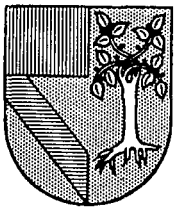


UNIVERSIDAD PANAMERICANA

Escuela de Ingeniería

Con estudios incorporados a la
Universidad Nacional Autónoma de México

Interfaz Gráfica para Simulación Computarizada de Procesos de Manufactura

T E S I S

Que para obtener el título de:

INGENIERO MECANICO ELECTRICISTA

Area:

INGENIERIA MECANICA

P r e s e n t a n :

Omar Aguirre Suárez
Gerardo Bárcena Ruiz

DIRECTOR:
Dr. Stanislaw Raczynski Gawin

México, D. F. 1993

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

ÍNDICE

INTRODUCCIÓN

Debido al gran desarrollo que han tenido las computadoras en los últimos años, la simulación ha cobrado un especial auge en los procesos de manufactura. Nos permite, entre otras cosas: observar, analizar y cambiar las condiciones de un modelo; de tal forma que se ha convertido en una herramienta de incalculable valor en el diseño, operación y mantenimiento de distintos procesos en la industria.

En un entorno de cambios y feroz competencia económica para la industria mexicana, es necesario aplicar estas herramientas que ayudan a mejorar los procesos productivos, disminuyendo costos de fabricación y, así lograr mayor competitividad en el mercado. Por tal motivo, la simulación necesita unirse estrechamente a la industria proporcionando resultados tangibles. Actualmente, podemos observar una gran variedad de simuladores de procesos de manufactura, la gran mayoría son sistemas costosos y con aplicación limitada que restringen su uso a las grandes corporaciones.

De ahí que surja la necesidad de crear una aplicación económica que establezca una metodología sencilla y rápida para definir los procesos de manufactura. El ingeniero electromecánico, por sus conocimientos de los procesos productivos, se convierte en una parte fundamental para desarrollar dicha metodología. Con la ayuda de lenguajes de simulación ya establecidos y por medio de la programación en Windows, le brindará al usuario una herramienta sencilla, amigable y eficaz para la definición de modelos en los procesos industriales.

Reconocemos que existen problemas inherentes a la simulación, como: suposiciones erróneas al elaborar el modelo, planteamiento poco claro de un objetivo, el análisis incorrecto de los resultados, etcétera; sin embargo, si estos problemas se logran superar, la simulación se convierte en un aliado de invaluable importancia.

Cabe mencionar que la simulación es una herramienta que nos permite evaluar los procesos productivos o de servicios para mejorar las operaciones que se llevan a cabo, dándole la máxima rentabilidad económica sin menoscabo de la calidad del producto que se fabrique o del servicio que se preste.

Este trabajo, por tanto, presenta el diseño de una interfaz para la definición de procesos de manufactura. En el primer capítulo se da una visión general de los procesos de manufactura, para posteriormente, en el segundo capítulo hablar de los antecedentes históricos de la simulación; así como, de algunos lenguajes y programas de manufactura. En el tercer capítulo se establecen los aspectos generales de PSI. El cuarto capítulo cubre el análisis y diseño orientado a objetos del sistema propuesto PSI (*Process Simulation Interface*) y, finalmente, el quinto capítulo da los lineamientos de programación seguidos en el desarrollo del sistema.

En los apéndices de este trabajo se encuentran el manual del usuario y el manual técnico, que complementan la información del sistema realizado.

I. PROCESOS DE MANUFACTURA

La manufactura es la fabricación de artículos u objetos por medio de procesos industriales; como ejemplos mencionaremos a las industrias: farmacéutica, de alimentos, maquinaria, artículos eléctricos, industria peletera y de calzado, y muchas otras.

Es importante resaltar la gran variedad de procesos de manufactura o productivos que cada una de las áreas citadas posee, y los distintos métodos para realizar una misma tarea; pero, ¿cuál de ellos es el mejor?; o bien, si fuéramos a instalar una industria, ¿cómo determinar las políticas de operación, y los procesos claves?.

Desde luego, las preguntas anteriores no son sencillas, y la respuesta puede venir de muchas y diversas fuentes; nosotros aquí expondremos una respuesta a estas interrogantes desde nuestro particular punto de vista sobre la aplicación práctica de la herramienta que ofrecemos.

Todo proceso de manufactura surge de un largo trabajo de diseño, donde se estudian las características propias para la realización de esa tarea, la disposición de recursos para lograrlo, y la creatividad y constancia de los responsables de la planta productiva.

El plan a desarrollar que se necesita para realizar un proceso productivo lo podemos dividir en cinco fases, que se expondrán a continuación.

1. Fase Introductoria:

Se basa en un *análisis* del proceso productivo., y es aquí donde se investigan las características y necesidades para realizar el trabajo. Es la fase de recopilación de información de: materiales, costos, maquinaria, posibles métodos de producción, distintos métodos de manufactura, así como la distribución de la planta.

La mejor ayuda en esta fase es hacer diagramas, bosquejos y dibujos que recopilen toda la información que se ha recabado.

2. Fase Creativa:

Aquí se realiza el *diseño* y *síntesis*. El diseñador establece las reglas de asociación y políticas que regirán al proceso productivo. Se someten todas las opciones recopiladas en la fase anterior a un escrupuloso proceso de discriminación, hasta llegar a la mejor opción en: tecnología, métodos de trabajo, proveedores, materiales, equipo, recursos humanos, y todo lo necesario para desarrollar la tarea encomendada.

El aspecto de *síntesis* al que nos referimos tiene que ver con la comprensión de los requerimientos anteriores, por ejemplo: ocupar proveedores ya existentes, ocupar maquinaria que se posee con anterioridad o, tal vez, ocupar una misma máquina con horarios distintos que realice diversas tareas del mismo proceso.

La *síntesis* también se refiere a una fase discriminatoria o de veto, pues el proceso diseñado cuenta con todas las posibilidades en potencia; pero aquí es donde los directivos aprueban o no la inversión, en función de la rentabilidad económica que les preporcione.

3. Fase Especulativa:

En el diseño se establecieron las reglas de funcionamiento de la planta, topes, límites, cuotas, producción y tiempos. Pero ésto según especificación de velocidad de las máquinas y de los métodos de producción. Con ello se pueden hacer proyecciones o *pronósticos* del comportamiento de todo el proceso productivo para periodos razonables como pueden ser a unas cuantas semanas o meses de haberse iniciado la manufactura.

Dependiendo de qué tan bueno haya sido la fase de diseño, las proyecciones se ajustarán más a la realidad.

4. Fase Operativa:

Una vez aprobado el proyecto se procede a su *realización*. Éste es un período difícil pero muy importante, pues se deben aplicar perfectamente las políticas para no viciar desde el nacimiento al proceso productivo. Desde luego, es evidente que cualquier aspecto que sea perfectible se asimila durante la ejecución misma de la instalación pues, como ya sabemos todo proceso productivo es flexible y siempre cambia para mejorar.

Aquí también se realiza una recopilación de la información que tiene que ver con los pronósticos para observar si se cumplen, o la desviación es muy alta.

5. Fase de Crítica:

Una vez que se han analizado los datos recopilados en la fase anterior, se debe investigar las causas fundamentales que provocaron las diferencias con nuestras proyecciones.

Entonces se procede a una *retroalimentación* que lleve a la fase de diseño, para el establecimiento de reglas y políticas, a fin de modificarlas o enriquecerlas con los cambios del análisis actual. Posteriormente, se debe notificar al área de producción el cambio de políticas para asimilarlas al proceso.

Si las políticas se definieron adecuadamente, la falla se encuentra en su aplicación, y se deben poner todos los medios para destruir las situaciones que vician nuestro proceso productivo.

En la medida en que menos se recurra a la retroalimentación, el responsable del proceso productivo mejorará en el diseño de manufactura: visto de otra manera, más se conoce el proceso y por tanto, se contempla desde el inicio todos sus casos y vertientes.

6. Relación con una interfaz gráfica

Ahora bien, con lo anteriormente expuesto, *¿cómo se relaciona con una interfaz gráfica de simulación de procesos de manufactura por computadora?*. Contestar con un contundente *¡En Todo!*, es exagerar; pero sí en un alto porcentaje que se explicará en seguida:

- En la fase de análisis no proporciona ningún beneficio, porque esta tarea tiene que ver con el factor humano y las negociaciones, pero es imprescindible que se realice con el mayor de los cuidados, pues es la alimentación del proceso siguiente.

✺ Para la fase creativa, el apoyo que presta una interfaz gráfica es muy importante, ya que se pueden realizar modelos de plantas y visualizar su funcionamiento y rendimiento. La información que se investigó en la fase anterior sobre maquinaria, por ejemplo (retrasos, número de piezas, distribución y otros aspectos), se ingresa en el editor para la simulación, y se pueden conocer gran cantidad de variantes en un tiempo muy corto, y la selección de un método, o maquinaria se facilita y se reduce en tiempo, con el consecuente ahorro económico.

La rapidez hace la diferencia, ya que se pueden investigar más configuraciones de planta dando mayor certidumbre a que los resultados sean los óptimos.

Una vez que el diseño llega a su fin, se cuenta con toda la descripción lógica y visual del proceso productivo, posibilitando su distribución entre el equipo de trabajo, con ello se logra extender el concepto a un mayor número de personas, que pueden contribuir a su mejoramiento. Es habitual que exista un experto en el diseño, pero no tiene la facilidad de compartir con los demás sus conocimientos. Con un simulador y su interfaz, todos hablan el mismo lenguaje.

En la etapa de síntesis, la interfaz gráfica permite *visualizar* el proceso productivo y tomar las decisiones que corresponden a este propósito.

✺ La fase especulativa se enriquece con los resultados del simulador -que es alimentado por medio de la interfaz gráfica-, pues al cambiar las condiciones de operación, se puede conseguir la respuesta del sistema de forma casi inmediata, se interpretan estos resultados, y nos ayuda a establecer los pronósticos, asegurando llegar a los mejores resultados en calidad y economía.

✺ Para la fase operativa y de recolección de datos no es de mucha ayuda, pues nuevamente este es un trabajo más dirigido hacia el factor humano.

✺ En la fase de retroalimentación es de gran utilidad, pues al ingresar los nuevos datos que caracterizan a nuestro proceso productivo, es muy sencillo por medio de la interfaz gráfica.

Se debe tener muy en cuenta que la interfaz gráfica y el simulador son herramientas muy útiles, pero no realizan nada por sí mismas, sino en colaboración con el usuario; ya que si se ingresa basura, entregará basura, por lo que el triunfo o fracaso del uso de una herramienta como ésta es responsabilidad del usuario y, claro, del programador que promete que funciona como afirma.

La utilización de una interfaz gráfica tiene tres aspectos que van en relación directa con el poseedor de la planta productiva:

a. El que va a iniciar un nuevo proceso productivo

Este caso es el descrito anteriormente.

b. El que ya cuenta con uno, pero desea mejorarlo

En este caso, el responsable del proceso productivo debe crear la planta lógica para simularla y verificar que los datos de la simulación tengan la debida correlación con los obtenidos del proceso real. Si no es así, investigar las causas y mejorar el proceso, o la definición lógica de la planta.

Éste es básicamente un proceso de fases operativa y crítica, que sirven para depurar el proceso, desde luego una vez que la descripción lógica de éste se tiene en la computadora.

c. El que desea conocer mejor cómo funciona el proceso.

En el tercer caso, el responsable de planta que se enfrenta a la tarea de crear la definición lógica del proceso productivo, debe recopilar información indispensable para el simulador, lo que le obliga a investigar y aprender del proceso productivo que tiene en sus manos; de ahí que, la investigación le ayuda a conceptualizar de mejor manera dicho proceso.

A continuación se presenta una tabla con las fases, las acciones generales, y la ayuda que nos brinda la interfaz gráfica:

FASE	ACCIÓN BÁSICA	LO QUE OFRECE LA INTERFAZ GRÁFICA
Introductoria	Análisis	
Creativa	Diseño Síntesis	Probar gran número de opciones Depurar mejor el proceso Extender el concepto al equipo de trabajo No crear personas indispensables
Especulativa	Pronóstico	Creación del pronóstico con ayuda de los resultados del simulador
Operativa	Realización Comprobación del pronóstico	
Crítica	Retroalimentación	Nueva depuración del modelo.

Recordemos una vez más que los ambientes gráficos, simuladores y computadoras son sólo herramientas, y que los responsables de los logros o fracasos somos nosotros, con nuestro trabajo y empeño.

Toda la explicación anterior está orientada hacia el área productiva, pero la interfaz puede soportar otro tipo de procesos como: hospitales, estaciones de servicio, supermercados, aeropuertos, escuelas, salas de espera de cualquier tipo, y un sin número de lugares donde se necesite simular y mejorar las condiciones para elevar la productividad y, por ende, la

economía del proceso en cuestión; pues la simulación es económica, se hace en espacio reducido, y los resultados son de rápida aplicación.

II. ANTECEDENTES

La simulación se puede entender como la utilización de un modelo que representa el comportamiento de un sistema, y nos da una visión del desarrollo de ciertas variables críticas bajo diversas circunstancias. Las simulaciones son a menudo utilizadas para determinar la productividad y el comportamiento de ciertos elementos del sistema.

Por ejemplo, podríamos desear observar cómo afecta la distribución de recursos en un hospital al tiempo que los pacientes esperan para recibir una serie de servicios. En este caso, nosotros diseñaríamos un modelo que represente este fenómeno, en donde tendríamos procesos como la llegada de pacientes dada una cierta distribución, la formación de colas de espera, el tipo de servicio que el paciente requiere, los médicos disponibles, etcétera. Gracias al modelo planteado, seríamos capaces de determinar y analizar cambios en el sistema original a fin de elevar la productividad, observando las consecuencias producidas. Es decir, nos permite preguntarnos: ¿Qué sucede si...?

En todos los ejemplos que se nos pudieran ocurrir, la computadora forma parte fundamental, pues gracias a su velocidad de procesamiento y disponibilidad, nos permite formular modelos complejos y de esta forma analizar los resultados.

1. Simulación de eventos discretos

Frecuentemente, escuchamos que la simulación es utilizada para distintas actividades tales como: el entrenamiento de pilotos, lanzamiento de cohetes, cuestiones meteorológicas. Estas simulaciones describen cómo un sistema cambia continuamente con respecto al tiempo, en respuesta a controles continuos (tales como el cambio de posición del volante en el coche), que pueden variar suavemente a través del tiempo.

En contraste, una simulación discreta describe sistemas que cambian instantáneamente en respuesta a ciertos eventos u ocurrencias. Por ejemplo, si estamos realizando un estudio de planeación de capacidades sobre un elevador de granos, podríamos simular cómo cambia la cantidad de grano almacenada en el curso de un año. Para este propósito, probablemente modelaríamos la llegada de un camión de carga de grano como un evento discreto. En otras palabras, ignoraríamos el hecho de que la cantidad de granos almacenados cambia lentamente mientras el grano es depositado en un almacén y, asumimos en cambio, que el nivel de grano cambia a un nuevo valor en un momento específico. Esta afirmación sería apropiada al estar modelando el sistema en una base día-a-día, camión-por-camión; en lugar de una base segundo-por-segundo, grano-por-grano.

