double A = pow(m_AlfaDiseno, 2);
		m_PerdidasOrientacion = 100 * ((1.2*X*B) + (3.5*Y*A));

		PerdidasOrientacionAux = m_PerdidasOrientacion;
	}

	UpdateData(FALSE);

if ((m_BetaDiseno > m_IncMaxCorr) || (m_BetaDiseno < m_IncMinCorr) ||
(m_PerdidasOrientacion > m_PerPerdidas))
	{
	MessageBox(_TEXT("Pérdidas o inclinación fuera de rango permitido"),
	_TEXT("Pérdidas inclinación y orientación"), MB_ICONERROR);
	}
}
```

Existe instalada una herramienta tipo botón, mediante la cual se puede lanzar la automatización del cálculo de las perdidas por sombreo (explicado en puntos posteriores), según lo cual se abriría un cuadro de diálogo especifico a tal uso.

```
void DGeneradorRed::OnBnClickedPersombras()
	{
	// TODO: Agregue aquí su código de controlador de notificación de control
	// acceso a la plantilla de calculo de perdidas por sombras

	CalcPerdSombra DlgCalcPerdSombra(NULL); //Llama al constructor del dialogo
	para lanzar la pantalla

		if (DlgCalcPerdSombra.DoModal() == IDOK)
		{
		//cuando se cierra dialogo con Ok, establece variable para sacarla
		por pantalla
			m_PerdSombras = PerdSombrasAux;
			UpdateData(FALSE);
		}

	}
```

La aplicación mediante la presente función, calcula las distancia mínimas entre filas y para con obstáculos, si existieran, en función de la latitud de ubicación de la instalación, y atendiendo a las ecuaciones pertinentes del pliego de condiciones técnicas específicas.

```
void DGeneradorRed::OnEnKillfocusHfilas()
{
// TODO: Agregue aquí su código de controlador de notificación de control
// cálculo distancias minimas entre filas y obstaculos
	//lectura variables de altura
	m_HObsataculo;
	m_HFilas;
	UpdateData(TRUE);

	//calulo parametro adimensional "K" y distancias
	K = 1 / (tan(61 - m_Latitud));
	m_DObstaculo = m_HObsataculo*K;
	m_DFilas = m_HFilas*K;
	UpdateData(FALSE);
}
```

Se han obviado en la explicación, cualquier código reiterativo y repetido en apartados anteriores y coincidentes con el desarrollo global de la aplicación, tales como inicialización de código de vista, funciones de adelanto o retorno en pantallas de usuario.

4.2.- Diseño sistema red

Función activada por pulsación de botón que permite la automatización o selección de valores predeterminados de pérdidas del sistema (el usuario puede optar por introducirlos manualmente mediante cuadros de edición que serán capturadas con *UpdateData).*

```
void DisSistemaRed::OnBnClickedVestimados()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	//cargar valores de tabla por defecto para perdidas

	m_LCab = 0.02;
	m_LPol = 0.03;
	m_LDis = 0.02;
	m_LRef = 0.03;
	m_Otros = 0.05;
	UpdateData(FALSE);
```

Una vez cargados los valores enla variable miembro pertinente, se invocará a la función específica que calcula las pérdidas asociadas de cada clase.

```
	//llama a funciones para generar salidas en cada casilla de perdida y
rendimiento
	DisSistemaRed::OnEnKillfocusPerdcable();
	DisSistemaRed::OnEnKillfocusPerdpol();
	DisSistemaRed::OnEnKillfocusPerddis();
	DisSistemaRed::OnEnKillfocusPerdref();
	DisSistemaRed::OnEnKillfocusPerdotros();
}
```

Como en el resto de secciones, en algún punto, al abandonarse una de las casillas de introducción de datos, se lanza una función, que captura datos, los opera. En la presente función se calculan los datos del generador en diseño, y se verifica que su potencia teorica.

```
void DisSistemaRed::OnEnKillfocusNpaneles()
{
	//modificacion y cargado variables en casillas de caracteristicas
	generador
	//se activa al perder el foco de atención sobre el número de paneles

	m_ICorto;
	m_IPMax;
	m_VAbierto;
	m_VPMax;
	m_PPanel;
	m_NPaneles;
	UpdateData(TRUE); //obtiene valores escritos de casillas

	m_PTeorica = m_PPanel*m_NPaneles;
	PTeoricaAux = m_PTeorica;
	UpdateData(FALSE); //muestra valores en casillas
}
```

Se procesan cada una de la funciones que calculan las pérdidas de rendimiento con diversos orígenes en el sistema, provenientes las llamas de las funciones invocadas en secciones previas.

```
void DisSistemaRed::OnEnKillfocusTamb()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	//modificacion y cargado variables en casillas de medidas instalacion
	//se activa al perder el foco de atención sobre Tª ambiente

	m_Tonc;
	m_E;
	m_PInv;
	m_TAmb;
	UpdateData(TRUE); //obtiene valores escritos de casillas

	//variables intercambio
	ToncAux = m_Tonc;
	EAux = m_E;

	m_Tc = m_TAmb + (((m_Tonc - 20)*m_E) / 800);
	UpdateData(FALSE); //muestra valores en casillas

}

// funciones de salida de datos de perdidas y rendimeintos
void DisSistemaRed::OnEnKillfocusPerdcable()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	m_LCab;
	UpdateData(TRUE);
	m_LTem = 0.0035*(m_Tc - 25);
	m_RCab = (1 - m_LCab);
	m_RTem = (1 - m_LTem);
```

```
	UpdateData(FALSE);
}

void DisSistemaRed::OnEnKillfocusPerdpol()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	UpdateData(TRUE);
	m_LPol;
	m_RPol = (1 - m_LPol);
	UpdateData(FALSE);
}

void DisSistemaRed::OnEnKillfocusPerddis()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	m_LDis;
	UpdateData(TRUE);
	m_RDis = (1 - m_LDis);
	UpdateData(FALSE);
}

void DisSistemaRed::OnEnKillfocusPerdref()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	m_LRef;
	UpdateData(TRUE);
	m_RRef = (1 - m_LRef);
	UpdateData(FALSE);
}

void DisSistemaRed::OnEnKillfocusPerdotros()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	m_Otros;
	UpdateData(TRUE);
	m_ROtros = (1 - m_Otros);
	m_RTotal = (1 - m_LPol)*(1 - m_LDis)*(1 - m_LRef)*(1 - m_Otros);
	m_PFov = m_PInv / (1 - m_LCab);
	m_PNominal = ((m_PFov * 1000) / (m_RTotal*(1-m_LTem)*m_E));
	UpdateData(FALSE);
}
```

4.3.- Potencia esperada red

El cuadro de dialogo presente, calcula la potencia esperada inyectada a red, en función de diversos parámetros mensuales, tales como irradiación, temperatura , perdidas etc. Según esto cada mes, requiere la introducción de dos parámetros, siendo estos temperatura media del mes, e irradiación media, siendo que al introducir los datos secuencialmente en cada mes, se activaran los calculo matemáticos correspondientes a este periodo de tiempo, y así secuencialmente y de manera idéntica de enero a diciembre.

```
void PEsperadaRed::OnEnKillfocusG0enero()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
```

Se capturan los valores introducidos en las dos casillas del mes requeridas por el sistema

```
	m_TEnero;
	m_G0Enero;
	UpdateData(TRUE);
```

Cargo los valores en variables auxiliares para operar con estas sin corromper los datos origen.

```
	Gdm0Aux = m_G0Enero; //carga variable auxiliar para calculos en funcion
	estandar
	TAux = m_TEnero;
```

Se hacen llamadas para el cálculo, externalizado fuera de las funciones de cada mes, del resto de casillas necesarias a implementar en la interface, operando en el interior de las mismas con las variables introducidas y datos preexistentes en el recorrido de los cuadros de diálogo.

```
	PEsperadaRed::CalculoGdm(); //llama a función para calcular valor
	Gdm(alfa,beta)
	PEsperadaRed::CalculoPR(); //llama a función para calcular PR.
	PEsperadaRed::CalculoEd(); //llama a función calculo energia.
```

Se retorna el valor devuelto por las funciones de cálculo, y se carga en variables miembro para presentación en interface.

```
	m_GEnero = GdmBetaAux;
	m_PREnero = PRAux;
	m_EEnero = m_E;
	UpdateData(FALSE);
```

Se hace una última llamada una vez se tiene todos los valores de cada mes, para incrementar o añadir tale valores al cálculo promedio anual.

```
	PEsperadaRed::Promedios(); //llama a función calculo promedios.
}

void PEsperadaRed::OnEnKillfocusG0febrero()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	m_TFebrero;
	m_G0Febrero;
	UpdateData(TRUE);

	Gdm0Aux = m_G0Febrero; //carga variable auxiliar para calculos en funcion
	estandar
	TAux = m_TFebrero;
	PEsperadaRed::CalculoGdm(); //llama a función para calcular valor
	Gdm(alfa,beta)
	PEsperadaRed::CalculoPR(); //llama a función para calcular PR.
	PEsperadaRed::CalculoEd(); //llama a función calculo energia.

	m_GFebrero = GdmBetaAux;
	m_PRFebrero = PRAux;
	m_EFebrero = m_E;
	UpdateData(FALSE);
```

```
	PEsperadaRed::Promedios(); //llama a función calculo promedios.
}

void PEsperadaRed::OnEnKillfocusG0marzo()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	m_TMarzo;
	m_G0Marzo;
	UpdateData(TRUE);

	Gdm0Aux = m_G0Marzo; //carga variable auxiliar para calculos en funcion
	estandar
	TAux = m_TMarzo;
	PEsperadaRed::CalculoGdm(); //llama a función para calcular valor
	Gdm(alfa,beta)
	PEsperadaRed::CalculoPR(); //llama a función para calcular PR.
	PEsperadaRed::CalculoEd(); //llama a función calculo energia.

	m_GMarzo = GdmBetaAux;
	m_PRMarzo = PRAux;
	m_EMarzo = m_E;
	UpdateData(FALSE);

	PEsperadaRed::Promedios(); //llama a función calculo promedios.
}

void PEsperadaRed::OnEnKillfocusG0abril()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	m_TAbril;
	m_G0Abril;
	UpdateData(TRUE);

	Gdm0Aux = m_G0Abril; //carga variable auxiliar para calculos en funcion
	estandar
	TAux = m_TAbril;
	PEsperadaRed::CalculoGdm(); //llama a función para calcular valor
	Gdm(alfa,beta)
	PEsperadaRed::CalculoPR(); //llama a función para calcular PR.
	PEsperadaRed::CalculoEd(); //llama a función calculo energia.

	m_GAbril = GdmBetaAux;
	m_PRAbril = PRAux;
	m_EAbril = m_E;
	UpdateData(FALSE);

	PEsperadaRed::Promedios(); //llama a función calculo promedios.
}


void PEsperadaRed::OnEnKillfocusG0mayo()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	m_TMayo;
	m_G0Mayo;
	UpdateData(TRUE);