Cuando escogemos modelar al mundo real utilizando simulaciones de eventos discretos, abandonamos la posibilidad de capturar detalles que sólo podrían ser descritos por cambios muy pequeños, casi continuos. A cambio, obtenemos la simplicidad que nos permite tener elementos importantes de muchos sistemas que son demasiados complejos para capturar con simulaciones continuas.

Sin embargo, el análisis de la simulación no se encuentra fuera de problemas. Primero, la calidad del análisis depende de la calidad del modelo. Segundo, es difícil a menudo determinar la razón del resultado de una observación hecha durante la ejecución de la simulación; no se sabe si es debida a una relación importante en el modelo, o es debido a la construcción aleatoria de la ejecución; de esta forma los resultados son difíciles de interpretar. Finalmente, la simulación normalmente es desgastante en tiempo y dinero; pudiendo llegar a un modelo que no represente fielmente la realidad. Los métodos analíticos para simulaciones parecen ser mejores para estimaciones rápidas.

Normalmente los problemas encontrados son:

1. Fallas en determinar un objetivo claro y conciso.
2. Fallas para determinar, en base al objetivo, una pregunta que responda el modelo.
3. Utilizar una simulación cuando existe una respuesta más sencilla.
4. Nivel de complejidad no apropiado.
5. Suposiciones erróneas.
6. Mala interpretación de los resultados.

Para que una simulación sea efectiva, se debe concentrar en la buena definición del problema (de otra forma no sabríamos qué elementos debemos incluir en el sistema y qué información debemos guardar, para analizar posteriormente). Hacer uso de la simulación sin antes haber definido bien el problema, puede llevarnos a diseños no apropiados de la realidad y, puede producir poca información de valor.
Es también importante utilizar un modelo con un nivel apropiado de detalle, pues a veces podemos separarnos del objetivo inicial, resultando un gasto de dinero y tiempo.

Actualmente, existe una infinidad de herramientas que nos ayudan a definir simulaciones discretas. Nosotros daremos un breve resumen de las características de algunas de ellas.

2. GPSS

El GPSS (*General Purpose Simulation System*) es un lenguaje de simulación utilizado para construir modelos de eventos discretos por computadora. Es especialmente usado para modelar sistemas en los cuales, las unidades discretas de tráfico compiten con otras unidades para utilizar ciertos recursos. Es de gran ayuda para determinar cómo responderá el sistema ante nuevas demandas, puede ser utilizado para modelar sistemas de manufactura, comunicación, cómputo, transporte y de inventarios, entre otros.
El GPSS ofrece una riqueza semántica con sintaxis clara. Por ejemplo, sólo son necesarios siete comandos (mas controles de ejecución), para modelar una línea con un servidor y una cola. Los comandos son realmente simples, por ejemplo: GENERATE 18,6 y QUEUE LINE. Automáticamente GPSS genera estadísticas de ocupación para servidores y colas.
Además de ser un lenguaje orientado hacia bloques, es bastante sencillo de aprender para los simuladores principiantes (sin embargo, requerirá de bastante tiempo para alcanzar un nivel de experto).

En GPSS se proponen los siguientes pasos:

a. Definición del problema.
b. Flujo del proceso.
c. Creación de un archivo del modelo.
d. Ejecución de la simulación.
e. Análisis de los resultados.

Para observar las características de este lenguaje veamos un pequeño ejemplo:

a. Definición del problema

En un sistema de manufactura, elementos provenientes de una fundición son pasados a una taladradora, donde a cada pieza se le hará un agujero. El tiempo de llegada de las piezas de fundición es una distribución uniforme en un intervalo de 15.0±4.5 minutos. El tiempo requerido por el taladro es de 13.5±3.0 minutos, uniformemente distribuido. Las piezas de fundición son pasadas al taladro como: primera que llega, primera que se taladra. Es necesario modelar este sistema en GPSS, obteniendo las estadísticas de tiempo de espera, así como el tiempo por unidad. Simular para 100 elementos.

b. Definición del flujo del proceso

Consideremos la serie de procesos por los que pasa una pieza fundida (llamada en GPSS transacción):

1. La pieza fundida llega al sistema.
2. La pieza fundida pide ser atendida.
3. La pieza fundida espera, si es necesario, a ser capturada por el taladro.
4. La pieza es capturada por el taladro.
5. A la pieza se le hace el agujero correspondiente.
6. La pieza sale del taladro.
7. La pieza deja de pertenecer al sistema.

c. Archivo del modelo

El archivo (con formato ASCII) tendría un aspecto semejante a:

```
SIMULATE
GENERATE        15.0,4.5    Llegada de la pieza fundida
QUEUE           DRILLQUE    Registrar la pieza en la cola
SEIZE           DRILL       Pide ser atendida
DEPART          DRILLQUE    Quita del registro la pieza en la cola para
estadísticas
ADVANCE         13.5,3.0    El tiempo que tarda el taladro en atender a la
pieza
RELEASE         DRILL       Libera al taladro para poder atender otra pieza
TERMINATE       1           La pieza deja el sistema

START           100         Empieza la simulación
END                         Fin del archivo de definición
```

d. Ejecución del modelo

Al correr la simulación se obtendrán una serie de valores estadísticos que permiten establecer: el promedio de espera de una pieza en la cola, número de piezas en la cola al finalizar la simulación, porcentaje de ocupación del taladro, tiempo total necesario para procesar 100 piezas, etcétera.

GPSS actualmente puede ejecutarse en equipos tales como: VAX, Sun 3, HP Apollo, NCR basadas en Unix; todas las máquinas basadas en la familia Motorola 68000, IBM PC y compatibles.

3. SIMFACTORY II.5

SIMFACTORY II.5 es un simulador de fábricas desarrollado en SIMSCRIPT II.5 que provee al usuario la habilidad de modelar rápidamente fábricas sin ser necesaria la programación. Esta capacidad es posible gracias a la interfaz gráfica, ayudada por el ratón, que permite al usuario construir representaciones gráficas de su fábrica.
SIMFACTORY II.5 ha sido escrito para ingenieros que, debido a sus múltiples actividades, no pueden dedicarse de tiempo completo a la simulación. Normalmente, esto es porque tienen otras responsabilidades que no están relacionadas con la simulación; y es por esto, que harán cambios en las líneas de producción sin previamente contemplar una simulación.
Modelar con SIMFACTORY es mucho más exitoso cuando es realizado de una manera interactiva, empezando con un modelo sencillo de la fábrica. Después de que el modelo inicial es desarrollado, se empieza a mejorar poco a poco, hasta llegar a un modelo que se ajuste correctamente a la realidad.
El primer modelo simplificado se le llama modelo base. Un modelo base de SIMFACTORY representa sólo las estaciones, colas y trayectorias de transporte que existen en el piso de la fábrica. Transportadores y bandas son ignorados. Aunque muchos productos pueden estar en la fábrica, sólo dos o tres serán incluidos en este primer modelo.

a. Definición de la representación gráfica

La representación consiste en las estaciones de procesado, colas, áreas de recepción y caminos de transportación. Es creada mediante la selección y el posicionamiento de imágenes que representan estos componentes. Tan pronto como una imagen es puesta en la pantalla, se ingresan los datos correspondientes a dicho elemento. Existen capacidades de edición, tales como: copiar, mover, borrar, etcétera. Después de que las imágenes son posicionadas y descritas, las rutas de transporte que conectan un elemento con otro son dibujadas.

b. Definición de los productos

En SIMFACTORY los pasos necesarios para realizar un producto son definidos en el plan de procesamiento, que consiste en la serie de operaciones necesarias para producir un producto. Los planes de procesamiento determinan qué operaciones son realizadas en la parte, la duración de cada operación, y el orden en el que serán realizadas. Ensamblados, desensamblados, divisores, etcétera, son parte del plan de procesamiento. Con esta forma es posible tener múltiples productos en la misma línea de producción. De hecho, cada producto puede tener su propio conjunto de tiempos de procesado. Retrabajos son fácilmente definidos.
Un plan de procesamiento consiste en tres listas: una lista de materia prima para el plan, una lista de operaciones realizadas en las partes, y una lista de las salidas de las partes generadas.

c. Definición de recursos

En SIMFACTORY el término recurso se refiere a cualquier elemento necesario para llevar a cabo alguna operación, en la estación de procesamiento. Por ejemplo, un recurso podría ser un operario que utiliza tres máquinas.
Los recursos son ingresados al modelo básico en dos pasos. En el primer paso, el recurso es definido y se establece la cantidad disponible en un principio. En el segundo paso, se definen las estaciones que requieren de dichos recursos (esto se realiza indicando qué estación requiere del recurso, y cuánto requiere de él).

d. Definición de los transportadores

Los transportadores en SIMFACTORY pueden ser transportadores por lotes o bandas. Para definir un transportador, primero es necesario establecer la posición inicial en la representación gráfica, y luego definir las características del transportador. Las características importantes de un transportador se refieren a su velocidad de avance, tiempo para carga, tiempo para descarga, capacidad. Posteriormente se definen los caminos de transporte para dichos elementos.

e. Definición de interrupciones

Las interrupciones son cualquier actividad que interfiere con la operación de una estación o transportador. Los dos ejemplos más comunes son las fallas del equipo y el mantenimiento preventivo. En SIMFACTORY podríamos decir que la falla es una interrupción de prioridad porque toma prioridad sobre cualquier otra cosa que la estación pudiera estar haciendo. El mantenimiento preventivo sólo ocurrirá cuando la estación está entre operaciones.
Otras características de las interrupciones son que pueden especificarse el tiempo promedio entre interrupciones, el tipo de interrupción de reloj y el promedio de tiempo para reactivar.

f. Reportes disponibles

Durante la ejecución de la simulación se despliegan diversos datos para dar un importante seguimiento a la misma. Avisa de la llegada de materia prima, el terminado de una parte, el comienzo de una interrupción, etcétera.
Después de la simulación, los reportes disponibles tienen un resumen de cómo se comportó el sistema. Dentro de esa información tenemos elementos en la utilización del equipo, estadísticas en las colas, tiempo de ocupación en las estaciones, etcétera.
La simulación también es posible en este paquete.

Actualmente, SIMFACTORY funciona en IBM PC AT's y compatibles (bajo DOS, Windows y OS/2), PS/2's, HP 9000 Series 300, Sun 3 y Sun 4.

4. PROMODEL

ProModel combina la flexibilidad de los lenguajes para propósitos en general y la facilidad de simuladores de procesos de manufactura. ProModel está diseñado para ser utilizado tanto por principiantes, como por expertos dentro del campo de la simulación. Por su facilidad de uso, es también atractivo para los profesores de ingeniería o negocios que están interesados en enseñar a modelar y analizar diversos conceptos en la simulación.
Cuando se simulan sistemas complejos que requieren un análisis extenso, normalmente está un experto en simulación envuelto en la actividad del modelado. En tales situaciones, la completa flexibilidad del modelo sólo puede ser lograda por programación tradicional. Para satisfacer esta necesidad, ProModel ofrece posibilidad de programar en un lenguaje tipo PASCAL-C, mediante el cual se puede accesar a la simulación, sin salir del programa.
En ProModel todo el desarrollo se intentó realizar gráficamente y orientado a objetos.
Se pueden construir complejos sistemas de manufactura involucrando transportadores, manufactura flexible, recursos inteligentes, etcétera.
Este programa fue realizado utilizando C++, lo que le da una robustez y una portabilidad importante.
Utiliza el intercambio de información mediante WINDOWS (*DDE Dynamic Data Exchange*) y OS/2.
El tamaño del modelo tan sólo es limitado por la cantidad de memoria disponible.
ProModel ofrece la posibilidad de editar imágenes en 2-D en un editor gráfico, así como permite comunicación con archivos DXF.
ProModel es capaz de ofrecer animación, y la información necesaria se captura en el momento de la definición.

a. Elementos para el modelado

En ProModel, un modelo define un sistema de producción o servicio, el cual consiste en una serie de elementos que se procesan, uno o más lugares de procesamiento, cualquier número de recursos auxiliares, logística y operaciones lógicas, además de un plan de producción. Los elementos para el modelado en ProModel son los bloques utilizados para representar los componentes físicos y lógicos de un sistema actualmente desarrollado.
Los elementos físicos de un sistema, tales como partes y recursos pueden ser llamados por nombre o gráficamente. Los nombres pueden ser cualquier palabra o combinación de ellas hasta 32000 caracteres de largo. Ahora veamos los principales elementos integrantes de ProModel.

b. Partes o entidades

Las partes o entidades se refieren a los elementos que están siendo procesados en el sistema. Éstos incluyen materia prima, piezas, productos terminados. Entidades de distinto

tipo pueden ser consolidadas en una sola entidad, separada en dos o más entidades adicionales o convertidas en una o más entidades nuevas.
A las entidades se les puede asignar atributos que pueden ser tomados en cuenta para hacer decisiones o para formar estadísticas especiales.

c. Localización de rutas

La localización de las rutas se refiere a los lugares fijos en el sistema (tales como máquinas, colas, almacenes, áreas de trabajo, etcétera), en donde las partes o entidades son transportadas para ser procesadas, almacenadas o simplemente para hacer una decisión de flujo del proceso. La localización de rutas pueden formarse con una o varias máquinas.
La localización de rutas puede tener una capacidad más grande que una parte y puede tener tiempos de apagado, mediante funciones de tiempo, tiempo de utilización, frecuencia de utilización, cambio de material.

d. Recursos

Un recurso puede ser una persona, herramienta, vehículo u otro dispositivo que se utiliza para transportar material entre distintas áreas de trabajo, para desarrollar algún tipo de operación, o para realizar mantenimiento. Dentro de estos recursos encontramos: recursos en general, móviles, robots, grúas.

e. Bandas

Una banda es un elemento de movimiento continuo a lo largo del cual se depositan entidades que son transportadas a otras áreas. Las bandas tienen ciertas características tales como: velocidad, espaciado del material, tiempo de descarga. Las bandas pueden ser configuradas con transferencias, ciclos de recirculación, acomodo en orden, acumulación y distribuciones. Bandas bidireccionales y complejos sistemas de bandas pueden ser modelados.

f. Variables, funciones y distribuciones

ProModel ofrece una gran cantidad de variables para la realización de decisiones y para reportes estadísticos. Las variables son de dos tipos principalmente: (1) de sistema o de estado, tales como el tiempo de simulación, o la capacidad de un recurso, y (2) definidas por el usuario que incluyen vectores.
Además de las variables, ProModel tiene muchas funciones definidas internamente y distribuciones incluyendo distribuciones discretas y continuas estándares. Dentro de las distribuciones típicas tenemos: exponencial, normal, uniforme, triangular, beta, gamma, erlang, weibull, lognormal.

ProModel actualmente funciona en equipo IBM -VGA, y existen versiones para red.

5. SIMAN IV

Siman IV (*SIMulation ANalysis*) es un simulador de propósitos múltiples para modelar cualquiera de las siguientes tres orientaciones: para sistemas discretos, ya sean orientados por procesos o por eventos; para sistemas continuos modelados algebraicamente, con ecuaciones diferenciales o en diferencias; y finalmente, para una combinación de ambos.

Siman IV cuenta con importantes características tales como:

1. Construcciones de propósito-específico para simplificar y mejorar el modelado de sistemas de manufactura.
2. Construcciones que hacen más fácil modelar el manejo de material (por ejemplo, AGV's y bandas).
3. Bloques que permiten la entrada-salida de información en el modelo sin necesidad de código escrito por el usuario. Soporta archivos con formato, sin formato, secuenciales, de acceso directo, de hojas de cálculo, dando al usuario una interfase hacia muchos productos o bases de datos.
4. Un asistente interactivo para encontrar errores (*debugger*) que permite monitorear y controlar la ejecución de una simulación.
5. Su ambiente es por medio de menús, que integra las funciones de definición, ejecución y simulación del modelo.
6. Uso transparente de CINEMA IV para las simulaciones.
7. Completa compatibilidad a través de estaciones de trabajo, microcomputadoras, etcétera. Los modelos pueden ser movidos de una plataforma a otra sin modificación.

SIMAN IV cuenta con un manejo de bloques similar al de GPSS. De esta forma contamos con elementos como: colas, servidores, divisores, etcétera. Su código podría verse de la siguiente forma:

```
Begin
Project, Simple System, Analyst;
Queues: Buffer;
Resources: Machine,1;
Replicate, 1, 0, 480;
End;
```

SIMAN IV maneja macros para un conjunto de elementos, plan de proceso, plan de recursos, transportadores y bandas.

6. SLAM II

SLAM II (*Simulation Language for Alternative Modeling*) fue el primer lenguaje de simulación que permitió al diseñador formular una descripción de un sistema utilizando procesos, eventos, vistas de un mundo continuo, o cualquier combinación entre ellos.
Junto con SLAM II se han desarrollado diversos paquetes para completar su función: TESS (*The Extended Simulation System*) provee un manejador de archivos para los resultados de la simulación y proporciona una interfaz gráfica para la descripción de modelos; MHEX (*Material Handling Extension*); IEE (*Interactive Execution Environment*), y SLAMSYSTEM que es un sistema integrado de simulación para computadoras personales bajo el ambiente OS/2 o *Windows*.

SLAM II proporciona un código de bloques semejante a GPSS. Podemos ver su semejanza en el siguiente ejemplo:

```
CREATE,EXPON(1.),0.,1;          Generar arrivos
QUEUE(1);                       Cola de espera para servicio
ACTIVITY(1),RNORM(1.,.2);       Procesado
COLCT,INT(1),TIME IN SYSTEM;    Coleccionar datos para estadísticas
ENDNETWORK
```

De esta forma SLAM se ha convertido en uno de los lenguajes más importantes dentro del mundo de la simulación.

7. ISI

ISI (*Intelligent Simulation Interface*) es una interfaz gráfica para lenguajes de simulación de propósitos múltiples. Actualmente provee una interfaz para el lenguaje de simulación SIMAN, aunque también podrían ser utilizados otros lenguajes tales como SLAM y GPSS. ISI proporciona un modelado hereditario, que satisface los tan frecuentes conflictos entre los requerimientos de facilidad y funcionalidad.

Algunos de los elementos más importantes en ISI son:

1. Integración. Todos los pasos de un proyecto de simulación pueden ser llevados a cabo sin salir de ISI. Facilidad para construir modelos, edición de datos, definición del experimento, generación de código, ejecución animada, procesado de resultados y preparación del reporte, también han sido integrados con ISI.
2. Facilidad de uso. ISI ha sido diseñado tanto para usuarios novatos, como para expertos. Tiene un menú tipo CAD. Ayuda con hipertexto.
3. Flexibilidad. ISI es totalmente compatible con SIMAN, por lo que es aplicable a una gran cantidad de aplicaciones.

4. Modelado hereditario. Librerías para la construcción de bloques con algún fin en específico pueden ser sintetizadas hereditariamente. De esta forma se pueden crear fácilmente modelos de simulación. ISI puede ser personalizado a distintas áreas de aplicación. El modelado con herencias permite el desarrollo de nuevos elementos funcionales a partir de los ya existentes.
5. Modelado gráfico.
6. Animación. ISI utiliza el mismo modelo gráfico desarrollado en la construcción del modelo para proveer animación inmediata. El avance de las entidades se logra moviendo imágenes en la pantalla, así como se despliegan barras que permiten observar la acumulación de entidades.
7. Procesamiento de resultados. ISI tiene múltiples formatos de salida que permiten al usuario tener la posibilidad de utilizar otros paquetes para su manejo.

8. PASION

PASION (*PAScal simulatION*) es un lenguaje de simulación orientado hacia eventos y procesos, diseñado para usuarios de Pascal. El lenguaje tiene una estructura con dos niveles (procesos/eventos) y permite todas las estructuras de Pascal. También ofrece las principales ventajas de la programación orientada a objetos. PASION provee los elementos necesarios para manejar secuencias de eventos aleatorios, colas y procesos cuasi-paralelos, tanto para modelos discretos, como para continuos.

Para describir una secuencia de eventos, debemos especificar las operaciones que se realizan en dicho evento, así como su relación con otros. Un lenguaje orientado a los procesos ofrece algo más. Por decirlo, define una estructura dentro de la cual un conjunto de eventos se dan lugar. Por proceso entendemos un segmento de código que contempla a un objeto. Este objeto puede tener ciertos atributos, que son utilizados por los eventos descritos. Los eventos normalmente establecen los servicios de un objeto. PASION contempla la posibilidad de herencia, además de la creación dinámica de objetos.

Un programa en PASION tendría el siguiente aspecto:

```
PROGRAM Trigger;
REF A,B: Class;
{A y B son objetos que pertenecen a la clase Class}

PROCESS Class,2; {Proceso o clase tipo Class, con 2 instancias}
ATR N:String[7];{Atributo de la clase Class}

    EVENT One; {One es un evento del proceso X}
        Writeln('Active object:',N);
        IF This=A THEN B.One:=Time + 1.0
        ELSE A.One:= Time + 1.0
```

```
ENDEV;

START {Programa principal}
NEWPR A; {Creación del objeto A}
A.N:='ObjectA';
NEWPR B; {Creación del objeto B}
B.N:='ObjectB';
A.One:= Time+1.0;
{El objeto A comienza dentro de una unidad de tiempo, B espera}
$ {Termina programa}
```

PASION se encarga, mediante un pre-compilador, de generar código en Pascal que posteriormente es compilado para generar un programa ejecutable.
PASION cuenta con dos programas que nos ayudan a generar simulaciones fácilmente. Uno de ellos para modelos discretos (QMG), y otro para modelos continuos (CMG).

a. QMG (Queuing Model Generator)

PASION tiene un generador de modelos de colas (*Queuing Model Generator*) donde se tiene acceso, de forma gráfica y con la ayuda del ratón, a una serie de elementos que nos permiten definir fácilmente modelos discretos. En este editor contamos con distintos tipos de elementos como son: generadores, terminadores, servidores, ensambladores, colas, divisores, etiquetas. Su facilidad permite que rápidamente se generen simulaciones discretas y podamos analizar los resultados.

QMG genera un archivo de comunicación con extensión .QMG y un archivo de atributos (.ATR) que convierte a lenguaje de PASION y posteriormente, mediante el pre-compilador, se genera código en Pascal.

Gracias a que PASION soporta toda la sintaxis de Pascal, es posible para un usuario el poder ingresar líneas de código que establezcan comportamientos específicos (esto da una gran flexibilidad a PASION).

b. CMG (Continuous Model Generator)

Este programa fue diseñado para facilitar la simulación de sistemas dinámicos continuos. El CMG es un programa que genera código en PASION y/o Pascal, de acuerdo a las especificaciones del modelo dadas por el usuario, principalmente en forma gráfica. El código de salida de CMG es creado en formato de procesos-eventos de PASION, que puede ser insertada en otros modelos de PASION (ya sea discretos o continuos). Los datos ingresados son formulados en términos de un diagrama gráfico que describe la dinámica del proceso simulado. Por diagrama gráfico entendemos un conjunto de nodos y enlaces con una función relacionada. CMG permite los siguientes tipos de enlace: estáticos, lineales, estáticos no

lineales, dinámicos lineales (dada la función de transferencia), tiempo de demora, *sample-hold*, y un superenlace (un sistema dinámico complejo).

PASION actualmente se ejecuta en IBM-PC, y existe una versión para RISC-6000.

III. ASPECTOS GENERALES

1. Introducción

El gran auge que han experimentado los programas amigables, nos brindó la oportunidad de realizar un trabajo que lograra una buena comunicación con el usuario.

La necesidad de crear programas que presentaran a las personas, objetos, conceptos, y tantos otros entes de nuestra vida, tal como son, presentaba un gran reto.

Pero el incluir gráficas en la computadora se debe a un mundo donde lo gráfico es indispensable.

Por poner un ejemplo: las señales de tránsito o de carreteras, que nos muestran el concepto, no nos lo dicen; pues este lenguaje es universal, y no tenemos que recurrir al lenguaje materno de la persona que lo ve.

Un concepto que se presenta visualmente se puede entender mejor, a uno que se explica oralmente o por escrito.

En muchas de las ocasiones las personas no cuentan con el lenguaje apropiado, fluidez o conceptos propios para darse a entender, con lo que caemos en una falta de comunicación.

Con un dibujo, bosquejo o símbolo gráfico, se puede transmitir mejor el mensaje que se desea compartir.

2. La filosofía

La filosofía fundamental de cualquier programa es la de servir, ya sea como una hoja de cálculo, un procesador de textos, una aplicación de fines lúdicos, o cualquier otra dentro de las existentes.

Pues bien, la interfaz que nosotros proponemos, también tiene como filosofía fundamental la de servicio, pero ese servicio ¿de cuántas maneras se puede presentar?. Trataremos de explicarlo de la mejor manera.

3. ¿Por qué Borland PASCAL?

PSI al haber sido desarrollado en Borland Pascal, presenta la primera ventaja; pues como es sabido, es uno de los lenguajes de programación de mayor difusión y de más rápido aprendizaje. Por lo que cualquier persona que conozca los conceptos fundamentales en la programación de este lenguaje, podrá continuar, modificar, o realizar sus propias ideas basadas en esta aplicación, sin demasiada dificultad.

Cabe señalar que la programación en Pascal no es fácil, requiere trabajo cuidadoso e investigación; pero en lo fundamental, Borland Pascal soporta los estándares internacionales.

4. Windows

La segunda ventaja es que fue desarrollado para ambiente *Windows*, que tiene todas las características de un programa amigable; no es necesario memorizar interminables listas de comandos para enfrentarse contra la computadora y no hacer un uso eficiente de ella.

La primera versión de *Windows* fue anunciada en noviembre de 1983, y salió a la luz pública dos años después. Lo anterior, nos ubica aproximadamente a ocho años después de su salida comercial, tiempo suficiente para que el producto haya madurado lo suficiente y poderlo explotar al máximo.

Los mismos creadores de *Windows* lo denominaron como una "interfaz visual", que era una verdadera revolución para esos años, cuando nació.

Algo más que hace a *Windows* interesante, es que nos permite el uso de varios programas a la vez, que conocemos como multi tareas, que describiremos su aprovechamiento posteriormente.

Todo debe de llevar un orden, y Microsoft lo logra con una línea de estilo muy definida, que todos los desarrolladores deben respetar al hacer sus paquetes comerciales; ya que los menús, mensajes, movimientos y aceleradores, cumplen con la misma forma, lo que hace que el usuario al aprender una aplicación, pueda emplear las demás con gran facilidad.

Otro de los aspectos que hace atractivo a *Windows*, es que los desarrolladores y constructores de equipo de cómputo, vieron en él la oportunidad de llegar con más fuerza al usuario; pues este ambiente hace uso de manejadores de dispositivos, lo que libera al programador de la necesidad de realizar sus propios manejadores.

Este ambiente gráfico se encarga de hacer uso de los dispositivos de video, impresores y otros, por medio de los manejadores, aprovechando todas las características aportadas (y sugeridas) por el fabricante. Lo anterior logra que el ambiente se presente siempre igual, no importando el equipo del que se trate.

Una vez que hemos explicado de manera general el ambiente de *Windows*, explicaremos como PSI hace uso de los recursos de éste.

5. Servicios de PSI

Nuestro programa presenta una estructura que soporta varios documentos al mismo tiempo, lo que permite modelar varias partes de una misma planta, o varios procesos sencillos al mismo tiempo.

El uso de los menús es igual al estándar, así como el de todas las demás funciones.

En la cuestión gráfica es donde se aprovechó más el uso de *Windows*. PSI cuenta con una extensa librería de imágenes propias de los sistemas de manufactura.

a. Librería gráfica

El modo de obtener estas imágenes, se realizó con la ayuda de un *scanner* adquirido ex profeso para esta ocasión.

Una vez que se tiene la imagen que se desea introducir a la computadora, se hace uso del *scanner*, por medio de programas como *Gray* o *Foto Touch* (los dos son programas de *Logitech*). Se especifican la cantidad de luz, contraste y resolución, se lee la imagen, y se escribe en la computadora.

Otra alternativa es por medio del uso de otro aparato llamado Foto Man (también de Logitech), que no es más que una cámara fotográfica, pero la fotografía se almacena en un EPROM. Tiene capacidad de 32 fotografías, y 24 horas de almacenamiento.

Ya que se tiene la imagen en la computadora, y se ha decidido que es la mejor (esto lo dice la experiencia y el estado en el que se encuentre), se reedita por medio de un buen programa de edición gráfica como Paint Brush 5 plus, sobre todo, por que debe soportar el formato TIFF de gráficos. Posteriormente, se debe realizar una traducción entre formatos, con traductores como Paint Shop Pro. El cambio de formato obedece a que el formato propio del scanner (o foto man) es el TIFF, y el de Windows es el BMP, con lo que debe llevarse acabo la traducción de un formato a otro para hacer uso de la imagen.

Como aclaración: el formato TIFF corresponde al estándar de Tagged Image File Format (formato de archivo de imagen etiquetada). Este formato es usado en computadoras tipo PC, tipo Macintosh, y sistemas Unix. Es un formato independiente, y no fue creado para un programa o equipo en específico.

El formato BMP es el estándar para Windows, Paint Brush para Windows, y el Portapapeles (Clipboard).

Cabe señalar que la traducción de un formato a otro (TIFF -> BMP), provoca una disminución de colores y con ello de la calidad de la imagen.

Lo anterior explica cómo logramos la librería de imágenes que incluimos con el programa, pero el usuario también puede seguir el procedimiento anterior para lograr sus propias imágenes, esto se explica con más detalle posteriormente, porque ahora debemos explicar las características del Portapapeles, para entender mejor cómo hacer la propia librería de imágenes.

b. El Portapapeles

La comunicación con el Portapapeles (Clipboard) tiene dos aspectos: la definición lógica de la planta se puede copiar allí, se copian tanto la información, como las imágenes, o sólo la imagen de pantalla (Ver Manual de Usuario). Esto nos permite la comunicación entre nuestros documentos por medio de la memoria, y no por el tardado y, en ocasiones, infructífero camino que puede ser el disco.

En cuanto a la copia de segmentos de procesos de manufactura al portapapeles, nos puede servir para hacer una librería de mini procesos bien definidos, y que pegamos cada vez que es necesario. Por ejemplo: se tiene una planta de producción de tornillos; donde se puede tener la definición lógica de una máquina, se copia cuantas veces sea necesario; además se puede guardar en disco y hacer uso de esa definición cada vez que así lo requiera.