	Gdm0Aux = m_G0Mayo; //carga variable auxiliar para calculos en funcion
	estandar
```

```
	TAux = m_TMayo;
	PEsperadaRed::CalculoGdm(); //llama a función para calcular valor
	Gdm(alfa,beta)
	PEsperadaRed::CalculoPR(); //llama a función para calcular PR.
	PEsperadaRed::CalculoEd(); //llama a función calculo energia.

	m_GMayo = GdmBetaAux;
	m_PRMayo = PRAux;
	m_EMayo = m_E;
	UpdateData(FALSE);

	PEsperadaRed::Promedios(); //llama a función calculo promedios.
}

void PEsperadaRed::OnEnKillfocusG0junio()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	m_TJunio;
	m_G0Junio;
	UpdateData(TRUE);

	Gdm0Aux = m_G0Junio; //carga variable auxiliar para calculos en funcion
	estandar
	TAux = m_TJunio;
	PEsperadaRed::CalculoGdm(); //llama a función para calcular valor
	Gdm(alfa,beta)
	PEsperadaRed::CalculoPR(); //llama a función para calcular PR.
	PEsperadaRed::CalculoEd(); //llama a función calculo energia.

	m_GJunio = GdmBetaAux;
	m_PRJunio = PRAux;
	m_EJunio = m_E;
	UpdateData(FALSE);

	PEsperadaRed::Promedios(); //llama a función calculo promedios.
}

void PEsperadaRed::OnEnKillfocusG0julio()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	m_TJulio;
	m_G0Julio;
	UpdateData(TRUE);

	Gdm0Aux = m_G0Julio; //carga variable auxiliar para calculos en funcion
	estandar
	TAux = m_TJulio;
	PEsperadaRed::CalculoGdm(); //llama a función para calcular valor
	Gdm(alfa,beta)
	PEsperadaRed::CalculoPR(); //llama a función para calcular PR.
	PEsperadaRed::CalculoEd(); //llama a función calculo energia.

	m_GJulio = GdmBetaAux;
	m_PRJulio = PRAux;
	m_EJulio = m_E;
	UpdateData(FALSE);

	PEsperadaRed::Promedios(); //llama a función calculo promedios.
}
```

```
void PEsperadaRed::OnEnKillfocusG0agosto()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	m_TAgosto;
	m_G0Agosto;
	UpdateData(TRUE);

	Gdm0Aux = m_G0Agosto; //carga variable auxiliar para calculos en funcion
	estandar
	TAux = m_TAgosto;
	PEsperadaRed::CalculoGdm(); //llama a función para calcular valor
	Gdm(alfa,beta)
	PEsperadaRed::CalculoPR(); //llama a función para calcular PR.
	PEsperadaRed::CalculoEd(); //llama a función calculo energia.

	m_GAgosto = GdmBetaAux;
	m_PRAgosto = PRAux;
	m_EAgosto = m_E;
	UpdateData(FALSE);

	PEsperadaRed::Promedios(); //llama a función calculo promedios.
}

void PEsperadaRed::OnEnKillfocusG0septiembre()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	m_TSeptiembre;
	m_G0Septiembre;
	UpdateData(TRUE);

	Gdm0Aux = m_G0Septiembre; //carga variable auxiliar para calculos en
	funcion estandar
	TAux = m_TSeptiembre;
	PEsperadaRed::CalculoGdm(); //llama a función para calcular valor
	Gdm(alfa,beta)
	PEsperadaRed::CalculoPR(); //llama a función para calcular PR.
	PEsperadaRed::CalculoEd(); //llama a función calculo energia.

	m_GSeptiembre = GdmBetaAux;
	m_PRSeptiembre = PRAux;
	m_ESeptiembre = m_E;
	UpdateData(FALSE);

	PEsperadaRed::Promedios(); //llama a función calculo promedios.
}

void PEsperadaRed::OnEnKillfocusG0octubre()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	m_TOctubre;
	m_G0Octubre;
	UpdateData(TRUE);

	Gdm0Aux = m_G0Octubre; //carga variable auxiliar para calculos en funcion
	estandar
	TAux = m_TOctubre;
	PEsperadaRed::CalculoGdm(); //llama a función para calcular valor
	Gdm(alfa,beta)
	PEsperadaRed::CalculoPR(); //llama a función para calcular PR.
	PEsperadaRed::CalculoEd(); //llama a función calculo energia.
```

```
	m_GOctubre = GdmBetaAux;
	m_PROctubre = PRAux;
	m_EOctubre = m_E;
	UpdateData(FALSE);
}

void PEsperadaRed::OnEnKillfocusG0noviembre()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	m_TNoviembre;
	m_G0Noviembre;
	UpdateData(TRUE);

	Gdm0Aux = m_G0Noviembre; //carga variable auxiliar para calculos en
	funcion estandar
	TAux = m_TNoviembre;
	PEsperadaRed::CalculoGdm(); //llama a función para calcular valor
	Gdm(alfa,beta)
	PEsperadaRed::CalculoPR(); //llama a función para calcular PR.
	PEsperadaRed::CalculoEd(); //llama a función calculo energia.

	m_GNoviembre = GdmBetaAux;
	m_PRNoviembre = PRAux;
	m_ENoviembre = m_E;
	UpdateData(FALSE);

	PEsperadaRed::Promedios(); //llama a función calculo promedios.
}

void PEsperadaRed::OnEnKillfocusG0diciembre()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	m_TDiciembre;
	m_G0Diciembre;
	UpdateData(TRUE);

	Gdm0Aux = m_G0Diciembre; //carga variable auxiliar para calculos en
	funcion estandar
	TAux = m_TDiciembre;
	PEsperadaRed::CalculoGdm(); //llama a función para calcular valor
	Gdm(alfa,beta)
	PEsperadaRed::CalculoPR(); //llama a función para calcular PR.
	PEsperadaRed::CalculoEd(); //llama a función calculo energia.

	m_GDiciembre = GdmBetaAux;
	m_PRDiciembre = PRAux;
	m_EDiciembre = m_E;
	UpdateData(FALSE);

	PEsperadaRed::Promedios(); //llama a función calculo promedios.
}
```

Función destinada al cálculo de Gdm, en función de la inclinación y ángulo azimut de diseño de la inclinación introducidos por el usuario. Función invocada desde cada uno de los meses que se operan al introducir valores.

```
void PEsperadaRed::CalculoGdm()
{
```

```
	//calculo de valores de Gdm para alfa y beta concreto utilizado
	m_Latitud = LatitudAux;
```

Se calcula matemáticamente la inclinación óptima para la latitud dela instalación, y con ella se calcula el valor de Gdm óptimo.

```
	BetaOp = 3.7 + 0.69*m_Latitud;
	const double M = pow(10, -4);
	double B = pow(BetaOp, 2);
	GdmOp = (Gdm0Aux)/(1-(4.46*M*BetaOp)-(1.19*M*B));

	K = GdmOp / Gdm0Aux;
```

En función de los factores de pérdidas que se estén presentando en la instalación, se selecciona el tipo de operación matemática necesaria para el cálculo del valor de Gdm de la instalación que se esté calculando, siendo este el valor de retorno a los cuadros de edición correspondientes de cada mes.

```
	double FI = 1-(PerdidasOrientacionAux/100); //perdidas orientacion
	double FS = 1-(PerdSombrasAux/100); // perdidas sombreo

	if ((FS > 0.90)&&(FI>0.000)) // si perdidas sombreo son menores 10% no se
	descuentan
	{
		GdmBetaAux = Gdm0Aux*K*FI;
	}

	else if ((FS > 0.90) && (FI==0.000))
	{
		GdmBetaAux = Gdm0Aux*K;
	}

	else if ((FS <= 0.90) && (FI == 0.000))  //si son superiores al 10% se
	descuentan de Gdm
	{
		GdmBetaAux = Gdm0Aux*K*FS;
	}

	else if ((FS <= 0.90) && (FI > 0.000))
	{
		GdmBetaAux = Gdm0Aux*K*FS*FI;
	}
}
```

Se calcula el rendimiento de la instalación mensual en función de los valores de pérdidas que se presenten, y en función de la variación de temperaturas medias mensuales.

```
void PEsperadaRed::CalculoPR()
{
	//calculo de valores de PR
	m_Tonc = ToncAux;
	m_E = EAux;

	Tc = TAux + (((m_Tonc - 20)*m_E) / 800);
	RTem = 1 - (0.0035*(Tc - 25));
```

```
        // intrecambio datos perdidas
        m_RCab = RCabAux;
        m_RPol = RPolAux;
        m_RDis = RDisAux;
        m_RRef = RRefAux;
        m_ROtros = ROtrosAux;

        PR = RTem*m_RCab*m_RPol*m_RDis*m_RRef*m_ROtros;  //calculo de PR en
        función rendimientos
        PRAux = PR;
}
```

Función para el cálculo de la energía producida media mensual.

```
void PEsperadaRed::CalculoEd()
{
        //calculo de valores de Ed
        m_PTeorica = PTeoricaAux/1000; // unidades KW
        m_E = (PRAux*GdmBetaAux*m_PTeorica); //unidades KWh/dia
}
```

Función para el cálculo de los promedios de cada uno de los cinco factores mostrado para cada mes, siendo así, que conos resultado completos en cada uno de los cuadros o casillas de cada mes, se promediaran entre los doce meses del año, a fin de dar salida por pantalla del promedio de la información representativa de producción a red.

```
void PEsperadaRed::Promedios()
{
        //calculo de promedios y salida de datos de casillas correspondientes
        m_TPromedio =
        (m_TEnero+m_TFebrero+m_TMarzo+m_TAbril+m_TMayo+m_TJunio+m_TJulio+m_TAgosto
        +m_TSeptiembre+m_TOctubre+m_TNoviembre+m_TDiciembre)/12;
        m_G0Promedio = (m_G0Enero + m_G0Febrero + m_G0Marzo + m_G0Abril + m_G0Mayo
        + m_G0Junio + m_G0Julio +
        m_G0Agosto + m_G0Septiembre + m_G0Octubre + m_G0Noviembre +
        m_G0Diciembre)/12;
        m_GPromedio = (m_GEnero + m_GFebrero + m_GMarzo + m_GAbril + m_GMayo +
        m_GJunio + m_GJulio +m_GAgosto + m_GSeptiembre + m_GOctubre + m_GNoviembre
        + m_GDiciembre)/12;
        m_PRPromedio = (m_PREnero + m_PRFebrero + m_PRMarzo + m_PRAbril + m_PRMayo
        + m_PRJunio + m_PRJulio + m_PRAgosto +m_PRSeptiembre + m_PROctubre +
        m_PRNoviembre + m_PRDiciembre) / 12;
        m_EPromedio = (m_EEnero + m_EFebrero + m_EMarzo + m_EAbril + m_EMayo +
        m_EJunio + m_EJulio + m_EAgosto + m_ESeptiembre + m_EOctubre +
        m_ENoviembre + m_EDiciembre) / 12;

        UpdateData(FALSE); //actualiza salidas a pantalla de promedios
}
```

4.4.- Cargas e inversión

Con respecto al cálculo de sobrecargas de cubierta y cálculo de rentabilidad económica de la inversión en la aplicación realizada por el usuario, se lanzarán y emplearán las mismas pantallas con similares características a la previamente explicadas

en el apartado correspondiente a instalaciones aisladas de red, siendo que el cálculo es análogo y permite la adaptación de estas pantallas de unos sistemas a otros.

5.- Base de datos

La realización de la base de datos para mostrar la información, está formada por cuadros de dialogo, embebidos en una interface gráfica que forma un cuadro de dialogo superior, esto es, hojas de propiedades. Se describirá el código ejecutado, obviando la explicación genérica de este tipo de control que fundamenta y articula la base de datos, dado que el mismo ha sido explicado ampliamente en la memoria del proyecto.

Cuando se invoca a la clase que contiene la base de datos, se inicializa el constructor, y como información de inicio se llama a la función *OnInitDialog()*, en la cual se carga la estructura generales de los controles de la vista, en este caso un cuadro de dialogo con múltiples hojas de propiedades insertadas.