En cuanto a la copia de imagen como tal del portapapeles, si el usuario de PSI cuenta con su propio scanner o Foto Man, puede copiar las fotografías de sus propias máquinas, procesos de producción, fotografías de los responsables, logotipos de cada parte del proceso, el logotipo propio de la empresa, o cualquier otra imagen que le sea de importancia. Desde luego, si la librería que aportamos no satisface todas sus necesidades, y si desea no hacer uso del formato de diagramas estándar con que cuenta PSI.

Aquí es importante señalar, que PSI propone un formato de imágenes para cada tipo de elemento para la simulación, pero como nuestra pretensión fue de brindar el mayor número de facilidades al usuario (hasta nuestras posibilidades y del presente trabajo), se extendió a la posibilidad de personalizar su proceso productivo, con las imágenes de procesos, la librería creada por el usuario; o no hacer uso de esto y usar el formato estándar (que es mucho más universal, pero al mismo tiempo más abstracto).

Con esto queda expuesto cómo el usuario puede importar imágenes de fuera de PSI a su interior, creando su propia librería, personalizando su proceso productivo. La mecánica exacta de cómo hacer esto, la puede encontrar en los manuales de Windows o el Manual del Usuario.

NOTA: Tenga cuidado de las imágenes que copia, no todo tiene la virtud de ser reproducido, y entra en las legislaciones del derecho de autor.

c. Documentos de resultados

Ahora explicaremos qué utilidad tiene el exportar imágenes de PSI. Suponiendo que se está haciendo un estudio sobre el diseño de un nuevo proceso productivo, una vez realizada la simulación, y analizado los resultados de la misma, se procede a realizar un documento que explique las condiciones del proceso, en ese momento se puede tomar una copia del proceso, o de una parte crítica de él. Ya que se encuentra en el portapapeles, se puede insertar en un procesador de textos como Write o Word para Windows, o pegarlo en Paint Brush, grabar la imagen, e importarla a cualquier procesador que soporte imágenes. Se imprime y se entrega, la presentación que logra es indiscutible, porque vale más una imagen que mil palabras.

El aspecto de multi tareas desarrollado por Windows, se relaciona completamente con lo anterior expuesto, pues se pueden tener corriendo al mismo tiempo el procesador de textos, y PSI para compartir la información.

Los límites que tiene una herramienta de interfaz gráfica, sólo están en la creatividad del usuario y, claro, del equipo con que cuente.

IV. ANÁLISIS Y DISEÑO

1. Análisis del problema

La definición del problema fue: *desarrollar programas para descripción gráfica de la estructura y funcionamiento de los sistemas de manufactura.*

Los objetivos se establecieron como:

- Desarrollo de un *método* para describir explícitamente los sistemas de manufactura.
- Realización de un *traductor* de pseudocódigo en el que se especifica el flujo del proceso.
- Análisis y diseño orientado a objetos para todo lo realizado (*OOA* y *OOD*).
- Establecer un medio de comunicación con aplicaciones de simulación (convertidores de código hacia aplicaciones de simulación en general).

Para poder establecer un método para describir los sistemas de manufactura tenemos que observar los elementos existentes:

> En general, podemos decir que existen 3 elementos básicos en una planta de manufactura: La(s) materia(s) prima(s), el proceso de manufactura y el(los) producto(s) terminado(s).
> El primer elemento básico del proceso productivo (la materia prima) comprende los materiales iniciales, transporte inicial, inventario inicial, etcétera.
> El segundo elemento básico (el proceso de manufactura), se compone de todo lo que interviene en la transformación de la materia prima para llegar a un producto terminado. Dentro de estos componentes podemos mencionar: las máquinas que intervienen, los operarios de dichas máquinas, medios de transporte del producto no terminado, inspección, inventarios intermedios, reglas de producción (*schedules*), flujo del proceso para un producto determinado, localización física de máquinas y espacios

destinados para inventarios o almacenes intermedios, características del producto respecto a la parte del proceso, etcétera.
El último elemento básico (el producto terminado), tiene como componentes al producto terminado con los requerimientos finales, transporte al inventario final.

Es importante manejar en todo el proceso productivo el costo acumulado en función al tiempo y de esta forma poder establecer el costo por unidad.

Ante lo anteriormente expuesto, nos podemos dar cuenta que principalmente estamos hablando de: partes o productos (en cualquier parte del proceso, llamadas también transacciones u objetos principales de producción), máquinas o elementos transformadores (que ayudan a hacer algún tipo de transformación en las partes; aplicando el concepto de caja negra con varias entradas y varias salidas), recursos (que son necesarios para la operación de los elementos transformadores), localización y transporte (que se refiere a la localización física de las entidades y su relación logística con las demás).

De esta forma, tenemos las siguientes partes:

a) Descripción física de la planta productiva.
b) Definición de los componentes del proceso productivo . Observándose: áreas de inventario intermedio, almacenes, transportes disponibles, inspección (control de calidad), materia(s) prima(s), etcétera.
c) Definición de los productos a realizar mediante los componentes arriba establecidos. Aquí se hace el *flujo del proceso* (ruteado) con las reglas correspondientes y definiciones de transporte entre una unidad y otra. Además se podrían definir las interrupciones en el proceso, así como, el ritmo de trabajo establecido (*process schedule*).

Después de este pequeño análisis, podemos establecer los principales elementos (que comenzaremos a llamar bloques) dentro de un modelo de simulación discreta para procesos de manufactura:

a) Partes. Elementos transformados durante el proceso productivo.
b) Generadores. Lugar de donde emanan las partes.
c) Colas. Almacenamiento de partes.
d) Servidores. Procesador de partes (máquinas, operación realizada sobre la parte, transporte de las partes, etcétera).

Los servidores presentan, según el número de entidades que entran y salen, lo siguiente:

- Procesamiento de 1 sola entidad (1 ⇨ 1).
- Procesamiento por lotes (N ⇨ N).
- Ensamblado (N ⇨ 1).
- Cortado de piezas, troquelado, recortado, punzonado, etcétera (1 ⇨ N).

- Combinación, mezclado de N piezas logrando M piezas finales (N ⇨ M), que se obtiene de combinar los anteriores incisos.

e) Recursos. Elementos necesarios para el funcionamiento de cualquier bloque dentro del proceso.
f) Ruteadores. Elementos que establecen las secuencias de los procesos.
g) Terminadores. Salida de las partes del sistema modelado.

Con esto en mente, pasemos a analizar y diseñar el programa.

2. Análisis y diseño del sistema

Actualmente existen diversos métodos para analizar y diseñar sistemas orientados a objetos; sin embargo, no existe un método estándar que nos proporcione la mejor descripción del sistema.

Existen prinicipalmente 8 métodos para analizar y diseñar sistemas orientados a objetos:

Booch	Rumbaugh
Coad/Yourdon	Shlaer/Mellor
Edwards/Odell/Martin	Wasserman/Pircher
Graham	Wirfs-Brock

Sin embargo, no existe un método estándar, por lo que nosotros realizaremos los siguientes pasos en el desarrollo de nuestro sistema:

a. Requerimientos del sistema.
b. Modelo de la interfaz externa.
c. Modelo de clases de interfaz.
d. Modelo de clases de aplicación.
e. Revisión de clases.

Los primeros cuatro pasos corresponden a análisis y el último a diseño.

a. Requerimientos del sistema

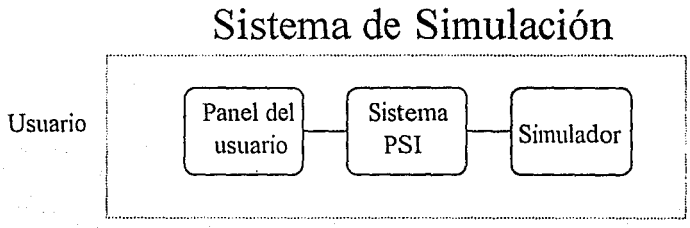
1) Objetivos
Los requerimientos del sistema son:

- Proveer un *método* para describir explícitamente los sistemas de manufactura.
- Un *traductor* de pseudocódigo en el que se especifique el flujo del proceso productivo.
- Establecer un medio de comunicación con aplicaciones de simulación (convertidores de código hacia aplicaciones de simulación en general).

- Un ambiente gráfico que proporcione facilidad para la definición del flujo productivo.

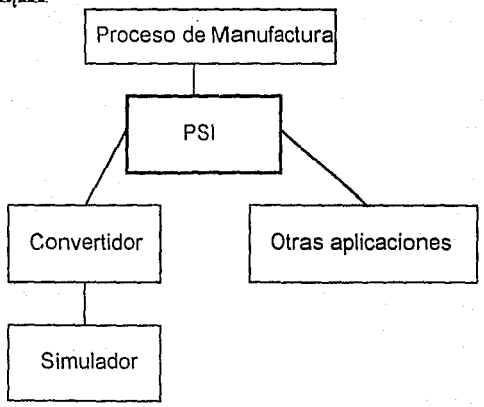
El sistema se llamará *Process Simulation Interface* (PSI).
El sistema será realizado bajo el ambiente de *Windows*, en el lenguaje Pascal (utilizando la librería *Object Windows*).

2) Dibujos del sistema
Representación física I



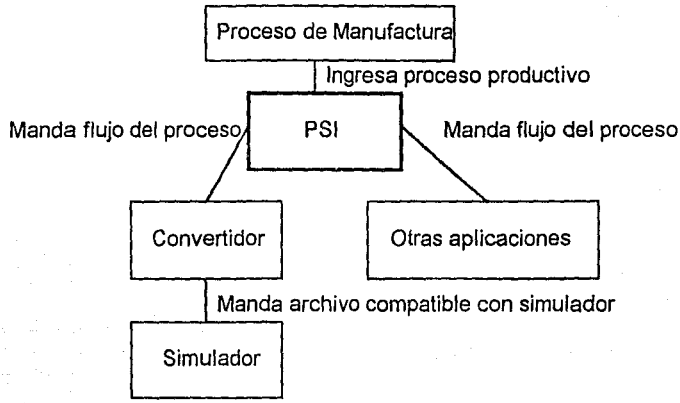
Representación física II

3) Diagrama de bloques



Localizamos 3 subsistemas: PSI, los convertidores, y el simulador.

b. Modelo de la interfaz externa



c. Modelo de clases de la interfaz

Debido a que utilizaremos la librería de *Object Windows*, todos los objetos ya existentes tienen en su nombre una letra T que precede al nombre; por ejemplo: *TWindow*. Los objetos diseñados por nosotros tendrán una letra O precediendo al nombre del mismo (así como los punteros tendrán una P); por ejemplo: *O_FrameWin*. Las especificaciones de lenguajes se muestran en detalle en el diseño de PSI.

1) Árbol de herencias de las clases de la interfaz

```
TDialog
        O_DlgConvert
        O_DlgGetAttr
        O_DlgGetInfo
                O_DlgGetInfoDiv
                O_DlgGetInfoFont
                O_DlgGetInfoGen
                O_DlgGetInfoQueue
                O_DlgGetInfoRes
                O_DlgGetInfoSrv
                        O_DlgGetInfoAss
                O_DlgGetInfoTer
```

O_DlgMoveGraph
O_DlgPasion
O_DlgPen
O_DlgPreview
O_DlgPrn
O_DlgSize

TObject
TPrintOut
O_TextPrint
O_Pen
O_UsrBMP
O_Tool
O_ToolBtn
O_ToolSpacer

TScroller
O_CanvasScrl

TWindow
O_Canvas
O_EditTB
O_GraphTB
O_LogDefWin
O_StBar
O_ViewBMP
TControl
TStatic
TEdit
O_PSIEditor
TEditWindow
TFileWindow
O_EditLangWin
TMDIWindow
O_FrameWin

2) Diagrama de Componentes Compuestos

Objeto	Clase	SuperClase	Notas
PSIApp	O_PSIApp	TApplication	
MainWindow	^TWindow	TWindowsObject	Enlace dinámico: O_FrameWin

Childlist	^TCollection	TObject	OWL. Enlace dinámico de elementos. Variable Child
Child	^O_EditLangWin	TWindow	
GraphWnd	^O_LogDefWin	TWindow	Puntero de referencia a la ventana gráfica correspondiente
ToolBar	^O_EditTB	TWindow	
Tools	TCollection	TObject	
Child	^O_LogDefWin	TWindow	
ToolBar	^O_GraphTB	TWindow	
Canvas	^O_Canvas	TWindow	
Scroller	^TScroller	TObject	

3) Diagrama de Componentes Compuestos por clase/Estructura

Clase/Esctructura	Atributo	Clase del atributo	Notas
E_State. Estructura que sirve tanto a objetos de interfaz como a objetos de aplicación	BezierPen	^O_Pen	
	Block	^O_Block	Enlace dinámico a cualquier subtipo
	Blocks	^O_CollecBlks	Colección de bloques y colecciones en el programa
	Canvas	^TWindow	Enlace dinámico a O_Canvas
	EditWin	^TWindow	Puntero de referencia a la ventana de edición de texto correspondiente. Enlace dinámico a: O_EditLangWin
	SelBlocks	^O_Rule	
	SimObj	^O_SimObj	
O_Canvas	Scroller	^TScroller	Enlace dinámico a: O_CanvasScroller
	State	^E_State	Referencia al estado del sistema.
O_DlgGetAttr	Blk	^O_SimObj	
O_DlgGetInfo	Blk	^O_Block	

O_DlgSize	State	^State	Referencia al estado del sistema
O_EditLangWin	GraphWnd	^O_LogDefWin	
	Toolbar	^O_GraphTB	
O_EditTB	Tools	TCollection	
O_FrameWin	ChildList	^TCollection	Enlaces dinámicos a dos tipos de ventana: O_EditLangWin O_LogDefWin
	StBar	^O_StBar	
	Printer	^TPrinter	
O_GraphTB	State	^E_State	Referencia al estado del sistema
O_LogDefWin	Canvas	^O_Canvas	
	Toolbar	^O_GraphTB	
	State	E_State	Estado del sistema
O_PreviewDialog	Bitmap	^O_UsrBMP	
O_PSIApp	MainWindow	^TWindow	
O_Tool	Parent	^TWindowsObject	
O_TextPrint	TheLines	^O_CharCollection	
O_ViewBMP	BMP	^O_UsrBMP	

La variable de estado contiene entre otras cosas: el diagrama del flujo, tamaño de la pantalla, los bloques seleccionados, etcétera. Esto se describirá posteriormente; por el momento basta decir que existe una sola variable y muchas referencias a ésta.

4) Definición básica de clases de interfaz

Para efectos de análisis mostramos sólo algunas de las declaraciones de los objetos de interfaz:

Clase:	O_DlgConvert
Descripción de la clase:	Diálogo que muestra los convertidores instalados en una lista. Los convertidores se leen del archivo WIN.INI con el formato: NombreDelConvertidor=CaminoDondeSeEncuentra Por ejemplo: Convertidor de PASION = C:\PSI\PASION.DLL
Superclase:	TDialog
Objetos de la clase:	Construcción en O_LogDefWin.CMConvert
Recurso de windows:	CONVS

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
NomCnv	Nombre del Convertidor	PChar		

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Init	Constructor	aParent CnvSel	PWindowsObj PChar	Ventana padre y el convertidor seleccionado
FillName	Copia el contenido de la selección a NomCnv			

Mensajes

Tipo de Msg.	Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Local	IDLb1	Manejador de la lista de convertidores	Msg	TMessage	Cuando el usuario activa la lista se manda llamar este servicio
Windows	WMInitDialog	Inicializador del diálogo. Se carga la lista de convertidores	Msg	TMessage	

Clase:	O_DlgGetAttr
Descripción de la clase:	Diálogo en el que se ingresan los tipos y atributos de la parte procesada. Son dos cajas de edición con botones de aceptar y cancelar.
Superclase:	TDialog
Objetos de la clase:	Utilizado en O_LogDefWin.CMAttr

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
Blk	Bloque de parte	PO_SimObj	<> NIL	Valor ingresado en INIT

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Init	Constructor	aParent aBlk	PWindowsObject PO_SimObj	

Mensajes

Tipo de Msg	Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Local	Ok	Servicio llamado cuando oprimen el botón de Aceptar	Msg	TMessage	Se graban las ediciones hechas en el objeto
Windows	WMInitDialog	Servicio llamado al iniciar el diálogo	Msg	TMessage	Se inicializan las cajas de edición con los valores de O_SimObj

Clase:	O_DlgGetInfo
Descripción de la clase:	Comandos básicos para un diálogo que tenga como finalidad capturar la información de un bloque. Contempla los botones de cambio de imagen, ingresado de recursos.
Superclase:	TDialog
Subclase:	O_DlgGetInfoDiv, O_DlgGetInfoFont, O_DlgGetInfoGen, O_DlgGetInfoQueue, O_DlgGetAttrRes, O_DlgGetInfoSrv
Objetos de la clase:	Clase abstracta

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
Blk	El bloque al que se le capturan datos	PO_Block		NIL

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Init	Constructor	aParent aBlk	PWindowsObject PO_Block	

SetStr	Procedimiento que se encarga de copiar a una variable PChar el contenido de una caja de edición	Id VAR Str	INTEGER PChar	La rutina crea espacio en el heap para el puntero Str

Mensajes

Tipo de Msg	Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Local	ChgBMP	Oprime el botón de cambiar BMP	Msg	TMessage	
Local	SetResources	Oprime el botón de recursos utilizados	Msg	TMessage	

Clase:	O_DlgGetInfoDiv
Descripción de la clase:	Diálogo para capturar la información de un divisor.
Superclase:	O_DlgGetInfo
Objetos de la clase:	Se crea en O_Divider.GetInfoFromUser

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
Id	Identificador del bloque en uso	INTEGER	1..32767	
DefByUsr	Valor de la regla, si es definida por usuario	PChar		NIL
RuleApp	Relga que se aplica	^TD_Rule	TD_Rule	ByPriority

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Init	Constructor	aParent VAR aRule aDefByUsr	PWindowsObject TD_Rule PChar	

Mensajes

Tipo de Msg	Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Local	Ok	Oprime botón de aceptar	Msg	TMessage	Se copian los valores al bloque
Windows	WMInitDialog	Inicializa el diálogo	Msg	TMessage	Se inicializa el diálogo con los valores del bloque

Clase:	O_DlgGetInfoFont
Descripción de la clase:	Se encarga de obtener la información del bloque tipo texto. Su declaración es muy parecida a la de los demás diálogos, por lo que sólo declaramos su existencia.
Superclase:	O_DlgGetInfo
Objetos de la clase:	Se utiliza en O_Text.GetInfoFromUser

Clase:	O_DlgGetInfoGen
Descripción de la clase:	Diálogo que se encarga de obtener la información de un bloque tipo generador.
Superclase:	O_DlgGetInfo
Objetos de la clase:	Se utiliza en O_Generator.GetInfoFromUser.

Clase:	O_DlgGetInfoQueue
Descripción de la clase:	Diálogo que se encarga de obtener la información de un bloque tipo cola.
Superclase:	O_DlgGetInfo
Objetos de la clase:	Se utiliza en O_Queue.GetInfoFromUser

Clase:	O_DlgGetAttrRes
Descripción de la clase:	Diálogo que se encarga de capturar los atributos de un recurso.
Superclase:	O_DlgGetInfo
Objetos de la clase:	Se utiliza en O_Resource.GetInfoFromUser

Clase:	O_DlgGetInfoSrv
Descripción de la clase:	Diálogo que captura la información de un bloque tipo servidor.
Superclase:	O_DlgGetInfo
Subclase:	O_DlgGetInfoAss
Objetos de la clase:	Se utiliza en O_SingleOp.GetInfoFromUser

Clase:	O_DlgGetInfoAss
Descripción de la clase:	Diálogo que captura la información de un bloque ensamblador.
Superclase:	O_DlgGetInfoSrv
Objetos de la clase:	Se utiliza en O_Assembler.GetInfoFromUser

Clase:	O_DlgGetInfoTer
Descripción de la clase:	Diálogo que captura la información de un bloque terminador.
Superclase:	O_DlgGetInfo
Objetos de la clase:	Se utiliza en O_Terminator.GetInfoFromUser

Clase:	O_DlgMoveGraph
Descripción de la clase:	Diálogo que pide la información necesaria para mover un serie de bloques. Pide los valores de X,Y además del tipo de movimiento que se hace: absoluto o relativo.
Superclase:	TDialog
Objetos de la clase:	Se utiliza en: O_LogDefWin.CMMoveGraph O_LogDefWin.CMAlign

Clase:	O_DlgPasion
Descripción de la clase:	Diálogo que pide los datos necesarios para el convertidor de PASION.
Superclase:	TDialog
Objetos de la clase:	Se utiliza en: O_LogDefWin.CMConvert

Clase:	O_DlgPen
Descripción de la clase:	Diálogo que captura el tipo de plumilla utilizada en las líneas.
Superclase:	TDialog
Objetos de la clase:	Se utiliza en: O_Pen.ChangePen

Clase:	O_DlgPreview
Descripción de la clase:	Diálogo que muestra los archivos BMP.
Superclase:	TDialog
Objetos de la clase:	Se utiliza en: O_DlgGetInfo.ChgBMP

Clase:	O_DlgPrn
Descripción de la clase:	Diálogo donde se ingresa el tipo de impresión.
Superclase:	TDialog
Objetos de la clase:	Se utiliza en: O_LogDefWin.CMPrnRes

Clase:	O_DlgSize
Descripción de la clase:	Diálogo que captura el tamaño del área de edición.

Superclase:	TDialog
Objetos de la clase:	Se utiliza en: O_LogDefWin.CMOptionSize

Clase:	O_TextPrint
Descripción de la clase:	Objeto para la impresión de un archivo de texto.
Superclase:	TPrintOut
Objetos de la clase:	Se utiliza en O_EditLangWin.PrnRes

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
TextHeight	Altura promedio del tipo de letra utilizado	INTEGER		Altura del font activo
LinesPerPage	Líneas por página	INTEGER		
FirstOnPage	Posición de la primera línea en la página	INTEGER		
LastOnPage	Posición de la última línea en la página	INTEGER		
TheLines	Una colección de elementos PChar	O_CharCollec		

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Init	Constructor	aTitle FileName	PChar PChar	
InitEdit	Constructor	aTitle aEdit	PChar PEdit	
Done	Destructor			Destruye la colección TheLines. Virtual
GetDialogInfo	Function:BOOLEAN	VAR Pages	INTEGER	Virtual
HasNextPage	Function:BOOLEAN	Page	WORD	Virtual
PrintPage		Page Rect Flags	WORD TRect WORD	Virtual
ReadFile	Ingresa el archivo en la colección de líneas			
ReadFromEdit	Ingresa las líneas de una caja de edición a la colección de líneas			

SetPrintParams		ADC aSize	HDC TPoint	Virtual

Clase:	O_Pen
Descripción de la clase:	Tipo de plumilla que muestra un diálogo para escoger la nueva plumilla.
Superclase:	TObject
Objetos de la clase:	E_State.BezierPen Se utiliza en O_Union.MouseDown

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
ChangePen	Muestra un diálogo para cambiar el tipo de plumilla	PenData Width Color	E_PenData INTEGER Longint	

Clase:	O_UsrBMP
Descripción de la clase:	Clase que se encarga de definir las operaciones de un mapa de bits (*Bitmap*).
Superclase:	TObject
Objetos de la clase:	O_Object.*Bitmap*, O_ViewBMP.*Bitmap*, O_PreviewDialog.*Bitmap*

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
BitmapHandle		HBitmap		0
BMP2	*Bitmap* para copia	HBitmap		0
CopyName		PChar		NIL
FileName	Nombre del archivo, en caso de ser un archivo externo	ARRAY[0 .. fsPathName] OF CHAR		"
IsExtFile	Indicador para archivos externos	BOOLEAN		False
IsPrinting	Indicador para pintado	^BOOLEAN		NIL
Mode	Modo de copia en el DC	WORD		SrcCopy
Offset	Offset del archivo	LONGINT		0
OrgHeight	Altura original del BMP	INTEGER		0
OrgWidth	Ancho original del BMP	INTEGER		0
PixelHeight	Altura actual del BMP	INTEGER		0
PixelWidth	Ancho actual del BMP	INTEGER		0

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Init	Constructor	Name anIsPrinting	PChar ^BOOLEAN	
InitBMP	Constructor	aBMP Name anIsPrinting	HBitmap PChar ^BOOLEAN	
Done	Destructor			Destruye BitmapHandle Virtual
CopyFrom	Copia el *bitmap*	aBMPHandle	HBitmap	
DrawUsrBMP	Dibuja el *bitmap*	DC X Y aWnd	HDC INTEGER INTEGER HDC	
HugeIO	Función para leer más de 64 KBytes	...		
LoadBitmapFile		WhatWindow	HDC	
OpenDIB		aFile	INTEGER	
Reload	Repite la lectura del BMP	WhatWindow	HDC	
Resize	Cambia el tamaño	aNewHeight aNewWidth	INTEGER INTEGER	
SetBitmap		aBmpHandle	HBitmap	
SetFile		Path	PChar	

Mensajes

Tipo de Msg	Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Stream	Load		VAR S	TStream	Virtual
Stream	Store		VAR S	TStream	Virtual

Clase:	O_Tool
Descripción de la clase:	Objeto que define un botón de la barra de utilería.
Superclase:	TObject
Subclase:	O_ToolBtn, O_ToolSpacer

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
Parent		PWindowsObject		NIL

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Init	Constructor	aParent	PWindowsObject	
BeginCapture		P	TPoint	
ContinueCapture		SendTo P	HWnd TPoint	
GetHeight	Function:INTEGER			
GetWidth	Function:INTEGER			
HasCommand	Function:BOOLEAN	Command	WORD	
HitTest	Function:BOOLEAN	P	TPoint	
Paint	Pintar botón	PaintDC PaintStruct	HDC PaintStruct	
SetOrigin	Poner el origen del botón	X Y	INTEGER INTEGER	

Mensajes

Tipo de Msg	Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Stream	Read		VAR S	TStream	Virtual
Stream	Write		VAR S	TStream	Virtual

Clase:	O_ToolBtn
Descripción de la clase:	Botón con un BMP.
Superclase:	O_Tool
Objetos de la clase:	O_EditTB.Tools

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
bmGlyph	Representación	HBitmap		0
CapDC	DC de captura	HDC		0
Capturing	Está siendo capturado	BOOLEAN		False
Command		WORD		0
GlypSize		TPoint		0,0

IsPressed	Indicador de estado	BOOLEAN		False
IsEnabled	Indicador de activación	BOOLEAN		False
MemDC	DC en memoria para copia	HDC		0

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Init	Constructor	aParent X Y aCommand	PWindowsObject INTEGER INTEGER WORD	
Done	Destructor			Destruye la representación gráfica. Virtual
BeginCapture		P	TPoint	Virtual
ContinueCapture		P	TPoint	Virtual
Enable		State	BOOLEAN	
EndCapture		SendTo P	HWnd TPoint	
GetHeight	Function: INTEGER			Virtual
GetWidth	Function: INTEGER			Virtual
HasCommand	Function: BOOLEAN	aCommand	WORD	Virtual
HitTest	Function: BOOLEAN	P	TPoint	Virtual
Paint		aDC aMemDC PS	HDC HDC TPaintStruct	
PressIn				
PressOut				
SetOrigin		X Y	INTEGER INTEGER	

Mensajes

Tipo de Msg	Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Read			VAR S	TStream	
Write			VAR S	TStream	

Clase:	O_ToolSpacer
Descripción de la clase:	Espacio en la barra de utilidades.

Superclase:	O_Tool
Objetos de la clase:	O_EditTB.Tools

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
Size	Tamaño del espacio	INTEGER	Menor ancho de la pantalla	32

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Init	Constructor	aParent aSize	PWindowsObject INTEGER	
GetHeight	Function:INTEGER			Virtual
GetWidth	Function: INTEGER			Virtual

Clase:	O_CanvasScroller
Descripción de la clase:	Scroll para la clase O_Canvas.
Superclase:	TScroller
Objetos de la clase:	O_Canvas.Scroller

Clase:	O_Canvas
Descripción de la clase:	Pantalla gráfica de edición de bloques de tamaño variable.
Superclase:	TWindow
Objetos de la clase:	O_LogDefWin.Canvas

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
BlockSel	Bloque seleccionado	PO_Block		NIL
MoveSel	Estado de movimiento	BOOLEAN		False
ToolsCrs	Cursores de las herramientas	ARRAY[TD_ToolName] OF hCursor		
OtherState	Copia del sistema	E_State		
State	Estado del sistema. Variable de referencia	P_State	<>NIL	

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Init	Constructor	aParent anState	PWindowObject P_State	
Done	Destructor			Virtual
CanClose	Function:BOOLEAN			Virtual
ClearAll	Borra todo lo existente			
Copy	Copiado de los bloques seleccionados al portapapeles			Copia en el formato PSICF. Ver manual técnico
CopyToClp	Copiado de una imagen al portapapeles	DC Rect	HDC TRect	La imagen se encuentra desplegada en la pantalla
Cut	Copiado al portapapeles y borrado			Copy + Delete
Delete	Borrado de las entidades seleccionadas			
DisableCCD	Deshabilita copiar, cortar y borrar del menú.			Opciones de edición
EnableCCD	Habilita copiar, cortar y borrar			Opciones de edición
MoveSelf	Cambia de posición la pantalla y su *scroll*	X, Y, W, H	INTEGER	
Paint	Pintado de la ventana	PaintDC PaintStruct	HDC TPaintStruct	Virtual
Paste	Pegado			
ReadPSICF	Leer el formato *PSI Clipboard Format*	aFile	PChar	Ver el manual técnico
ReleaseSelection	Deselecciona todos los bloques			Repinta
Verify	Verificación del flujo del proceso			Utiliza O_Verify

WritePSICF	Escribir el formato *PSI Clipboard Format* de los bloques seleccionados			Ver el manual técnico

Mensajes

Tipo de Msg	Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Windows	WMLButtonDown	El botón izquierdo del ratón fue presionado	Msg	TMessage	
Windows	WMLButtonUp	El botón izquierdo del ratón no está presionado	Msg	TMessage	
Windows	WMMouseMove	Se mueve el ratón	Msg	TMessage	
Windows	WMRButtonDown	El botón derecho del ratón fue presionado	Msg	TMessage	
Windows	WMSetCursor	Activa el cursor correspondiente	Msg	TMessage	

Clase:	O_EditTB
Descripción de la clase:	Barra de herramientas para una ventana de edición de código.
Superclase:	TWindow
Objetos de la clase:	O_EditLangWin.Toolbar