```
BOOL BaseDatosGeneral::OnInitDialog()
{
	CDialogEx::OnInitDialog();

	// TODO:  Agregue aquí la inicialización adicional
```

Se definen y muestran los controles insertados en la hoja de propiedades base, numerándose los mismos y poniéndoles nombres. Estos controles se insertan y controlan con una variable miembro de la case base de la plantilla en la que se incrustan.

```
	m_CtrlBaseDatosGeneral.InsertItem(0, _T("Paneles fotovoltaicos"));
	m_CtrlBaseDatosGeneral.InsertItem(1, _T("Inversores"));
	m_CtrlBaseDatosGeneral.InsertItem(2, _T("Acumuladores"));
	m_CtrlBaseDatosGeneral.InsertItem(3, _T("Regulador Carga"));
```

Se crean los controles incrustado, a modo de botones de pulsación, con la identificación de cada uno de ellos, y una variable asociada a la vista de ventana.

```
	CRect rect;
	m_CtrlBaseDatosGeneral.GetClientRect(&rect);
	m_DatosPanel.Create(IDD_BASEDATOSPANELES, &m_CtrlBaseDatosGeneral);
	m_WndShow = &m_DatosPanel;
	m_DatosInversor.Create(IDD_BASEDATOSINVERSORES, &m_CtrlBaseDatosGeneral);
	m_WndShow = &m_DatosInversor;
	m_DatosAcumulador.Create(IDD_BASEDATOSACUMULADORES,
	&m_CtrlBaseDatosGeneral);
	m_WndShow = &m_DatosAcumulador;
	m_DatosRegulador.Create(IDD_BASEDATOSREGULADOR, &m_CtrlBaseDatosGeneral);
	m_WndShow = &m_DatosRegulador;
```

Se inicializa, mostrando por defecto el primer botón de pulsación, de modo que este es el que aparentemente está marcado en la vista.

```
	//comienza activada por defecto la ventana 0
	iSelControl = 0;
```

```
	m_DatosPanel.ShowWindow(SW_SHOW);
	m_WndShow = &m_DatosPanel;

	return TRUE;  // return TRUE unless you set the focus to a control
				// EXCEPCIÓN: las páginas de propiedades OCX deben
devolver FALSE
}
```

Mediante el uso de la siguiente función, se detecta si el botón o pestaña de la vista principal ha cambiado, entrándose en una selección, en la cual, en función del número de control o identificación pulsada, entra en el código que muestra la pantalla o información embebida correspondiente a la clase que activa el botón de pulsación. Esto es, muestra la ventana seleccionada, y oculta el resto.

```
void BaseDatosGeneral::OnTcnSelchangeBasedatos(NMHDR *pNMHDR, LRESULT *pResult)
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	iSelControl = m_CtrlBaseDatosGeneral.GetCurSel();

	if (iSelControl == 0)
	{
		m_DatosPanel.ShowWindow(SW_SHOW);
		m_DatosInversor.ShowWindow(SW_HIDE);
		m_DatosAcumulador.ShowWindow(SW_HIDE);
		m_DatosRegulador.ShowWindow(SW_HIDE);
		m_WndShow = &m_DatosPanel;

	}
	else if (iSelControl == 1)
	{
		m_DatosInversor.ShowWindow(SW_SHOW);
		m_DatosPanel.ShowWindow(SW_HIDE);
		m_DatosAcumulador.ShowWindow(SW_HIDE);
		m_DatosRegulador.ShowWindow(SW_HIDE);
		m_WndShow = &m_DatosInversor;
	}
	else if (iSelControl == 2)
	{
		m_DatosInversor.ShowWindow(SW_HIDE);
		m_DatosPanel.ShowWindow(SW_HIDE);
		m_DatosAcumulador.ShowWindow(SW_SHOW);
		m_DatosRegulador.ShowWindow(SW_HIDE);
		m_WndShow = &m_DatosAcumulador;
	}
	else if (iSelControl == 3)
	{
		m_DatosInversor.ShowWindow(SW_HIDE);
		m_DatosPanel.ShowWindow(SW_HIDE);
		m_DatosAcumulador.ShowWindow(SW_HIDE);
		m_DatosRegulador.ShowWindow(SW_SHOW);
		m_WndShow = &m_DatosRegulador;
	}

	*pResult = 0;
}
```

Se habilita una función general, que mediante pulsación, cierra la base de datos, esté en la pantalla que se esté de la misma, al ser una función asociada a la vista o clase principal.

```
void BaseDatosGeneral::OnBnClickedOkgeneral()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	EndDialog(NULL);
}
```

Cada una de las clases que lanzan o permiten gestionar las vistas de la base de datos se crea de manera análoga, con las mismas funciones, en las cuales únicamente se particulariza la información a mostrar, pero las actuaciones son las mismas, siendo por tanto que solo se explica una de ellas, la correspondiente a la primera pestaña.

Según lo anteriormente comentado, cada una de las pestañas, a pesar de estar integradas en la vista de una clase superior, están formadas por su propia clase, admitiendo las funciones de inicialización, siendo necesario su constructor, destructor, declaraciones variables miembro y/o globales, y en general las mismas estructura que si de una clase normal se tratase. Por tal motivo, a pesar de haberse utilizado la función *OnInitDialog()* en la clase base de la hoja de propiedades, dentro de la función de la vista concreta del cuadro a visualizar (en este ejemplo panel solares), se vuelve a invocar a dicha función, pero esta vez, no para predefinir estructuras generales, sino los encabezados y tamaños de las columnas de la hoja o control de lista empotrado en la misma (descrito en memoria).

```
BOOL BaseDatosPanel::OnInitDialog()
{
	CDialogEx::OnInitDialog();

	// TODO:  Agregue aquí la inicialización adicional
	//se inician las columnas del control de lista de base datos panles
	m_DatosSalidaPaneles.InsertColumn(0, _T("Descripción"), LVCFMT_LEFT, 310);
	m_DatosSalidaPaneles.InsertColumn(1, _T("Modelo"), LVCFMT_CENTER, 60);
	m_DatosSalidaPaneles.InsertColumn(2, _T("Potencia (W)"), LVCFMT_CENTER,
	80);
	m_DatosSalidaPaneles.InsertColumn(3, _T("Icc (A)"), LVCFMT_CENTER, 60);
	m_DatosSalidaPaneles.InsertColumn(4, _T("Imp (A)"), LVCFMT_CENTER, 60);
	m_DatosSalidaPaneles.InsertColumn(5, _T("Vmp (V)"), LVCFMT_CENTER, 60);
	m_DatosSalidaPaneles.InsertColumn(6, _T("Vca (V)"), LVCFMT_CENTER, 60);
	m_DatosSalidaPaneles.InsertColumn(7, _T("Base (m)"), LVCFMT_CENTER, 60);
	m_DatosSalidaPaneles.InsertColumn(8, _T("Altura (m)"), LVCFMT_CENTER, 60);
	m_DatosSalidaPaneles.InsertColumn(9, _T("Peso (Kg)"), LVCFMT_CENTER, 60);
	m_DatosSalidaPaneles.InsertColumn(10, _T("Precio (€)"), LVCFMT_CENTER,
	60);
```

En la propia inicialización, el panel o control de lista, debe acceder a la base de datos guardada en disco duro de manera permanente, a fin de mostrar en la pantalla, los elementos que estén guardado en cada vista de la base (estos cambiarán en función de las pestaña que se esté consultando, siendo que cada vista mostrara los específicos de la misma.

```
	//lee archivo de datos guardados de la base de datos
	CString GuardadoPaneles = _T("BaseDatosPaneles");
```

```
CFile BasePaneles;
CFileStatus EstadoFi;

if (!BasePaneles.GetStatus(GuardadoPaneles, EstadoFi)) //si no existe
archivo lo crea
{
	BasePaneles.Open(GuardadoPaneles, CFile::modeCreate |
	CFile::modeRead);
}

else
{
	BasePaneles.Open(GuardadoPaneles, CFile::modeRead); //si existe lo
	abre para lectura
}
```

Tal acción conllevará el deserializado de los datos contenidos en el archivo de disco, recorriendo mediante un bucle *While* todos ellos, e instándolo en la fila correspondiente (al igual que se ha explicado en apartados previos correspondientes a seriación y deseriacion, concretamente en controles de lista), cargándoselo en n la variable miembro de la vista y mostrando su contenido mediante la función *SetItemText*.