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
Capture	Herramienta actual	PO_Tool		
ResName	Nombre del recurso	PChar		
Tools	Herramientas	TCollection		Enlace dinámico a objetos de tipo O_Tool

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Init	Contructor	aParent aName	PWindowsObject PChar	
Done	Destructor			Virtual
CreateTool	Function:PO_Tool	Num Origin Command Bitmap	INTEGER TPoint WORD HBitmap	
EnableTool	Activa las herramientas	Commmad NewState	WORD BOOLEAN	
NextToolOrigin		Num VAR Origin Tool	WORD TPoint PO_Tool	
Paint		PaintDC PS	HDC TPaintStruct	Virtual
ReadResource				Virtual
SwitchTo		NewName	PChar	

Mensajes

Tipo de Msg	Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Stream	Load		VAR S	TStream	Virtual
Stream	Write		VAR S	TStream	Virtual
Windows	WMLButtonDown		Msg	TMessage	Virtual
Windows	WMLButtonUp		Msg	TMessage	Virtual
Windows	WMMouseMove		Msg	TMessage	Virtual

Clase:	O_GraphTB
Descripción de la clase:	Barra de utilería gráfica.
Superclase:	TWindow
Objetos de la clase:	O_LogDefWin.Toolbar

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
anState	Copia del estado del sistema	P_State	<>NIL	
NoIconsTB	Número de herramientas	INTEGER		0

ToolsIcon	Representaciones gráficas de las herramientas	ARRAY[TD_ToolName] OF HIcon		Se cargan del recurso correspondiente

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Init	Constructor	aParent TheState	PWindowsObject P_State	
Paint		PaintDC PS	HDC TPaintStruct	
ToolSelection	Selección de herramienta	Tool	TD_ToolName	

Mensajes

Tipo de Msg	Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Windows	WMLButtonDown	Se ha seleccionado alguna herramienta	Msg	TMessage	

Clase:	O_LogDefWin
Descripción de la clase:	Ventana de definición lógica del proceso.
Superclase:	TWindow
Objetos de la clase:	O_FrameWin.ChildList (Enlace dinámico)

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
Canvas	Ventana de edición	^O_Canvas		NIL
CBChainNxt	Sig. Ventana en la cadena de ventanas de portapapeles	HWnd		0
FileName	Nombre del archivo	ARRAY[0.. fsPathName] OF CHAR		"
Toolbar	Barra de herramientas	^O_GraphTB	<>NIL	
State	Estado del sistema	E_State		

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Init	Constructor	aParent aTitle	PWindowsObject PChar	
Done	Destructor			Destruye *Canvas* y *Toolbar*. Virtual
Arrange	Acomoda los bloques			
CanClose	Function:BOOLEAN			Virtual
CreateGraph				
MoveGraph	Mover los bloques	dx dy	INTEGER INTEGER	
ReadFile	Lectura de un archivo tipo PSI			Ver manual técnico
ReadLNK	Lectura de un archivo tipo LNK			Ver manual técnico
WriteFile	Guarda el archivo tipo PSI			Ver manual técnico

Mensajes

Tipo de Msg	Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Menú	CMAlign	Alinear bloques	Msg	TMessage	
Menú	CMArrange	Acomoda bloques	Msg	TMessage	
Menú	CMAttr	Edita atributos	Msg	TMessage	
Menú	CMCompleteFlux	Completa el flujo	Msg	TMessage	
Menú	CMConvert	Convierte a un lenguaje de simulador	Msg	TMessage	
Menú	CMEditBlk	Edición del bloque	Msg	TMessage	
Menú	CMEditClear	Borrar todo	Msg	TMessage	
Menú	CMEditCopy	Copiar al portapapeles	Msg	TMessage	
Menú	CMEditCopyImage	Copia la imagen al portapapeles	Msg	TMessage	
Menú	CMEditCut	Copia y borra	Msg	TMessage	
Menú	CMEditDelete	Borrra los bloques seleccionados	Msg	TMessage	
Menú	CMEditDesel	Deselecciona todo	Msg	TMessage	

Menú	CMEditPaste	Pegar	Msg	TMessage	
Menú	CMEditSelAll	Selecciona todo	Msg	TMessage	
Menú	CMFileopen	Abrir archivo	Msg	TMessage	
Menú	CMFileSave	Guardar archivo	Msg	TMessage	
Menú	CMFileSaveAs	Guardar como	Msg	TMessage	
Menú	CMMoveGraph	Mover grafo	Msg	TMessage	
Menú	CMOptionSize	Tamaño del área de edición	Msg	TMessage	
Menú	CMRenum	Reenumera bloques	Msg	TMessage	
Menú	CMShowIds	Muestra/Esconde los identificadores	Msg	TMessage	
Menú	CMVerify	Verificar flujo	Msg	TMessage	
Windows	WMChgCBChain	Cambiar el enlace de ventanas para la información del portapapeles	Msg	TMessage	
Windows	WMChar	Alguna tecla se oprimió	Msg	TMessage	
Windows	WMDestroy	Destruye a la ventana	Msg	TMessage	
Windows	WMDrawClipBoard	Cambio el contenido del CB	Msg	TMessage	
Windows	WMMdiActivate	Ventana activa, cambia el menú	Msg	TMessage	
Windows	WMSize	Ajusta el tamaño de la ventana, junto con el Canvas y su barra de utilería	Msg	TMessage	

Clase:	O_StBar
Descripción de la clase:	Barra de estado de PSI.
Superclase:	TWindow
Objetos de la clase:	O_FrameWin.StBar

Clase:	O_ViewBMP
Descripción de la clase:	Ventana que permite observar los archivos BMP.
Superclase:	TWindow
Objetos de la clase:	O_Server.GetInfoFromUser

Clase:	O_PSIEditor
Descripción de la clase:	Caja de edición para los archivos que contienen el código.
Superclase:	TEdit
Objetos de la clase:	O_EditLangWin.Editor

Clase:	O_EditLangWin
Descripción de la clase:	Ventana de edición de código.
Superclase:	TFileWindow
Objetos de la clase:	O_FrameWin.ChildList

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
GraphWnd	Referencia a la ventana gráfica	PO_LogDefWin		NIL
Toolbar	Herramientas	PO_EditTB		

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Init	Constructor	aParent aFileName	PWindowsObject PChar	Inicializa sus atributos
Done	Destructor			Manda cerrar la ventana gráfica. Virtual

Mensajes

Tipo de Msg	Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Menú	CMCompile	Traduce código a bloques en pantalla gráfica	Msg	TMessage	
Menú	CMPrnRes	Imprime archivo	Msg	TMessage	
Windows	WMMDIActivate	Cambia menú	Msg	TMessage	
Windows	WMSize	Define el tamaño de la ventana	Msg	TMessage	

Clase:	O_FrameWin
Descripción de la clase:	Ventana principal de la aplicación.
Superclase:	TMDIWindow
Objetos de la clase:	Application^.MainWindow

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
StBar	Barra de estado	PO_StBar		NIL

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Init	Constructor	aTitle aMenu	PChar HMenu	
Done	Destructor			Destruye la barra de estado. Virtual

Mensajes

Tipo de Msg	Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Menú	CMAbout	Acerca de ...	Msg	TMessage	
Menú	WMHelp...	Mensajes del menú para ayuda	Msg	TMessage	
Menú	CMNewFile	Abrir archivo nuevo	Msg	TMessage	
Menú	CMOpenFile	Abrir archivo	Msg	TMessage	
Menú	CMPrnSetup	Configurar impresora	Msg	TMessage	
Windows	WMMenuSelect	Selección de un elemento del menú para mostrar la ayuda	Msg	TMessage	
Windows	WMSize	Establece el tamaño de la ventana	Msg	TMessage	

d. Modelo de clases de aplicación

Es necesario establecer los siguientes tipos básicos para la definición de las clases:

```
TA_Coef        = ARRAY[1..4] Of REAL;
TA_Ptos        = ARRAY[0..3] OF TPoint;
TD_Cola        = (LIFO,FIFO,RAND);
TD_Direc       = (Rt, UR, Up, UL, Lt, DL, Dn, DR, Null);
TD_IOMode      = (NotIO, InOut);
TD_MvSzMode    = (Nothing_M, Move_M, SizeX_M, SizeY_M, SizeXY_M);
TD_PrtType     = (PrtScaled, PrtOnlyDisp, PrtByPrnRes );
TD_Rule        = (Probability , Priority , ByUser);
TD_SourceDs    = (byProm, byOverlap);
TD_ToolName    = (
      SelTool {100},      UnionTool {101},    AlignTool{102},
      BeginTool1 {103},   BeginTool2 {104},   QueueTool {105},
      SingleOpTool{106},  AssemTool {107},    SumFlow {108},
      DivTool {109},      BatchTool {110},    EndTool1 {111},
      EndTool2 {112},     TransTool {113},    ConveyTool {114},
      AGVTool {115},      MotorTool {116},    CraneTool{117},
      QueueTool2 {118},   ResourceTool {119}, TextTool {120},
      BMPTool {121}
);
TSetToolN      = SET OF TD_ToolName;
```

1) Árbol de herencias de las clases de aplicación

```
TCollection
      O_CollecBlks
      O_Rule
      O_CharCollection

TObject
      O_Block
            O_BMPBlk
            O_BlockChk
                  O_Divider
                  O_Generator
                  O_Queue
                  O_Server
                        O_Assem
                        O_Batch
                        O_SingleOp
                        O_SumFlow
                        O_Transport
                              O_AGV
                              O_Convey
```

O_Crane
O_MotorTrep
O_Terminator
O_Union
O_Resource
O_Select
O_TextBlk
O_Renum
O_SimObj
O_Translate
O_Verify

2) Diagrama de Componentes Compuestos por clase/Estructura

Clase/Esctructura	Atributo	Clase del atributo	Notas
E_State	BezierPen	^O_Pen	
	Block	^O_Block	Enlace dinámico a cualquier subtipo
	Blocks	^O_CollecBlks	Colección de bloques y colecciones en el programa
	Canvas	^TWindow	Enlace dinámico a *O_Canvas*
	EditWin	^TWindow	Puntero de referencia a la ventana de edición de texto correspondiente. Enlace dinámico a: *O_EditLangWin*
	SelBlocks	^O_Rule	
	SimObj	^O_SimObj	
O_Block	Bitmap	^O_UsrBMP	
	BlockSigs	^O_Rule	
	BlockPrevs	^O_Rule	
	BlockRes	^O_Rule	
	State	^E_State	Referencia al estado del sistema
O_CollecBlks	List	Arreglo de punteros	Enlace dinámico con cada elemento
O_PSIApp	MainWindow	^TWindow	

O_Renum	State	^E_State	Referencia al estado del sistema
O_Resource	Attributes	^TStrCollection	
	Types	^TStrCollection	
O_Rule	List	Arreglo de punteros	
	State	^E_State	Referencia al estado del sistema
O_SimObj	Attributes	^TStrCollection	
O_Translate	Edit	^Edit	Enlace dinámico a: *O_EditPSI*
	Parent	^TWindowsObject	
	State	^E_State	Referencia al estado del sistema
O_Verify	Parent	^TWindowObject	
	State	^E_State	Referencia al estado del sistema

3) Definición básica de las clases de aplicación

Clase:	O_CharCollection
Descripción de la clase:	Clase para guardar punteros de caracteres (sin ordenar).
Superclase:	TCollection
Subclase:	
Objetos de la clase:	O_TextPrint

Clase:	O_CollecBlks
Descripción de la clase:	Colección de bloques.
Superclase:	TCollection
Subclase:	
Objetos de la clase:	O_LogDefWin.State.Blocks

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
CallUserFunc	Llamada del procedimiento del usuario	Action NBytes	POINTER INTEGER	Ver manual técnico para las restricciones
Del	Borrar un bloque de la lista	IdBlk	INTEGER	

DelBlksIn	Borrar los bloques que se encuentran en una colección de identificadores (*O_Rule*)	CollecIds	PO_Rule	
FirstThatIn	El primer bloque que se encuentra en la colección de identificadores que cumple con la condición del usuario. Function: PO_Block	CollecIds Action	PO_Rule POINTER	Se maneja de forma similar a *FirstThat*
FirstNotUnion	Primer bloque distinto de unión que cumple con la condición. Function: PO_Block	Action	POINTER	Se maneja de forma similar a *FirstThat*
FirstUnionThat	Primer bloque de unión que cumple con una condición. Function: PO_Union	Action	POINTER	Se maneja de forma similar a *FirstThat*
ForEachIn	Para todos los bloques que estén dentro de una colección de Id's, se ejecuta una accción	CollecIds Action	PO_Rule POINTER	Se maneja de forma similar a *ForEach*
GetId	Obtiene el bloque con el identificador dado. Function:PO_Block	Id	INTEGER	En caso de no existir devuelve nil
IsAlready	Function:BOOLEAN	Id	INTEGER	Utiliza *GetId*

Clase:	O_Rule
Descripción de la clase:	Colección de identificadores, con algún tipo de regla de asociación.
Superclase:	TCollection
Subclase:	
Objetos de la clase:	O_LogDefWin.State.SelBlocks, O_Block.NxtBlocks, O_Block.PrvBlocks, O_Block.ResBlocks

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
Rule	Regla de asociación	TD_Rule	ByProbability, ByPriority, ByUser	ByPriority
State	Referencia al estado del sistema	P_State		NIL
ValUsr	Valor de la regla de asociacion, si es definida por el usuario	ARRAY[0..50] OF CHAR		"

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Init	Constructor	aLimit aDelta anState	INTEGER INTEGER P_State	anState es una copia del estado del sistema
Done	Destructor			Virtual
Del	Borrar de la colección un identificador	IdDel	INTEGER	
DeselectAll	Deselecciona todos los bloques dentro de esta colección			Manda llamar Deselect de cada bloque
GetFirstLinkable	Obtiene el primer bloque enlazable. Funciton:PO_Block			Si no existe devuelve NIL
GetId	Obtiene el bloque con el identificador especificado. Function:PO_Block	Id	INTEGER	Si no existe devuelve NIL
GetSecLinkable	Obtiene el segundo bloque enlazable			Si no existe devuelve NIL
HowManyLinkable	Cuenta el número de bloques enlazables. Function:INTEGER;			
SetWeight	Se especifica el valor de la regla para el elemento con el identificador dado	Id W	INTEGER REAL	

Mensajes

Tipo de Msg	Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Stream	Load	Cargar la lista	VAR S	TStream	
Stream	Store	Guardar la lista	VAR S	TStream	

Clase:	O_Block
Descripción de la clase:	Clase que define el comportamiento básico de un bloque.
Superclase:	TObject
Subclase:	O_BMPBLK,O_BlockChk,O_Resource,O_Select
Objetos de la clase:	O_LogDefWin.State.Block, O_LogDefwin.State.Blocks^.Items

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
IdNum	Identificador del bloque	INTEGER	1..32546	
Selected	Inidicador de selección	BOOLEAN		False
Pos	Posición del bloque (esquina superior izquierda)	TPoint		0,0
WH	Ancho y alto del bloque	TPoint		32,32
State	Referencia al estado del sistema	P_State	<>NIL	
Bitmap	Representación gráfica	PO_UsrBMP		NIL
NxtBlocks	Bloques siguientes	PO_Rule	<>NIL	
PrvBlocks	Bloques anteriores	PO_Rule	<>NIL	
ResBlocks	Recursos utilizados	PO_Rule	<>NIL	