```
CArchive ar(&BasePaneles, CArchive::load);

ar >> ElementoPanel;// extrae el número de elementos de la lista
ElementoPanel = ElementoPanel - 1;
//realiza un bucle de desseriacion de los datos guardados para cada linea
de la base de datos
//hasta que se hayan recorrido todos los elementos guardados
while (ElementoPanel > -1)   // mientras haya objetos en la lista
{
	ar >> DescripcionBase >> ModeloPanelBase >> PotPanelBase >>
	ICCPanelBase >> IMPPanelBase >> VMPPanelBase >> VCAPanelBase >>
	BasePanelBase >> AltoPanelBase >> PesoPanelBase >> CostePanelBase;

	int nItem;
	nItem = m_DatosSalidaPaneles.InsertItem(0, DescripcionBase);

	// insertar en cada columna de una misma linea cada uno de los
	datos de los elementos
	m_DatosSalidaPaneles.SetItemText(nItem, 0, DescripcionBase);
	m_DatosSalidaPaneles.SetItemText(nItem, 1, ModeloPanelBase);
	m_DatosSalidaPaneles.SetItemText(nItem, 2, PotPanelBase);
	m_DatosSalidaPaneles.SetItemText(nItem, 3, ICCPanelBase);
	m_DatosSalidaPaneles.SetItemText(nItem, 4, IMPPanelBase);
	m_DatosSalidaPaneles.SetItemText(nItem, 5, VMPPanelBase);
	m_DatosSalidaPaneles.SetItemText(nItem, 6, VCAPanelBase);
	m_DatosSalidaPaneles.SetItemText(nItem, 7, BasePanelBase);
	m_DatosSalidaPaneles.SetItemText(nItem, 8, AltoPanelBase);
	m_DatosSalidaPaneles.SetItemText(nItem, 9, PesoPanelBase);
	m_DatosSalidaPaneles.SetItemText(nItem, 10, CostePanelBase);

//actualizar variables salida
UpdateData(FALSE);

	ElementoPanel = ElementoPanel - 1; //contador de elementos
	pendientes
}
```

```
	UpdateData(FALSE);
	ar.Close(); //cierra archivo de seriación

	BasePaneles.Close(); //cierra archivo datos paneles

	return TRUE;  // return TRUE unless you set the focus to a control
	 // EXCEPCIÓN: las páginas de propiedades OCX deben devolver FALSE
}
```

Se agrega la presente función, para que el usuario pueda introducir nuevos datos en la base mediante la pulsación del botón “añadir”, siendo que este invocara a un cuadro de dialogo, completado con las casillas necesaria s(cuadros de edición), para que el usuario introduzca los datos que se deben mostrar en la vista de la base de datos, retornado las variables miembro capturadas en cuada cuadro en el cierre de dicho cuadro modal (no se entra a explicar dada la similitud con otros cuadros de introducción de datos en controles de lista explicados).

```
void BaseDatosPanel::OnBnClickedAnadirpaneles()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	IntroDatosPaneles DlgIntroDatosPaneles(NULL);    //Llama al constructor
	del dialogo para introducir puntos consumo

	if (DlgIntroDatosPaneles.DoModal() == IDOK)

	{

	}
```

Al igual que en situaciones previas, el usuario introduce, muchas veces, en las casillas de información datos números, siendo que los controles de lista, como el presente, solo admiten salida en formato texto, siendo necesaria una conversión de tales datos, para que se puedan mostrar. De manera análoga la recuperación de la información (no activada en la presente base de datos), exigiría la conversión inversa para que los datos fueran aprovechables.

```
	if (DatoPanel)
	{
		//añadir fila al control de lista
		int nItem;

		nItem = m_DatosSalidaPaneles.InsertItem(0, DescripcionBaseAux);

		//dar formato a los archivos numericos para salir en pantalla como
		texto
		CString q, w, e, r, t, y, u, i, o;

		q.Format(_T("%f"), PotPanelBaseAux);
		w.Format(_T("%f"), ICCPanelBaseAux);
		e.Format(_T("%f"), IMPPanelBaseAux);
		r.Format(_T("%f"), VMPPanelBaseAux);
		t.Format(_T("%f"), VCAPanelBaseAux);
		y.Format(_T("%f"), BasePanelBaseAux);
		u.Format(_T("%f"), AltoPanelBaseAux);
```

```
            i.Format(_T("%f"), PesoPanelBaseAux);
            o.Format(_T("%f"), CostePanelBaseAux);

            // insertar en cada columna de una misma linea cada uno de los
            datos de los elementos
            m_DatosSalidaPaneles.SetItemText(nItem, 0, DescripcionBaseAux);
            m_DatosSalidaPaneles.SetItemText(nItem, 1, ModeloPanelBaseAux);
            m_DatosSalidaPaneles.SetItemText(nItem, 2, q);
            m_DatosSalidaPaneles.SetItemText(nItem, 3, w);
            m_DatosSalidaPaneles.SetItemText(nItem, 4, e);
            m_DatosSalidaPaneles.SetItemText(nItem, 5, r);
            m_DatosSalidaPaneles.SetItemText(nItem, 6, t);
            m_DatosSalidaPaneles.SetItemText(nItem, 7, y);
            m_DatosSalidaPaneles.SetItemText(nItem, 8, u);
            m_DatosSalidaPaneles.SetItemText(nItem, 9, i);
            m_DatosSalidaPaneles.SetItemText(nItem, 10, o);
      }
      //actualizar variables salida
      UpdateData(TRUE);
}
```

Para borrar cualquier elemento de la vista de la base de datos, bastara con seleccionarlo mediante le ratón, y pulsar el botón eliminar, siendo que este elemento, desaparecerá de la vista medina tela función DeleteItem(), resultando que cuando se guarde al salir, quedará definitivamente eliminado del archivo de disco, dado que los datos se sobrescribirán con la nueva información que este reflejada en tal momento en pantalla.

```
void BaseDatosPanel::OnBnClickedBorrarpaneles()
{
      // TODO: Agregue aquí su código de controlador de notificación de control
      // borrar el elemento de la lista
      m_DatosSalidaPaneles.DeleteItem(Elemento);

      //actualizar variables salida
      UpdateData(TRUE);
}
```

El guardado de elementos de la base de datos, bien sean en este caso los panales, o los de cualquier otra vista, se ejecutará mediante pulsación de botón especifico, en el cual se procederá a lanzar un proceso de seriación, similar al explicado previamente y extensivamente en la memoria del presente proyecto.

```
void BaseDatosPanel::OnBnClickedGuardarpanel()
{
      // TODO: Agregue aquí su código de controlador de notificación de control

      this->UpdateData();

      //obtención del numero de elementos o referencia lista elementos
      ElementoPanel = m_DatosSalidaPaneles.GetItemCount();
      //SeriacionItemPan(); //almacena el número de elementos existentes

      CString GuardadoPaneles = _T("BaseDatosPaneles");

            //abrir el fichero para escribir
            CFile BasePaneles;
```

```
            //estructura para almacenar el estado del fichero
            BasePaneles.Open(GuardadoPaneles, CFile::modeCreate |
            CFile::modeWrite );

            //bucle de carga y seriacin de cada una de las lineas de la base de
            datos hasta que se terminen
            CArchive ar(&BasePaneles, CArchive::store);

            ar << ElementoPanel; //carga como primer elemento el numero de
            elemntos de la lista
            ElementoPanel = ElementoPanel - 1; //resta uno para recuperar datos
            del valor 0;
            while (ElementoPanel > -1)   // mientras haya objetos en la lista
            {
            //Recuperaicon de los valores de los elementos de lista en formato
            texto
                  DescripcionBase =
                  m_DatosSalidaPaneles.GetItemText(ElementoPanel, 0);
                  ModeloPanelBase =
                  m_DatosSalidaPaneles.GetItemText(ElementoPanel, 1);
                  PotPanelBase =
                  m_DatosSalidaPaneles.GetItemText(ElementoPanel, 2);
                  ICCPanelBase =
                  m_DatosSalidaPaneles.GetItemText(ElementoPanel, 3);
                  IMPPanelBase =
                  m_DatosSalidaPaneles.GetItemText(ElementoPanel, 4);
                  VMPPanelBase =
                  m_DatosSalidaPaneles.GetItemText(ElementoPanel, 5);
                  VCAPanelBase =
                  m_DatosSalidaPaneles.GetItemText(ElementoPanel, 6);
                  BasePanelBase =
                  m_DatosSalidaPaneles.GetItemText(ElementoPanel, 7);
                  AltoPanelBase =
                  m_DatosSalidaPaneles.GetItemText(ElementoPanel, 8);
                  PesoPanelBase =
                  m_DatosSalidaPaneles.GetItemText(ElementoPanel, 9);
                  CostePanelBase =
                  m_DatosSalidaPaneles.GetItemText(ElementoPanel, 10);

                  ElementoPanel = ElementoPanel - 1;

                  ar << DescripcionBase << ModeloPanelBase << PotPanelBase <<
                  ICCPanelBase << IMPPanelBase << VMPPanelBase << VCAPanelBase
                  <<BasePanelBase << AltoPanelBase << PesoPanelBase <<
                  CostePanelBase;
            }

            //cierre seriacin y archivo datos
            ar.Close();
            BasePaneles.Close();

      MessageBox(_T("Archivo base datos guardado correctamente"),
      _T("Base Datos Paneles Fotovoltaicos"), MB_ICONINFORMATION);
}
```

La presente función, se encamina exclusivamente a capturar la referencia del elemento que esta pulsado o enfocado por el usuario en la vista del cuadro o control de lista. Mediante esta captura, se podría conocer el elemento a fin de operar sobre él, bien

sea borrado, o en su caso recuperación de datos o cualquier otro tipo de operación sobre él.

```
void BaseDatosPanel::OnNMClickDatospaneles(NMHDR *pNMHDR, LRESULT *pResult)
{
	LPNMITEMACTIVATE pNMItemActivate =
	reinterpret_cast<LPNMITEMACTIVATE>(pNMHDR);
	// TODO: Agregue aquí su código de controlador de notificación de control

	NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;

	//obtención del numero de elemento o referencia lista
	Elemento = m_DatosSalidaPaneles.GetHotItem();

	// en caso de pensar en utilizar datos de base habría que recuperarlo aqui

	*pResult = 0;
}
```

Mediante la opción de pulsación de botón, el usuario puede solicitar un listado en archivo de texto manipulable de la base de datos que este consultando, no explicándose la generación de la misma o procedo interno del código, dado que se ha explicado en memoria de proyecto en aparado de salida de datos.