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Init	Constructor	aState aBMP	P_State PChar	
Done	Destructor			Borra las colecciones asociadas y su BMP. Virtual
ChangeIdTo	Cambia el identificador del bloque	NewId	INTEGER	<>0

CheckUnions	Revisa las conexiones			Virtual
Deselect	Deselecciona el bloque			Virtual
GenCode	Genera el código correspondiente al bloque	EditWnd	PFileWindow	Virtual
GenCodeLinks	Genera el código correspondiente a las uniones del bloque	EditWnd	PFileWindow	Virtual
GetInfoFromUser	Captura la información del bloque			Virtual
IsIn	Determina si un punto está dentro del bloque. Function: BOOLEAN	X Y	INTEGER INTEGER	
IsLinkable	Determina si el tipo de bloque es enlazable. Function: BOOLEAN			Virtual
IsLinkableIn	Determina si el tipo de bloque puede ser enlazado por otro bloque. Function: BOOLEAN			Virtual
IsLinkableOut	Determina si el tipo de bloque puede enlazar a otro bloque. Function: BOOLEAN			Virtual
LinkWith	Enlaza con el bloque especificado	Id	INTEGER	
NeedToBeDeleted	Determina si necesita ser borrado	IdDel	INTEGER	Virtual
NotifyChange	Notifica de cambios en identificadores a las colecciones internas	IdOld IdNew	INTEGER INTEGER	
Paint		PaintDC PS	HDC TPaintStruct	Virtual
ReadComm	Lee los comandos establecidos en la caja de edición	aLine VAR NumLine Edit	PChar INTEGER PEdit	Virtual
Select	Selecciona el bloque			Virtual
TypeTool	Obtiene el tipo de bloque. Function:TD_ToolName			Virtual

Mensajes

Tipo de Msg	Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Stream	Load	Cargar la información del bloque	VAR S	TStream	Virtual
Stream	Store	Guardar la información del bloque	VAR S	TStream	Virtual

Clase:	O_BMPBlk
Descripción de la clase:	Bloque que permite poner imágenes.
Superclase:	O_Block
Objetos de la clase:	O_LogDefWin.State.Blocks^.Items

Clase:	O_BlockChk
Descripción de la clase:	Bloque que tiene la capacidad de revisar sus uniones con otros bloques.
Superclase:	O_Block
Subclase:	O_Divider, O_Generator, O_Queue, O_Terminator, O_Union
Objetos de la clase:	Clase abstracta

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
CheckWith	Revisa si las uniones ya realizadas son válidas; en caso de no serlo, entonces borra la unión y genera los enlaces lógicos correspondientes	Prevs WhichNot TypeCompatilbe	BOOLEAN TSetToolN TD_ToolName	TSetToolN debe ser un conjunto de ToolNames
BuildUnions	Construye las uniones correspondientes al bloque			Virtual

Clase:	O_Divider
Descripción de la clase:	Divisor de flujos.
Superclase:	O_BlockChk
Subclase:	
Objetos de la clase:	O_LogDefWin.State.Blocks^.Items

Clase:	O_Generator
Descripción de la clase:	Generador de partes.
Superclase:	O_BlockChk
Objetos de la clase:	O_LogDefWin.State.Blocks^.Items

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
Distribution	Distribución	PChar	Expresión de Pascal	'NEGEXP(1.0)
GenGroups	Número de elementos por grupo	PChar	Expresión de Pascal	'1.0'
InitTime	Tiempo inicial para generar la primera parte	PChar	Expresión de Pascal	'0'
MaxNumber	Máximo número a generar de entidades	PChar	Expresión de Pascal	'0'

Clase:	O_Queue
Descripción de la clase:	Bloque que representa una cola de partes.
Superclase:	O_BlockChk
Objetos de la clase:	O_LogDefWin.State.Blocks^.Items

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
CostPerUnit	Costo por unidad	PChar	Expresión de Pascal	'1.0'
InitLen	Longitud inicial	PChar	Expresión de Pascal	'0'
MaxNumber	Capacidad máxima. Si tiene un valor de cero es ilimitada	PChar	Expresión de Pascal	'0'
TypeQ	Tipo de cola (LIFO, FIFO, RAND)	TD_Cola		'NEGEXP(3.0)'

Clase:	O_Server
Descripción de la clase:	Comportamiento básico de un servidor.
Superclase:	O_BlockChk
Subclase:	O_Assem,O_Batch,O_SingleOp,O_SumFlow,O_Transport
Objetos de la clase:	Clase abstracta.

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
AddDelay	Tiempo añadido a la salida	PChar	Expresión de Pascal	'0'
CostPerTime	Costo por hora	PChar	Expresión de Pascal	'1.0'
CostPerOper	Costo por operario	PChar	Expresión de Pascal	'1.0'
Distribution	Distribución	PChar	Función de Pascal	'NEGEXP(3.0)'

Clase:	O_Assem
Descripción de la clase:	Bloque de ensamblado.
Superclase:	O_Server
Objetos de la clase:	O_LogDefWin.State.Blocks^.Items

Clase:	O_Batch
Descripción de la clase:	Bloque para procesamiento por lotes.
Superclase:	O_Server
Objetos de la clase:	O_LogDefWin.State.Blocks^.Items

Clase:	O_SingleOp
Descripción de la clase:	Servidor típico.
Superclase:	O_Server
Objetos de la clase:	O_LogDefWin.State.Blocks^.Items

Clase:	O_SumFlow
Descripción de la clase:	Sumador de flujos.
Superclase:	O_SumFlow
Objetos de la clase:	O_LogDefWin.State.Blocks^.Items

Clase:	O_Transport
Descripción de la clase:	Transporte básico.
Superclase:	O_Server
Subclase:	O_AGV, O_Convey, O_Crane, O_MotorTrsp
Objetos de la clase:	O_LogDefWin.State.Blocks^.Items

Clase:	O_Terminator
Descripción de la clase:	Terminador del flujo del proceso. Fin del modelo.
Superclase:	O_BlockChk
Objetos de la clase:	O_LogDefWin.State.Blocks^.Items

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
NoPartsDec	No. de partes que se restan	PChar	Expresión de Pascal	'0'
NoMaxParts	No. máximo de partes a recibir	PChar	Expresión de Pascal	'0'

Clase:	O_Union
Descripción de la clase:	Bloque que representa la unión de dos bloques.
Superclase:	O_BlockChk
Objetos de la clase:	O_LogDefWin.State.Blocks^.Items

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
Back	Si la dirección de la unión es inversa	BOOLEAN		False
ColLine	Color de la línea	TColorRef		Rojo brillante
EditLine	Indicador de edición de línea	BOOLEAN		False
LineData	Tipo de línea	TPenData		
Width	Ancho de la línea	INTEGER		1
ToDelete	Es una conexión no válida	BOOLEAN		False
XY	Puntos bezier que definen la curva de unión	TA_Ptos (Vector de puntos)		

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Init	Constructor	anState ShowUnion	P_State BOOLEAN	Revisa la lista de seleccionados y toma los dos primeros bloques enlazables

InitComp	Constructor sin validación	anState Id1 Id2	P_State INTEGER INTEGER	Sólo ingresa a las colecciones correspondientes los identificadores definidos
Recalc	Recalcula las coordenadas de la unión			Recalcula los puntos Bezier
UnLinkBlks	Deshacer la unión			Quita de las colecciones los valores correspondientes
... Servicios definidos para todos los bloques.	...	...	...	...

Clase:	O_Select
Descripción de la clase:	Herramienta que selecciona bloques.
Superclase:	O_Block
Objetos de la clase:	O_LogDefWin.WMLButtonDown,Move,Up.

Clase:	O_TextBlk
Descripción de la clase:	Bloque para definir un texto en la pantalla.
Superclase:	O_Block
Objetos de la clase:	O_LogDefWin.State.Blocks^.Items

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
Font	Tipo de letra	HFont		System Font
PText	Texto	PChar		NIL

Clase:	O_Renum
Descripción de la clase:	Objeto que reenumera los bloques definidos, en base a la estructura del flujo.
Superclase:	TObject
Objetos de la clase:	Se utiliza en O_LogDefWin.CMRenum

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
anState	Referencia al estado del sistema	P_State	<>NIL	

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Init	Constructor	State	P_State	
Run	Ejecuta la reenumeración			

Clase:	O_SimObj
Descripción de la clase:	Objeto que representa la parte en proceso.
Superclase:	TObject
Objetos de la clase:	O_LogDefWin.State.SimObj

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
Attributes	Atributos definidos para la parte	PStrCollection		
State	Referencia del estado del sistema		<>NIL	
Types	Tipos definidos para la parte	PStrCollection		

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Init	Constructor	anState	P_State	
Done	Destructor			Virtual
GenCode	Generación de código	EditWnd	PFileWindow	
GetInfoFromUser	Captura los tipos y atributos de la parte			
ReadFromEdit	Lee parámetros a partir de un bloque de código	aWnd Edit VAR NumLine	PWindowsObject PEdit INTEGER	

Mensajes

Tipo de Msg	Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Stream	Load	Cargar la información del bloque	VAR S	TStream	
Stream	Store	Guardar la información del bloque			

Clase:	O_Translate
Descripción de la clase:	Clase encargada de tradudir código en una representación gráfica.
Superclase:	TObject
Objetos de la clase:	Se utiliza en O_EditLangWin.CMTranslate

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
Edit	Caja de edición donde se encuentra el código	PEdit		
Parent	Ventana padre	PWindowsObject		NIL
State		P_State		

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Init	Constructor	anState	P_State	
Run	Traducir el código			Virtual

Clase:	O_Verify
Descripción de la clase:	Verificador de flujos.
Superclase:	TObject
Objetos de la clase:	Se utiliza principalmente en O_LogDefWin

Atributos

Nombre	Descripción	Tipo	Valores aceptados	Valor inicial
FullVerif	Indicador de verificación completa	BOOLEAN		True
Parent	Ventana padre	PWindowsObject		

State	Referencia del estado del sistema	P_State	<>NIL	
VerifyBE	Verificación de existencia de bloques generadores y terminadores por flujo	BOOLEAN		True

Servicios

Nombre	Descripción	Parámetros principales	Tipo	Comentarios
Init	Constructor	aParent anState FullVerifReq VerifyBEReq	PWindowsObject P_State BOOLEAN BOOLEAN	
CheckIds	Revisa los identificadores	Blk	PO_Block	Si los identificadores no existen, los borra de la lista
CheckWeights	Revisa los pesos	Blk	PO_Block	
DelInvalid	Borra los bloque inválidos	Collec	PO_Rule	
VerifyExist	Verifica la existencia de dichos bloques	SetmustExist	TSetToolN	Si no existen muestra un mensaje