```
void BaseDatosPanel::OnBnClickedButton1()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	//limpia el documento preexistente antes de guardar actual
	ofstream BaseDatosPaneles;
	BaseDatosPaneles.open("BaseDatosPaneles.txt", ios::out);
	BaseDatosPaneles.close();

	//obtención del numero de elementos o referencia lista elementos
	ElementoPanel = m_DatosSalidaPaneles.GetItemCount();
	ElementoPanel = ElementoPanel- 1; //resta uno para recuperar datos del
	valor 0;

	//crea el archivo TXT en el que se va a generar el archivo
	BaseDatosPaneles.open("BaseDatosPaneles.txt", ios::out |ios::in |
	ios::app); //archivo que escribe secuencial

	//salidas de texto de puntos consumo
	BaseDatosPaneles <<
"************************************************************\n";
	BaseDatosPaneles << "********** BASE DE DATOS DE PANELES FOTOVOLTAICOS
**********\n";
	BaseDatosPaneles <<
"************************************************************\n";
	BaseDatosPaneles << "\n";
	BaseDatosPaneles << "Descripción:" << "                              " <<
"Modelo" << "      " << "Potencia" << "      " << "Icc" << "      " << "Imp" << "
" << "Vmp" << "      " << "Vca" << "      " << "Base" << "      " << "Altura" <<
"      " << "Peso" << "      " << "Precio""\n";

	while (ElementoPanel > -1)   // mientras haya objetos en la lista
	{
	//Recuperaicon de los valores de los elementos de lista en formato texto
	DescripcionBase = m_DatosSalidaPaneles.GetItemText(ElementoPanel, 0);
```

```
        ModeloPanelBase = m_DatosSalidaPaneles.GetItemText(ElementoPanel, 1);
        PotPanelBase = m_DatosSalidaPaneles.GetItemText(ElementoPanel, 2);
        ICCPanelBase = m_DatosSalidaPaneles.GetItemText(ElementoPanel, 3);
        IMPPanelBase = m_DatosSalidaPaneles.GetItemText(ElementoPanel, 4);
        VMPPanelBase = m_DatosSalidaPaneles.GetItemText(ElementoPanel, 5);
        VCAPanelBase = m_DatosSalidaPaneles.GetItemText(ElementoPanel, 6);
        BasePanelBase = m_DatosSalidaPaneles.GetItemText(ElementoPanel, 7);
        AltoPanelBase = m_DatosSalidaPaneles.GetItemText(ElementoPanel, 8);
        PesoPanelBase = m_DatosSalidaPaneles.GetItemText(ElementoPanel, 9);
        CostePanelBase = m_DatosSalidaPaneles.GetItemText(ElementoPanel, 10);

        ElementoPanel = ElementoPanel - 1;

        //cargar en archivo texto
        int Descrip = DescripcionBase.GetLength() * sizeof(TCHAR);
        int Mod = ModeloPanelBase.GetLength() * sizeof(TCHAR);
        int Pot = PotPanelBase.GetLength() * sizeof(TCHAR);
        int ICC = ICCPanelBase.GetLength() * sizeof(TCHAR);
        int IMPP = IMPPanelBase.GetLength() * sizeof(TCHAR);
        int VMP = VMPPanelBase.GetLength() * sizeof(TCHAR);
        int VCA = VCAPanelBase.GetLength() * sizeof(TCHAR);
        int BPan = BasePanelBase.GetLength() * sizeof(TCHAR);
        int APan = AltoPanelBase.GetLength() * sizeof(TCHAR);
        int PesoPan = PesoPanelBase.GetLength() * sizeof(TCHAR);
        int CostePan = CostePanelBase.GetLength() * sizeof(TCHAR);

        BaseDatosPaneles.write((LPCSTR)(LPCTSTR)DescripcionBase, Descrip);
        BaseDatosPaneles << "           ";
        BaseDatosPaneles.write((LPCSTR)(LPCTSTR)ModeloPanelBase, Mod);
        BaseDatosPaneles << "           ";
        BaseDatosPaneles.write((LPCSTR)(LPCTSTR)PotPanelBase, Pot);
        BaseDatosPaneles << "           ";
        BaseDatosPaneles.write((LPCSTR)(LPCTSTR)ICCPanelBase, ICC);
        BaseDatosPaneles << "           ";
        BaseDatosPaneles.write((LPCSTR)(LPCTSTR)IMPPanelBase, IMPP);
        BaseDatosPaneles << "           ";
        BaseDatosPaneles.write((LPCSTR)(LPCTSTR)VMPPanelBase, VMP);
        BaseDatosPaneles << "           ";
        BaseDatosPaneles.write((LPCSTR)(LPCTSTR)VCAPanelBase, VCA);
        BaseDatosPaneles << "           ";
        BaseDatosPaneles.write((LPCSTR)(LPCTSTR)BasePanelBase, BPan);
        BaseDatosPaneles << "           ";
        BaseDatosPaneles.write((LPCSTR)(LPCTSTR)AltoPanelBase, APan);
        BaseDatosPaneles << "           ";
        BaseDatosPaneles.write((LPCSTR)(LPCTSTR)PesoPanelBase, PesoPan);
        BaseDatosPaneles << "           ";
        BaseDatosPaneles.write((LPCSTR)(LPCTSTR)CostePanelBase, CostePan);
        BaseDatosPaneles << "\n";
        BaseDatosPaneles << "\n";
        }

        // cierre de archico de salida
        BaseDatosPaneles.close();

        ShellExecute(NULL, _T("open"), _T("BaseDatosPaneles.txt"), NULL, NULL,
        SW_SHOWNORMAL);
}
```

6.- Cálculo de perdidas

Mencion especial se hará al sistema de cálculo de perdidas por sombreo, al constituir este en si un módulo independiente constituido por varias clases independientes para su funcionamiento.

El cálculo de pérdidas por sombras, según pliego técnico especifico, responderá a una seria de cuadro so tablas base, de la cual se escogerá la de características más aproximadas a la realizada d dela instalación generada con la aplicación. Según esto, la interface principal para este cálculo, será un cuadro de dialogo que permitirá escoger al usuario una de esta tablas concretas mediante pulsación de botón.

En una primera función, se implementará la apertura de un cuadro de dialogo, que se encamina únicamente a mostrar un control de imagen, para que el usuario pueda tener disponible, en cualquier r momento, el diagrama que corresponde a las posiciones solares, con la numeración o referencias del mismo.

```
void CalcPerdSombra::OnBnClickedButton1()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	PerdSombrasGraf DlgPerdSombrasGraf(NULL); //Llama al constructor del
dialogo para lanzar la pantalla

	if (DlgPerdSombrasGraf.DoModal() == IDOK)
	{

	}
}
```

La totalidad del cuadro de dialogo, permite al usuario lanzar cuadros de dialogo modales, cada uno correspondiendo a cada una de las tablas de las que se dispone para seleccionar los módulos de pérdidas. Según esto mediante una herramienta da botón o pulsación, se lanzara el cuadro de dialogo modal (del que se espera una respuesta en vuelta o cierre del mismo), mediante el cual el usuario tendrá acceso a los datos números de cada tabal (que corresponden a las pérdidas de sol en cada posición e inclinación). Como se ha indicado, el cierre del cuadro citado, devolverá un valor, que se mostrará en un cuadro de edición y cargará en las casillas de pérdidas correspondientes.

```
void CalcPerdSombra::OnBnClickedV1()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	TablaV1 DlgTablaV1(NULL); //Llama al constructor del dialogo para lanzar
la pantalla

	if (DlgTablaV1.DoModal() == IDOK)
	{
		m_PerdSombras = PerdSombrasAux;
		UpdateData(FALSE);
	}
}
```

```
void CalcPerdSombra::OnBnClickedV2()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	TablaV2 DlgTablaV2(NULL); //Llama al constructor del dialogo para lanzar
	la pantalla

	if (DlgTablaV2.DoModal() == IDOK)
	{
		m_PerdSombras = PerdSombrasAux;
		UpdateData(FALSE);
	}
}


void CalcPerdSombra::OnBnClickedV3()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	TablaV3 DlgTablaV3(NULL); //Llama al constructor del dialogo para lanzar
la pantalla

	if (DlgTablaV3.DoModal() == IDOK)
	{
		m_PerdSombras = PerdSombrasAux;
		UpdateData(FALSE);
	}
}


void CalcPerdSombra::OnBnClickedV4()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	TablaV4 DlgTablaV4(NULL); //Llama al constructor del dialogo para lanzar
la pantalla

	if (DlgTablaV4.DoModal() == IDOK)
	{
		m_PerdSombras = PerdSombrasAux;
		UpdateData(FALSE);
	}
}


void CalcPerdSombra::OnBnClickedV5()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	TablaV5 DlgTablaV5(NULL); //Llama al constructor del dialogo para lanzar
la pantalla

	if (DlgTablaV5.DoModal() == IDOK)
	{
		m_PerdSombras = PerdSombrasAux;
		UpdateData(FALSE);
	}
}


void CalcPerdSombra::OnBnClickedV6()
```

```
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	TablaV6 DlgTablaV6(NULL); //Llama al constructor del dialogo para lanzar
la pantalla

	if (DlgTablaV6.DoModal() == IDOK)
	{
		m_PerdSombras = PerdSombrasAux;
		UpdateData(FALSE);
	}
}


void CalcPerdSombra::OnBnClickedV7()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	TablaV7 DlgTablaV7(NULL); //Llama al constructor del dialogo para lanzar
la pantalla

	if (DlgTablaV7.DoModal() == IDOK)
	{
		m_PerdSombras = PerdSombrasAux;
		UpdateData(FALSE);
	}
}


void CalcPerdSombra::OnBnClickedV8()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	TablaV8 DlgTablaV8(NULL); //Llama al constructor del dialogo para lanzar
la pantalla

	if (DlgTablaV8.DoModal() == IDOK)
	{
		m_PerdSombras = PerdSombrasAux;
		UpdateData(FALSE);
	}
}


void CalcPerdSombra::OnBnClickedV9()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	TablaV9 DlgTablaV9(NULL); //Llama al constructor del dialogo para lanzar
la pantalla

	if (DlgTablaV9.DoModal() == IDOK)
	{
		m_PerdSombras = PerdSombrasAux;
		UpdateData(FALSE);
	}
}


void CalcPerdSombra::OnBnClickedV10()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	TablaV10 DlgTablaV10(NULL); //Llama al constructor del dialogo para lanzar
la pantalla
```

```
	if (DlgTablaV10.DoModal() == IDOK)
	{
		m_PerdSombras = PerdSombrasAux;
		UpdateData(FALSE);
	}
}


void CalcPerdSombra::OnBnClickedV11()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	TablaV11 DlgTablaV11(NULL); //Llama al constructor del dialogo para lanzar
la pantalla

	if (DlgTablaV11.DoModal() == IDOK)
	{
		m_PerdSombras = PerdSombrasAux;
		UpdateData(FALSE);
	}
}
```

6.1.- Selección valores

En el interior de cada uno de los cuadros de dialogo lanzados en la fase previa de cálculo, tal como se ha indicado, se mostrarán los valores numéricos de perdidas seleccionables a cada casilla o tramo de perdidas correspondiente el diagrama respectivo, pudiéndose seleccionar visualmente de estas tablas los valores de pérdidas que corresponde a la instalación en calculo. En el presente anexó solo se muestra la tabla 1, siendo el funcionamiento e implementación de todas las restantes, idéntico, pero cada una con sus valores numéricos pertinentes y diferenciados.

```
// TablaV1.cpp: archivo de implementación
// Controladores de mensajes de TablaV1
```

La tabla que se despliega, muestra múltiples casillas, cada una implementada como un control de pulsación, de manera que si el usuario pulsa la fila y columna correspondiente, se actúe generando el valor número que afecta a la perdida de tal casilla. Esta actuación, entrará directamente en el código correspondiente a la casilla afectada.

```
void TablaV1::OnBnClickedA1()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
```

Cuando el usuario pulsa un botón de la tabla correspondiente a una pérdida, puede darse el caso, de que sea la primera vez que lo pulsa, o que el botón estuviera previamente pulsado. Esto es, que el valor de la tabla, estuviera seleccionado o no. Según esto si el valor no estaba seleccionado, se capturará el valor numérico de pérdidas de esa casilla y se almacenará, y se cambiará el texto reflejado en ese botón (que inicialmente es el valor numérico de la perdida), para mostrar "XXXX", mediante la función "*SetWindowText*", y que así el usuario identifique que esa casilla esta seleccionada.

```
	if (conexionA1 == true)
	{
		GetDlgItem(IDC_A1)->SetWindowText(_TEXT("XXXX"));
		conexionA1 = false;
		ValorA1 = 0.00;
```

Al seleccionar el valor numérico de la casilla correspondiente, se invocará a una cuadro de dialogo asociado, en el cual únicamente se seleccionara el porcentaje de sombreo que afecta a esta casilla, desde el 0 al 100%, devolviendo el valor de tal porcentaje, de manera que la pérdida absoluta de la casilla seleccionada y por tanto el valor devuelto (calculado en otra sección de este archivo), será el producto del valor de pérdidas de la casilla, por el porcentaje de influencia u aculatan de este.

```
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
		% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenA1 = VSombraAux;
		}
		VSombraAux = 0;
	}
```

Por el contrario, si cuando el usuario actúa sobre una casilla, esta estaba previamente marcada, se entrara en la sección de código que cambia el texto “XXXX”, por el valor numérico original de la casilla, deseleccionado la misma, y cargando su valor de pérdida a 0, para que en el cálculo no compute entidad alguna.

```
	else if (conexionA1 == false)
	{
		GetDlgItem(IDC_A1)->SetWindowText(_TEXT("0.00"));
		conexionA1 = true;
		ValorA1 = 0.00;
		PorcenA1 = 0.00;
	}
}
```

Tal proceso de actuación, será repetido para cada una de las casillas que el usuario pueda seleccionar dentro de una tabla de pérdidas por sombreo.

```
void TablaV1::OnBnClickedA2()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionA2 == true)
	{
		GetDlgItem(IDC_A2)->SetWindowText(_TEXT("XXXX"));
		conexionA2 = false;
		ValorA2 = 0.00;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenA2 = VSombraAux;
		}
```

```
			VSombraAux = 0;
		}

		else if (conexionA2 == false)
		{
			GetDlgItem(IDC_A2)->SetWindowText(_TEXT("0.00"));
			conexionA2 = true;
			ValorA2 = 0.00;
			PorcenA2 = 0.00;
		}
}


void TablaV1::OnBnClickedA3()
{
		// TODO: Agregue aquí su código de controlador de notificación de control

		if (conexionA3 == true)
		{
			GetDlgItem(IDC_A3)->SetWindowText(_TEXT("XXXX"));
			conexionA3 = false;
			ValorA3 = 0.13;
			Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

			if (DlgOcultacion.DoModal() == IDOK)
			{
				PorcenA3 = VSombraAux;
			}
			VSombraAux = 0;
		}

		else if (conexionA3 == false)
		{
			GetDlgItem(IDC_A3)->SetWindowText(_TEXT("0.13"));
			conexionA3 = true;
			ValorA3 = 0.00;
			PorcenA3 = 0.00;
		}
}
void TablaV1::OnBnClickedA4()
{
		// TODO: Agregue aquí su código de controlador de notificación de control

		if (conexionA4 == true)
		{
			GetDlgItem(IDC_A4)->SetWindowText(_TEXT("XXXX"));
			conexionA4 = false;
			ValorA4 = 1.00;
			Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

			if (DlgOcultacion.DoModal() == IDOK)
			{
				PorcenA4 = VSombraAux;
			}
			VSombraAux = 0;
		}

		else if (conexionA4 == false)
```

```
	{
		GetDlgItem(IDC_A4)->SetWindowText(_TEXT("1.00"));
		conexionA4 = true;
		ValorA4 = 0.00;
		PorcenA4 = 0.00;
	}
}


void TablaV1::OnBnClickedA5()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionA5 == true)
	{
		GetDlgItem(IDC_A5)->SetWindowText(_TEXT("XXXX"));
		conexionA5 = false;
		ValorA5 = 1.84;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenA5 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionA5 == false)
	{
		GetDlgItem(IDC_A5)->SetWindowText(_TEXT("1.84"));
		conexionA5 = true;
		ValorA5 = 0.00;
		PorcenA5 = 0.00;
	}
}


void TablaV1::OnBnClickedA6()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionA6 == true)
	{
		GetDlgItem(IDC_A6)->SetWindowText(_TEXT("XXXX"));
		conexionA6 = false;
		ValorA6 = 2.70;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenA6 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionA6 == false)
	{
		GetDlgItem(IDC_A6)->SetWindowText(_TEXT("2.70"));
```

```
			conexionA6 = true;
			ValorA6 = 0.00;
			PorcenA6 = 0.00;
		}
}


void TablaV1::OnBnClickedA7()
{
		// TODO: Agregue aquí su código de controlador de notificación de control

		if (conexionA7 == true)
		{
			GetDlgItem(IDC_A7)->SetWindowText(_TEXT("XXXX"));
			conexionA7 = false;
			ValorA7 = 3.15;
			Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

			if (DlgOcultacion.DoModal() == IDOK)
			{
				PorcenA7 = VSombraAux;
			}
			VSombraAux = 0;
		}

		else if (conexionA7 == false)
		{
			GetDlgItem(IDC_A7)->SetWindowText(_TEXT("3.15"));
			conexionA7 = true;
			ValorA7 = 0.00;
			PorcenA7 = 0.00;
		}
}


void TablaV1::OnBnClickedA8()
{
		// TODO: Agregue aquí su código de controlador de notificación de control

		if (conexionA8 == true)
		{
			GetDlgItem(IDC_A8)->SetWindowText(_TEXT("XXXX"));
			conexionA8 = false;
			ValorA8 = 3.17;
			Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

			if (DlgOcultacion.DoModal() == IDOK)
			{
				PorcenA8 = VSombraAux;
			}
			VSombraAux = 0;
		}

		else if (conexionA8 == false)
		{
			GetDlgItem(IDC_A8)->SetWindowText(_TEXT("3.17"));
			conexionA8 = true;
			ValorA8 = 0.00;
```

```
		PorcenA8 = 0.00;
	}
}


void TablaV1::OnBnClickedA9()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionA9 == true)
	{
		GetDlgItem(IDC_A9)->SetWindowText(_TEXT("XXXX"));
		conexionA9 = false;
		ValorA9 = 2.70;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenA9 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionA9 == false)
	{
		GetDlgItem(IDC_A9)->SetWindowText(_TEXT("2.7"));
		conexionA9 = true;
		ValorA9 = 0.00;
		PorcenA9 = 0.00;
	}
}


void TablaV1::OnBnClickedA10()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionA10 == true)
	{
		GetDlgItem(IDC_A10)->SetWindowText(_TEXT("XXXX"));
		conexionA10 = false;
		ValorA10 = 1.79;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenA10 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionA10 == false)
	{
		GetDlgItem(IDC_A10)->SetWindowText(_TEXT("1.79"));
		conexionA10 = true;
		ValorA10 = 0.00;
		PorcenA10 = 0.00;
	}
```

```
}
oid TablaV1::OnBnClickedA11()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionA11 == true)
	{
		GetDlgItem(IDC_A11)->SetWindowText(_TEXT("XXXX"));
		conexionA11 = false;
		ValorA11 = 0.98;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenA11 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionA11 == false)
	{
		GetDlgItem(IDC_A11)->SetWindowText(_TEXT("0.98"));
		conexionA11 = true;
		ValorA11 = 0.00;
		PorcenA11 = 0.00;
	}
}


void TablaV1::OnBnClickedA12()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionA12 == true)
	{
		GetDlgItem(IDC_A12)->SetWindowText(_TEXT("XXXX"));
		conexionA12 = false;
		ValorA12 = 0.11;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenA12 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionA12 == false)
	{
		GetDlgItem(IDC_A12)->SetWindowText(_TEXT("0.11"));
		conexionA12 = true;
		ValorA12 = 0.00;
		PorcenA12 = 0.00;

	}
}
```

```
void TablaV1::OnBnClickedA13()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionA13 == true)
	{
		GetDlgItem(IDC_A13)->SetWindowText(_TEXT("XXXX"));
		conexionA13 = false;
		ValorA13 = 0.00;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenA13 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionA13 == false)
	{
		GetDlgItem(IDC_A13)->SetWindowText(_TEXT("0.00"));
		conexionA13 = true;
		ValorA13 = 0.00;
		PorcenA13 = 0.00;
	}
}


void TablaV1::OnBnClickedA14()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionA14 == true)
	{
		GetDlgItem(IDC_A14)->SetWindowText(_TEXT("XXXX"));
		conexionA14 = false;
		ValorA14 = 0.00;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenA14 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionA14 == false)
	{
		GetDlgItem(IDC_A14)->SetWindowText(_TEXT("0.00"));
		conexionA14 = true;
		ValorA14 = 0.00;
		PorcenA14 = 0.00;
	}
}

void TablaV1::OnBnClickedB1()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
```

```
	if (conexionB1 == true)
	{
		GetDlgItem(IDC_B1)->SetWindowText(_TEXT("XXXX"));
		conexionB1 = false;
		ValorB1 = 0.00;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenB1 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionB1 == false)
	{
		GetDlgItem(IDC_B1)->SetWindowText(_TEXT("0.00"));
		conexionB1 = true;
		ValorB1 = 0.00;
		PorcenB1 = 0.00;
	}
}


void TablaV1::OnBnClickedB2()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionB2 == true)
	{
		GetDlgItem(IDC_B2)->SetWindowText(_TEXT("XXXX"));
		conexionB2 = false;
		ValorB2 = 0.01;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenB2 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionB2 == false)
	{
		GetDlgItem(IDC_B2)->SetWindowText(_TEXT("0.01"));
		conexionB2 = true;
		ValorB2 = 0.00;
		PorcenB2 = 0.00;
	}
}


void TablaV1::OnBnClickedB3()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionB3 == true)
	{
```

```
		GetDlgItem(IDC_B3)->SetWindowText(_TEXT("XXXX"));
		conexionB3 = false;
		ValorB3 = 0.41;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenB3 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionB3 == false)
	{
		GetDlgItem(IDC_B3)->SetWindowText(_TEXT("0.41"));
		conexionB3 = true;
		ValorB3 = 0.00;
		PorcenB3 = 0.00;
	}
}
void TablaV1::OnBnClickedB4()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionB4 == true)
	{
		GetDlgItem(IDC_B4)->SetWindowText(_TEXT("XXXX"));
		conexionB4 = false;
		ValorB4 = 0.95;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenB4 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionB4 == false)
	{
		GetDlgItem(IDC_B4)->SetWindowText(_TEXT("0.95"));
		conexionB4 = true;
		ValorB4 = 0.00;
		PorcenB4 = 0.00;
	}
}


void TablaV1::OnBnClickedB5()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionB5 == true)
	{
		GetDlgItem(IDC_B5)->SetWindowText(_TEXT("XXXX"));
		conexionB5 = false;
		ValorB5 = 1.50;
```

```
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenB5 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionB5 == false)
	{
		GetDlgItem(IDC_B5)->SetWindowText(_TEXT("1.50"));
		conexionB5 = true;
		ValorB5 = 0.00;
		PorcenB5 = 0.00;
	}
}


void TablaV1::OnBnClickedB6()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionB6 == true)
	{
		GetDlgItem(IDC_B6)->SetWindowText(_TEXT("XXXX"));
		conexionB6 = false;
		ValorB6 = 1.88;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenB6 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionB6 == false)
	{
		GetDlgItem(IDC_B6)->SetWindowText(_TEXT("1.88"));
		conexionB6 = true;
		ValorB6 = 0.00;
		PorcenB6 = 0.00;
	}
}


void TablaV1::OnBnClickedB7()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionB7 == true)
	{
		GetDlgItem(IDC_B7)->SetWindowText(_TEXT("XXXX"));
		conexionB7 = false;
		ValorB7 = 2.12;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra
```

```
		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenB7 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionB7 == false)
	{
		GetDlgItem(IDC_B7)->SetWindowText(_TEXT("2.12"));
		conexionB7 = true;
		ValorB7 = 0.00;
		PorcenB7 = 0.00;
	}
}


void TablaV1::OnBnClickedB8()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionB8 == true)
	{
		GetDlgItem(IDC_B8)->SetWindowText(_TEXT("XXXX"));
		conexionB8 = false;
		ValorB8 = 2.12;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenB8 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionB8 == false)
	{
		GetDlgItem(IDC_B8)->SetWindowText(_TEXT("2.12"));
		conexionB8 = true;
		ValorB8 = 0.00;
		PorcenB8 = 0.00;
	}
}


void TablaV1::OnBnClickedB9()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionB9 == true)
	{
		GetDlgItem(IDC_B9)->SetWindowText(_TEXT("XXXX"));
		conexionB9 = false;
		ValorB9 = 1.89;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
```

```
		{
			PorcenB9 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionB9 == false)
	{
		GetDlgItem(IDC_B9)->SetWindowText(_TEXT("1.89"));
		conexionB9 = true;
		ValorB9 = 0.00;
		PorcenB9 = 0.00;
	}
}


void TablaV1::OnBnClickedB10()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionB10 == true)
	{
		GetDlgItem(IDC_B10)->SetWindowText(_TEXT("XXXX"));
		conexionB10 = false;
		ValorB10 = 1.59;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenB10 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionB10 == false)
	{
		GetDlgItem(IDC_B10)->SetWindowText(_TEXT("1.59"));
		conexionB10 = true;
		ValorB10 = 0.00;
		PorcenB10 = 0.00;
	}
}


void TablaV1::OnBnClickedB11()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionB11 == true)
	{
		GetDlgItem(IDC_B11)->SetWindowText(_TEXT("XXXX"));
		conexionB11 = false;
		ValorB11 = 0.99;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenB11 = VSombraAux;
```

```
		}
		VSombraAux = 0;
	}

	else if (conexionB11 == false)
	{
		GetDlgItem(IDC_B11)->SetWindowText(_TEXT("0.99"));
		conexionB11 = true;
		ValorB11 = 0.00;
		PorcenB11 = 0.00;
	}
}
void TablaV1::OnBnClickedB12()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionB12 == true)
	{
		GetDlgItem(IDC_B12)->SetWindowText(_TEXT("XXXX"));
		conexionB12 = false;
		ValorB12 = 0.42;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenB12 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionB12 == false)
	{
		GetDlgItem(IDC_B12)->SetWindowText(_TEXT("0.42"));
		conexionB12 = true;
		ValorB12 = 0.00;
		PorcenB12 = 0.00;
	}
}


void TablaV1::OnBnClickedB13()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionB13 == true)
	{
		GetDlgItem(IDC_B13)->SetWindowText(_TEXT("XXXX"));
		conexionB13 = false;
		ValorB13 = 0.02;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenB13 = VSombraAux;
		}
		VSombraAux = 0;
	}
```

```
	else if (conexionB13 == false)
	{
		GetDlgItem(IDC_B13)->SetWindowText(_TEXT("0.02"));
		conexionB13 = true;
		ValorB13 = 0.00;
		PorcenB13 = 0.00;
	}
}


void TablaV1::OnBnClickedB14()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionB14 == true)
	{
		GetDlgItem(IDC_B14)->SetWindowText(_TEXT("XXXX"));
		conexionB14 = false;
		ValorB14 = 0.00;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenB14 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionB14 == false)
	{
		GetDlgItem(IDC_B14)->SetWindowText(_TEXT("0.00"));
		conexionB14 = true;
		ValorB14 = 0.00;
		PorcenB14 = 0.00;
	}
}
void TablaV1::OnBnClickedC1()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	if (conexionC1 == true)
	{
		GetDlgItem(IDC_C1)->SetWindowText(_TEXT("XXXX"));
		conexionC1 = false;
		ValorC1 = 0.00;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenC1 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionC1 == false)
	{
		GetDlgItem(IDC_C1)->SetWindowText(_TEXT("0.00"));
		conexionC1 = true;
		ValorC1 = 0.00;
```

```
		PorcenC1 = 0.00;

	}
}


void TablaV1::OnBnClickedC2()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionC2 == true)
	{
		GetDlgItem(IDC_C2)->SetWindowText(_TEXT("XXXX"));
		conexionC2 = false;
		ValorC2 = 0.12; Ocultacion DlgOcultacion(NULL); //Llama al
constructor del dialogo % de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenC2 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionC2 == false)
	{
		GetDlgItem(IDC_C2)->SetWindowText(_TEXT("0.12"));
		conexionC2 = true;
		ValorC2 = 0.00;
		PorcenC2 = 0.00;
	}
}


void TablaV1::OnBnClickedC3()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionC3 == true)
	{
		GetDlgItem(IDC_C3)->SetWindowText(_TEXT("XXXX"));
		conexionC3 = false;
		ValorC3 = 0.62;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenC3 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionC3 == false)
	{
		GetDlgItem(IDC_C3)->SetWindowText(_TEXT("0.62"));
		conexionC3 = true;
		ValorC3 = 0.00;
		PorcenC3 = 0.00;
	}
```

```
}


void TablaV1::OnBnClickedC4()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionC4 == true)
	{
		GetDlgItem(IDC_C4)->SetWindowText(_TEXT("XXXX"));
		conexionC4 = false;
		ValorC4 = 1.27;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenC4 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionC4 == false)
	{
		GetDlgItem(IDC_C4)->SetWindowText(_TEXT("1.27"));
		conexionC4 = true;
		ValorC4 = 0.00;
		PorcenC4 = 0.00;
	}
}
void TablaV1::OnBnClickedC5()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionC5 == true)
	{
		GetDlgItem(IDC_C5)->SetWindowText(_TEXT("XXXX"));
		conexionC5 = false;
		ValorC5 = 1.83;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenC5 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionC5 == false)
	{
		GetDlgItem(IDC_C5)->SetWindowText(_TEXT("1.83"));
		conexionC5 = true;
		ValorC5 = 0.00;
		PorcenC5 = 0.00;
	}
}


void TablaV1::OnBnClickedC6()
```

```
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionC6 == true)
	{
		GetDlgItem(IDC_C6)->SetWindowText(_TEXT("XXXX"));
		conexionC6 = false;
		ValorC6 = 2.21;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenC6 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionC6 == false)
	{
		GetDlgItem(IDC_C6)->SetWindowText(_TEXT("2.21"));
		conexionC6 = true;
		ValorC6 = 0.00;
		PorcenC6 = 0.00;
	}
}


void TablaV1::OnBnClickedC7()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionC7 == true)
	{
		GetDlgItem(IDC_C7)->SetWindowText(_TEXT("XXXX"));
		conexionC7 = false;
		ValorC7 = 2.43;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenC7 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionC7 == false)
	{
		GetDlgItem(IDC_C7)->SetWindowText(_TEXT("2.43"));
		conexionC7 = true;
		ValorC7 = 0.00;
		PorcenC7 = 0.00;
	}
}


void TablaV1::OnBnClickedC8()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
```

```
	if (conexionC8 == true)
	{
		GetDlgItem(IDC_C8)->SetWindowText(_TEXT("XXXX"));
		conexionC8 = false;
		ValorC8 = 2.33;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenC8 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionC8 == false)
	{
		GetDlgItem(IDC_C8)->SetWindowText(_TEXT("2.33"));
		conexionC8 = true;
		ValorC8 = 0.00;
		PorcenC8 = 0.00;
	}
}


void TablaV1::OnBnClickedC9()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionC9 == true)
	{
		GetDlgItem(IDC_C9)->SetWindowText(_TEXT("XXXX"));
		conexionC9 = false;
		ValorC9 = 2.01;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenC9 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionC9 == false)
	{
		GetDlgItem(IDC_C9)->SetWindowText(_TEXT("2.01"));
		conexionC9 = true;
		ValorC9 = 0.00;
		PorcenC9 = 0.00;
	}
}


void TablaV1::OnBnClickedC10()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionC10 == true)
```

```
	{
		GetDlgItem(IDC_C10)->SetWindowText(_TEXT("XXXX"));
		conexionC10 = false;
		ValorC10 = 1.65;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenC10 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionC10 == false)
	{
		GetDlgItem(IDC_C10)->SetWindowText(_TEXT("1.65"));
		conexionC10 = true;
		ValorC10 = 0.00;
		PorcenC10 = 0.00;
	}
}


void TablaV1::OnBnClickedC11()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionC11 == true)
	{
		GetDlgItem(IDC_C11)->SetWindowText(_TEXT("XXXX"));
		conexionC11 = false;
		ValorC11 = 1.08;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenC11 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionC11 == false)
	{
		GetDlgItem(IDC_C11)->SetWindowText(_TEXT("1.08"));
		conexionC11 = true;
		ValorC11 = 0.00;
		PorcenC11 = 0.00;
	}
}
void TablaV1::OnBnClickedC12()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionC12 == true)
	{
		GetDlgItem(IDC_C12)->SetWindowText(_TEXT("XXXX"));
		conexionC12 = false;
		ValorC12 = 0.52;
```

```
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenC12 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionC12 == false)
	{
		GetDlgItem(IDC_C12)->SetWindowText(_TEXT("0.52"));
		conexionC12 = true;
		ValorC12 = 0.00;
		PorcenC12 = 0.00;
	}
}


void TablaV1::OnBnClickedC13()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionC13 == true)
	{
		GetDlgItem(IDC_C13)->SetWindowText(_TEXT("XXXX"));
		conexionC13 = false;
		ValorC13 = 0.10;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenC13 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionC13 == false)
	{
		GetDlgItem(IDC_C13)->SetWindowText(_TEXT("0.10"));
		conexionC13 = true;
		ValorC13 = 0.00;
		PorcenC13 = 0.00;
	}
}


void TablaV1::OnBnClickedC14()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionC14 == true)
	{
		GetDlgItem(IDC_C14)->SetWindowText(_TEXT("XXXX"));
		conexionC14 = false;
		ValorC14 = 0.00;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra
```

```
		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenC14 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionC14 == false)
	{
		GetDlgItem(IDC_C14)->SetWindowText(_TEXT("0.00"));
		conexionC14 = true;
		ValorC14 = 0.00;
		PorcenC14 = 0.00;
	}
}
void TablaV1::OnBnClickedD1()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	if (conexionD1 == true)
	{
		GetDlgItem(IDC_D1)->SetWindowText(_TEXT("XXXX"));
		conexionD1 = false;
		ValorD1 = 0.03;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenD1 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionD1 == false)
	{
		GetDlgItem(IDC_D1)->SetWindowText(_TEXT("0.03"));
		conexionD1 = true;
		ValorD1 = 0.00;
		PorcenD1 = 0.00;
	}
}


void TablaV1::OnBnClickedD2()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionD2 == true)
	{
		GetDlgItem(IDC_D2)->SetWindowText(_TEXT("XXXX"));
		conexionD2 = false;
		ValorD2 = 0.44;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenD2 = VSombraAux;
		}
```

```
		VSombraAux = 0;
	}

	else if (conexionD2 == false)
	{
		GetDlgItem(IDC_D2)->SetWindowText(_TEXT("0.44"));
		conexionD2 = true;
		ValorD2 = 0.00;
		PorcenD2 = 0.00;
	}
}


void TablaV1::OnBnClickedD3()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionD3 == true)
	{
		GetDlgItem(IDC_D3)->SetWindowText(_TEXT("XXXX"));
		conexionD3 = false;
		ValorD3 = 1.49;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenD3 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionD3 == false)
	{
		GetDlgItem(IDC_D3)->SetWindowText(_TEXT("1.49"));
		conexionD3 = true;
		ValorD3 = 0.00;
		PorcenD3 = 0.00;
	}
}


void TablaV1::OnBnClickedD4()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionD4 == true)
	{
		GetDlgItem(IDC_D4)->SetWindowText(_TEXT("XXXX"));
		conexionD4 = false;
		ValorD4 = 2.76;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenD4 = VSombraAux;
		}
		VSombraAux = 0;
	}
```

```
	else if (conexionD4 == false)
	{
		GetDlgItem(IDC_D4)->SetWindowText(_TEXT("2.76"));
		conexionD4 = true;
		ValorD4 = 0.00;
		PorcenD4 = 0.00;
	}
}
void TablaV1::OnBnClickedD5()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionD5 == true)
	{
		GetDlgItem(IDC_D5)->SetWindowText(_TEXT("XXXX"));
		conexionD5 = false;
		ValorD5 = 3.87;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenD5 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionD5 == false)
	{
		GetDlgItem(IDC_D5)->SetWindowText(_TEXT("3.87"));
		conexionD5 = true;
		ValorD5 = 0.00;
		PorcenD5 = 0.00;
	}
}


void TablaV1::OnBnClickedD6()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionD6 == true)
	{
		GetDlgItem(IDC_D6)->SetWindowText(_TEXT("XXXX"));
		conexionD6 = false;
		ValorD6 = 4.67;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenD6 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionD6 == false)
	{
		GetDlgItem(IDC_D6)->SetWindowText(_TEXT("4.67"));
```

```
			conexionD6 = true;
			ValorD6 = 0.00;
			PorcenD6 = 0.00;
		}
}


void TablaV1::OnBnClickedD7()
{
		// TODO: Agregue aquí su código de controlador de notificación de control

		if (conexionD7 == true)
		{
			GetDlgItem(IDC_D7)->SetWindowText(_TEXT("XXXX"));
			conexionD7 = false;
			ValorD7 = 5.04;
			Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

			if (DlgOcultacion.DoModal() == IDOK)
			{
				PorcenD7 = VSombraAux;
			}
			VSombraAux = 0;
		}

		else if (conexionD7 == false)
		{
			GetDlgItem(IDC_D7)->SetWindowText(_TEXT("5.04"));
			conexionD7 = true;
			ValorD7 = 0.00;
			PorcenD7 = 0.00;
		}
}


void TablaV1::OnBnClickedD8()
{
		// TODO: Agregue aquí su código de controlador de notificación de control

		if (conexionD8 == true)
		{
			GetDlgItem(IDC_D8)->SetWindowText(_TEXT("XXXX"));
			conexionD8 = false;
			ValorD8 = 4.99;
			Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

			if (DlgOcultacion.DoModal() == IDOK)
			{
				PorcenD8 = VSombraAux;
			}
			VSombraAux = 0;
		}

		else if (conexionD8 == false)
		{
			GetDlgItem(IDC_D8)->SetWindowText(_TEXT("4.99"));
			conexionD8 = true;
			ValorD8 = 0.00;
```

```
		PorcenD8 = 0.00;
	}
}


void TablaV1::OnBnClickedD9()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionD9 == true)
	{
		GetDlgItem(IDC_D9)->SetWindowText(_TEXT("XXXX"));
		conexionD9 = false;
		ValorD9 = 4.46;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenD9 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionD9 == false)
	{
		GetDlgItem(IDC_D9)->SetWindowText(_TEXT("4.46"));
		conexionD9 = true;
		ValorD9 = 0.00;
		PorcenD9 = 0.00;
	}
}


void TablaV1::OnBnClickedD10()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionD10 == true)
	{
		GetDlgItem(IDC_D10)->SetWindowText(_TEXT("XXXX"));
		conexionD10 = false;
		ValorD10 = 3.63;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenD10 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionD10 == false)
	{
		GetDlgItem(IDC_D10)->SetWindowText(_TEXT("3.63"));
		conexionD10 = true;
		ValorD10 = 0.00;
		PorcenD10 = 0.00;
	}
```

```
}


void TablaV1::OnBnClickedD11()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionD11 == true)
	{
		GetDlgItem(IDC_D11)->SetWindowText(_TEXT("XXXX"));
		conexionD11 = false;
		ValorD11 = 2.55;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenD11 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionD11 == false)
	{
		GetDlgItem(IDC_D11)->SetWindowText(_TEXT("2.55"));
		conexionD11 = true;
		ValorD11 = 0.00;
		PorcenD11 = 0.00;
	}
}


void TablaV1::OnBnClickedD12()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionD12 == true)
	{
		GetDlgItem(IDC_D12)->SetWindowText(_TEXT("XXXX"));
		conexionD12 = false;
		ValorD12 = 1.33;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenD12 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionD12 == false)
	{
		GetDlgItem(IDC_D12)->SetWindowText(_TEXT("1.33"));
		conexionD12 = true;
		ValorD12 = 0.00;
		PorcenD12 = 0.00;
	}
}
void TablaV1::OnBnClickedD13()
```

```
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionD13 == true)
	{
		GetDlgItem(IDC_D13)->SetWindowText(_TEXT("XXXX"));
		conexionD13 = false;
		ValorD13 = 0.40;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenD13 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionD13 == false)
	{
		GetDlgItem(IDC_D13)->SetWindowText(_TEXT("0.40"));
		conexionD13 = true;
		ValorD13 = 0.00;
		PorcenD13 = 0.00;
	}
}


void TablaV1::OnBnClickedD14()
{
	// TODO: Agregue aquí su código de controlador de notificación de control

	if (conexionD14 == true)
	{
		GetDlgItem(IDC_D14)->SetWindowText(_TEXT("XXXX"));
		conexionD14 = false;
		ValorD14 = 0.02;
		Ocultacion DlgOcultacion(NULL); //Llama al constructor del dialogo
% de sombra

		if (DlgOcultacion.DoModal() == IDOK)
		{
			PorcenD14 = VSombraAux;
		}
		VSombraAux = 0;
	}

	else if (conexionD14 == false)
	{
		GetDlgItem(IDC_D14)->SetWindowText(_TEXT("0.02"));
		conexionD14 = true;
		ValorD14 = 0.00;
		PorcenD14 = 0.00;
	}
}
```

Una vez se hayan seleccionado todas las casillas de pérdidas de la tabla, al "aceptar" y cerrar el cuadro de dialogo, se invocará a la función *OnBnClickedOk*, en la pulsación del botón especifico, que operará, la suma de los productos del valor

seleccionado de cada casilla, por su porcentaje de representatividad u ocultación, devolviendo esto el calor de las perdidas por sombreo calculado.

```
void TablaV1::OnBnClickedOk()
{
	// TODO: Agregue aquí su código de controlador de notificación de control
	//calulo final perdidas sombreo
	m_PerdSombras = ValorA1*PorcenA1 + ValorA2*PorcenA2 + ValorA3*PorcenA3 +
	ValorA4*PorcenA4 + ValorA5*PorcenA5 + ValorA6*PorcenA6 + ValorA7*PorcenA7
	+ ValorA8*PorcenA8 + ValorA9*PorcenA9 + ValorA10*PorcenA10 +
	ValorA11*PorcenA11 + ValorA12*PorcenA12 + ValorA13*PorcenA13 +
	ValorA14*PorcenA14 + ValorB1*PorcenB1 + ValorB2*PorcenB2 +ValorB3*PorcenB3
	+ ValorB4*PorcenB4 + ValorB5*PorcenB5 + ValorB6*PorcenB6 +
	ValorB7*PorcenB7 + ValorB8*PorcenB8 + ValorB9*PorcenB9 +
	ValorB10*PorcenB10 + ValorB11*PorcenB11 + ValorB12*PorcenB12 +
	ValorB13*PorcenB13 + ValorB14*PorcenB14 + ValorC1*PorcenC1 +
	ValorC2*PorcenC2 + ValorC3*PorcenC3 + ValorC4*PorcenC4 + ValorC5*PorcenC5
	+ ValorC6*PorcenC6 + ValorC7*PorcenC7 + ValorC8*PorcenC8 +
	ValorC9*PorcenC9 + ValorC10*PorcenC10 + ValorC11*PorcenC11 +
	ValorC12*PorcenC12 + ValorC13*PorcenC13 + ValorC14*PorcenC14 +
	ValorD1*PorcenD1 + ValorD2*PorcenD2 + ValorD3*PorcenD3 + ValorD4*PorcenD4
	+ ValorD5*PorcenD5 + ValorD6*PorcenD6 + ValorD7*PorcenD7 +
	ValorD8*PorcenD8 + ValorD9*PorcenD9 + ValorD10*PorcenD10 +
	ValorD11*PorcenD11 + ValorD12*PorcenD12 + ValorD13*PorcenD13 +
	ValorD14*PorcenD14;

	//variable almacenaje perdidas sombras
	PerdSombrasAux = m_PerdSombras;

	CDialogEx::OnOK();
}
```