e. Revisión de clases

Como ya se estableció previamente el lenguaje de programación será *Turbo Pascal para Windows*, utilizando *Object Windows Library*, por lo que a continuación mostramos la estructura básica de los principales objetos y estructuras:

```
TYPE
    P_State         = ^E_State;
    P_RuleItem      = ^E_RuleItem;
    PO_Block        = ^O_Block;
    PO_CollecBlks   = ^O_CollecBlks;
    PO_Rule         = ^O_Rule;
    PO_SimObj       = ^O_SimObj;
    PO_StBar        = ^O_StBar;
    PO_Assemble     = ^O_Assemble;
```

```
PO_Batch        = ^O_Batch;
PO_BMP          = ^O_BMPBLK;
PO_Divider      = ^O_Divider;
PO_Generator    = ^O_Generator;
PO_Queue        = ^O_Queue;
PO_Resource     = ^O_Resource;
PO_Select       = ^O_Select;
PO_Server       = ^O_Server;
PO_SingleOp     = ^O_SingleOp;
PO_SumFlow      = ^O_SumFlow;
PO_Terminator   = ^O_Terminator;
PO_Text         = ^O_TextBlk;
PO_Transport    = ^O_Transport;
PO_AGV          = ^O_AGV;
PO_Convey       = ^O_Convey;
PO_Crane        = ^O_Crane;
PO_MotorTrsp    = ^O_MotorTrsp;
PO_Union        = ^O_Union;
PO_Translate    = ^O_Translate;

E_State         = RECORD
    PenSize,ItemCount,Ds,Ds_,Dx_,Dy_,
    L_Arrow,MinX,MinY,MaxX,MaxY : INTEGER;
    Change                      : BOOLEAN;
    Font                        : HFont;
    ToolSel                     : TD_ToolName;
    Offset,XYScale              : TPoint;
    Block                       : PO_Block;
    SimObj                      : PO_SimObj;
    Blocks                      : PO_CollecBlks;
    SelBlocks                   : PO_Rule;
    BezierPen                   : PO_Pen;
    Canvas                      : PWindow;
    EditWin                     : PFileWindow;
END;

E_RuleItem      = RECORD
    Id                          : INTEGER;
    Weight                      : REAL;
END;

O_Block         = OBJECT(TObject)
    IdNum,NApunt,NApOr          : INTEGER;
```

```
FirstTime,Selected,Flag          : BOOLEAN;
DC                               : HDC;
Pos,WH                           : TPoint;
Mode                             : TD_MvSzMode;
State                            : P_State;
Bitmap                           : PO_UserBMP;
ResBlocks,NxtBlocks,PrvsBlocks   : PO_Rule;

CONSTRUCTOR     Init(AState: P_State; aBMP: PChar);
DESTRUCTOR      Done; VIRTUAL;
PROCEDURE       BuildFromCopy(VAR aStream: TStream; aDeltaIds: INTEGER;
  VAR aDx , aDy , ModeInc: INTEGER);
PROCEDURE       BuildUnions;VIRTUAL;
PROCEDURE       ChangeIdTo(NewId:INTEGER);VIRTUAL;
PROCEDURE       CheckUnions; VIRTUAL;
PROCEDURE       Deselect; VIRTUAL;
PROCEDURE       GenCode(EditWnd:PFileWindow); VIRTUAL;
PROCEDURE       GenCodeLinks(EditWnd:PFileWindow); VIRTUAL;
FUNCTION        GetClassName:PChar;VIRTUAL;
PROCEDURE       GetInfoFromUser;VIRTUAL;
FUNCTION        IsIn(X,Y:INTEGER):BOOLEAN; VIRTUAL;
FUNCTION        IsInSizeRgn(X,Y:INTEGER;  VAR  ModeBy:  TD_MvSzMode
  ):BOOLEAN;
FUNCTION        IsLinkable:BOOLEAN;VIRTUAL;
FUNCTION        IsLinkableIn:BOOLEAN; VIRTUAL;
FUNCTION        IsLinkableOut:BOOLEAN; VIRTUAL;
PROCEDURE       KeyChar(Key, Count, lParamHi: WORD);VIRTUAL;
PROCEDURE       LinkWith(ABlock:PO_Block); VIRTUAL;
PROCEDURE       Load(VAR S:TStream);VIRTUAL;
PROCEDURE       MouseDown(AWindow: HWnd; X, Y: INTEGER; AState: P_State);
  VIRTUAL;
PROCEDURE       MouseMove(X, Y: INTEGER);VIRTUAL;
PROCEDURE       MouseUp; VIRTUAL;
FUNCTION        NeedToBeDeleted(IdDel:INTEGER):BOOLEAN;VIRTUAL;
PROCEDURE       NotifyChange(IdOld,IdNew:INTEGER);VIRTUAL;
PROCEDURE       Paint(PaintDC: HDC; VAR PaintInfo: TPaintStruct);VIRTUAL;
FUNCTION        ReadComm(aLine:PChar;VAR
  NumLine:INTEGER;Edit:PEdit):BOOLEAN; VIRTUAL;
FUNCTION        ReadFromEdit(aWnd:PWindowsObject;Edit:PEdit;VAR
  NumLine:INTEGER):BOOLEAN;
PROCEDURE       Select; VIRTUAL;
PROCEDURE       SetCurs(X,Y:INTEGER); VIRTUAL;
PROCEDURE       SetRule(aRule:POINTER); VIRTUAL;
```

```
    PROCEDURE       Store(VAR S:TStream);VIRTUAL;
    FUNCTION        TypeTool:TD_ToolName; VIRTUAL;
END;

O_CollecBlks =  OBJECT(TCollection)
    FUNCTION        CallUserFunc(Action:Pointer;NBytes:Integer):BOOLEAN;
    PROCEDURE       Del(IdBlk:INTEGER);
    PROCEDURE       DelBlksIn(CollecIds:PO_Rule);
    FUNCTION        FirstThatIn(CollecIds:PO_Rule;Action:Pointer):PO_Block;
    FUNCTION        FirstNotUnion(Action:Pointer):PO_Block;
    FUNCTION        FirstUnionThat(Action:Pointer):PO_Block;
    PROCEDURE       ForEachIn(VAR CollecIds:PO_Rule;Action:Pointer);
    PROCEDURE       ForEachNotUnion(Action:Pointer);
    PROCEDURE       ForEachUnion(Action:Pointer);
    FUNCTION        GetId(Id:INTEGER):PO_Block;
    FUNCTION        GetMaxIdAvail:INTEGER;
    FUNCTION        GetMinIdAvail:INTEGER;
    FUNCTION        GetUnion(IniId,EndId:INTEGER):PO_Block;
    FUNCTION        IsAlready(Id:INTEGER):BOOLEAN;
END;

O_Rule          = OBJECT(TCollection)
    Rule                    : TD_Rule;
    ValUsr                  : ARRAY[0..50] OF CHAR;
    State                   : P_State;
    RuleItem                : P_RuleItem;

    CONSTRUCTOR     Init(ALimit,ADelta:INTEGER;AState:P_State);
    DESTRUCTOR      Done; VIRTUAL;
    PROCEDURE       Del(I:INTEGER);
    PROCEDURE       DeselectAll;
    PROCEDURE       FreeItem(Item:POINTER); VIRTUAL;
    FUNCTION        GetItem(VAR S:TStream):POINTER; VIRTUAL;
    FUNCTION        GetFirstLinkable:PO_Block;
    FUNCTION        GetId(I:INTEGER):PO_Block;
    FUNCTION        GetMaxX:INTEGER;
    FUNCTION        GetMaxY:INTEGER;
    FUNCTION        GetMinX:INTEGER;
    FUNCTION        GetMinY:INTEGER;
    FUNCTION        GetSecondLinkable:PO_Block;
    FUNCTION        GetWeight(I:INTEGER):Real;
    FUNCTION        HowManyLinkable:INTEGER;
    PROCEDURE       Ins(I:INTEGER);
```

```
    FUNCTION        IsAlready(I:INTEGER):BOOLEAN;
    PROCEDURE       Load(VAR S:TStream);
    PROCEDURE       PutItem(VAR S:TStream; Item:POINTER); VIRTUAL;
    PROCEDURE       SetWeight(I:INTEGER;W:Real);
    PROCEDURE       Store(VAR S:TStream);
END;

O_SimObj        = OBJECT(TObject)
    Types,Attributes            : PStrCollection;
    State                       : P_State;

    CONSTRUCTOR     Init(aState:P_State);
    DESTRUCTOR      Done; VIRTUAL;
    PROCEDURE       GenCode(EditWnd:PFileWindow);
    PROCEDURE       GetInfoFromUser;
    PROCEDURE       Load(VAR S:TStream); VIRTUAL;
    FUNCTION        ReadFromEdit(aWnd:PWindowsObject;Edit:PEdit;VAR
      NumLine:INTEGER):BOOLEAN;
    PROCEDURE       Store(VAR S:TStream); VIRTUAL;
END;

O_BlockChk      = OBJECT(O_Block)
    PROCEDURE       CheckWith(Prevs:    BOOLEAN;    WhichNOT:    TSetToolN;
      TypeCompatible: TD_ToolName);
    PROCEDURE       BuildUnions;VIRTUAL;
END;

O_Server        = OBJECT(O_BlockChk)
    Distribution,
    AddDelay,
    CostPerTime,
    CostPerOper                 : PChar;

    CONSTRUCTOR     Init(AState:P_State);
    DESTRUCTOR      Done;  VIRTUAL;
    PROCEDURE       CheckUnions;  VIRTUAL;
    PROCEDURE       GenCode(EditWnd:PFileWindow); VIRTUAL;
    FUNCTION        GetClassName:PChar;     VIRTUAL;
    PROCEDURE       GetInfoFromUser;    VIRTUAL;
    FUNCTION        IsLinkableOut:BOOLEAN;    VIRTUAL;
    PROCEDURE       Load(VAR S:TStream);      VIRTUAL;
    FUNCTION        ReadComm(aLine:PChar;VAR
      NumLine:INTEGER;Edit:PEdit):BOOLEAN; VIRTUAL;
```

```
    PROCEDURE       Store(VAR S:TStream);      VIRTUAL;
END;

O_Assemble      = OBJECT(O_Server)
    PROCEDURE       CheckUnions;  VIRTUAL;
    PROCEDURE       GenCode(EditWnd:PFileWindow);     VIRTUAL;
    FUNCTION        GetClassName:PChar;        VIRTUAL;
    PROCEDURE       GetInfoFromUser;     VIRTUAL;
    FUNCTION        TypeTool:TD_ToolName;      VIRTUAL;
END;

O_Batch         = OBJECT(O_Server)
    PROCEDURE       GenCode(EditWnd:PFileWindow); VIRTUAL;
    FUNCTION        GetClassName:PChar; VIRTUAL;
    FUNCTION        TypeTool:TD_ToolName; VIRTUAL;
END;

O_BMPBLK        = OBJECT(O_Block)
    PROCEDURE       GenCode(EditWnd:PFileWindow);     VIRTUAL;
    FUNCTION        GetClassName:PChar;        VIRTUAL;
    FUNCTION        IsLinkable:BOOLEAN;  VIRTUAL;
    PROCEDURE       Paint(PaintDC: HDC; VAR PaintInfo: TPaintStruct);VIRTUAL;
    FUNCTION        TypeTool:TD_ToolName;      VIRTUAL;
END;

O_Divider       = OBJECT(O_BlockChk)
    CONSTRUCTOR     Init(AState:P_State);
    PROCEDURE       CheckUnions;  VIRTUAL;
    PROCEDURE       GenCode(EditWnd:PFileWindow);     VIRTUAL;
    FUNCTION        GetClassName:PChar;        VIRTUAL;
    PROCEDURE       GetInfoFromUser;     VIRTUAL;
    FUNCTION        IsLinkableIn:BOOLEAN;      VIRTUAL;
    FUNCTION        ReadComm(aLine:PChar;VAR
      NumLine:INTEGER;Edit:PEdit):BOOLEAN; VIRTUAL;
    FUNCTION        TypeTool:TD_ToolName;      VIRTUAL;
END;

O_Generator     = OBJECT(O_BlockChk)
    Distribution                :PChar;
    InitTime                    :PChar;
    GenGroups                   :PChar;
    MaxNumber                   :PChar;
```

```
    CONSTRUCTOR     Init(AState:P_State;aBMP:PChar);
    DESTRUCTOR      Done;  VIRTUAL;
    PROCEDURE       CheckUnions;  VIRTUAL;
    PROCEDURE       GenCode(EditWnd:PFileWindow); VIRTUAL;
    FUNCTION        GetClassName:PChar;        VIRTUAL;
    PROCEDURE       GetInfoFromUser;     VIRTUAL;
    FUNCTION        IsLinkableIn:BOOLEAN;      VIRTUAL;
    FUNCTION        IsLinkableOut:BOOLEAN;     VIRTUAL;
    PROCEDURE       Load(VAR S:TStream);       VIRTUAL;
    FUNCTION        ReadComm(aLine:PChar;VAR
      NumLine:INTEGER;Edit:PEdit):BOOLEAN; VIRTUAL;
    PROCEDURE       Store(VAR S:TStream);      VIRTUAL;
    FUNCTION        TypeTool:TD_ToolName;      VIRTUAL;
END;

O_Queue         = OBJECT(O_BlockChk)
    CostPerUnit                 : PChar;
    InitLen                     : PChar;
    MaxNumber                   : PChar;
    Tipo                        : TD_Cola;

    CONSTRUCTOR Init(AState:P_State);
    DESTRUCTOR      Done; VIRTUAL;
    PROCEDURE       CheckUnions;  VIRTUAL;
    PROCEDURE       GenCode(EditWnd:PFileWindow); VIRTUAL;
    FUNCTION        GetClassName:PChar; VIRTUAL;
    PROCEDURE       GetInfoFromUser; VIRTUAL;
    FUNCTION        IsLinkableIn:BOOLEAN; VIRTUAL;
    FUNCTION        IsLinkableOut:BOOLEAN;     VIRTUAL;
    PROCEDURE       Load(VAR S:TStream); VIRTUAL;
    FUNCTION        ReadComm(aLine:PChar;VAR
      NumLine:INTEGER;Edit:PEdit):BOOLEAN; VIRTUAL;
    PROCEDURE       Store(VAR S:TStream); VIRTUAL;
    FUNCTION        TypeTool:TD_ToolName;      VIRTUAL;
END;

O_Resource      = OBJECT(O_Block)
    Types,
    Attributes                  : PStrCollection;

    CONSTRUCTOR Init(AState:P_State);
    DESTRUCTOR      Done;  VIRTUAL;
    PROCEDURE       GenCode(EditWnd:PFileWindow); VIRTUAL;
```

```
    FUNCTION        GetClassName:PChar; VIRTUAL;
    PROCEDURE       GetInfoFromUser; VIRTUAL;
    FUNCTION        IsLinkable:BOOLEAN; VIRTUAL;
    PROCEDURE       Load(VAR S:TStream);VIRTUAL;
    FUNCTION        ReadComm(aLine:PChar;VAR
      NumLine:INTEGER;Edit:PEdit):BOOLEAN; VIRTUAL;
    PROCEDURE       Store(VAR S:TStream); VIRTUAL;
    FUNCTION        TypeTool:TD_ToolName;       VIRTUAL;
END;

O_Select        = OBJECT(O_Block)
    Pen:HPen;

    CONSTRUCTOR     Init(AState:P_State);
    DESTRUCTOR      Done; VIRTUAL;
    FUNCTION        GetClassName:PChar; VIRTUAL;
    PROCEDURE       MouseDown(AWindow: HWnd; X, Y: INTEGER; AState: P_State);
      VIRTUAL;
    PROCEDURE       MouseMove(X, Y: INTEGER); VIRTUAL;
    PROCEDURE       MouseUp; VIRTUAL;
    FUNCTION        TypeTool:TD_ToolName;       VIRTUAL;
END;

O_SingleOp      = OBJECT(O_Server)
    PROCEDURE       GenCode(EditWnd:PFileWindow); VIRTUAL;
    FUNCTION        GetClassName:PChar; VIRTUAL;
    FUNCTION        TypeTool:TD_ToolName;       VIRTUAL;
END;

O_SumFlow       = OBJECT(O_Server)
    PROCEDURE       GenCode(EditWnd:PFileWindow); VIRTUAL;
    FUNCTION        GetClassName:PChar; VIRTUAL;
    FUNCTION        TypeTool:TD_ToolName; VIRTUAL;
END;

O_Terminator    = OBJECT(O_BlockChk)
    NoPartsDec,
    NoMaxParts:                 PChar;

    CONSTRUCTOR     Init(AState:P_State;aBMP:PChar);
    DESTRUCTOR      Done; VIRTUAL;
    PROCEDURE       CheckUnions; VIRTUAL;
    PROCEDURE       GenCode(EditWnd:PFileWindow); VIRTUAL;
```

```
    FUNCTION        GetClassName:PChar; VIRTUAL;
    PROCEDURE       GetInfoFromUser; VIRTUAL;
    FUNCTION        IsLinkableIn:BOOLEAN;VIRTUAL;
    FUNCTION        IsLinkableOut:BOOLEAN;      VIRTUAL;
    PROCEDURE       Load(VAR S:TStream); VIRTUAL;
    FUNCTION        ReadComm(aLine:PChar;VAR
      NumLine:INTEGER;Edit:PEdit):BOOLEAN; VIRTUAL;
    PROCEDURE       Store(VAR S:TStream); VIRTUAL;
    FUNCTION        TypeTool:TD_ToolName;       VIRTUAL;
END;

O_TextBlk       = OBJECT(O_Block)
    Font                        :HFont;
    PText                       :PChar;

    CONSTRUCTOR     Init(AState:P_State);
    DESTRUCTOR      Done; VIRTUAL;
    PROCEDURE       ChangeFont(NewOne:hFont);
    PROCEDURE       ChangeFontInd(NewOne:TLogFont);
    PROCEDURE       GenCode(EditWnd:PFileWindow); VIRTUAL;
    FUNCTION        GetClassName:PChar; VIRTUAL;
    PROCEDURE       GetInfoFromUser; VIRTUAL;
    FUNCTION        IsLinkable:BOOLEAN; VIRTUAL;
    PROCEDURE       Load(VAR S:TStream); VIRTUAL;
    PROCEDURE       MouseDown(AWindow: HWnd; X, Y: INTEGER; AState: P_State);
      VIRTUAL;
    PROCEDURE       MouseMove(X, Y: INTEGER); VIRTUAL;
    PROCEDURE       MouseUp; VIRTUAL;
    PROCEDURE       Paint(PaintDC: HDC; VAR PaintInfo: TPaintStruct);VIRTUAL;
    PROCEDURE       SetCurs(X,Y:INTEGER); VIRTUAL;
    PROCEDURE       Store(VAR S:TStream); VIRTUAL;
    FUNCTION        TypeTool:TD_ToolName;       VIRTUAL;
END;

O_Transport     = OBJECT(O_Server)
    PROCEDURE       GenCode(EditWnd:PFileWindow); VIRTUAL;
    FUNCTION        GetClassName:PChar; VIRTUAL;
    FUNCTION        TypeTool:TD_ToolName; VIRTUAL;
END;

O_AGV           = OBJECT(O_Transport)
    PROCEDURE       GenCode(EditWnd:PFileWindow); VIRTUAL;
    FUNCTION        GetClassName:PChar; VIRTUAL;
```

```
    FUNCTION        TypeTool:TD_ToolName; VIRTUAL;
END;

O_Convey        = OBJECT(O_Transport)
    FUNCTION        GetClassName:PChar; VIRTUAL;
    FUNCTION        TypeTool:TD_ToolName; VIRTUAL;
    PROCEDURE       GenCode(EditWnd:PFileWindow); VIRTUAL;
END;

O_Crane         = OBJECT(O_Transport)
    PROCEDURE       GenCode(EditWnd:PFileWindow); VIRTUAL;
    FUNCTION        GetClassName:PChar; VIRTUAL;
    FUNCTION        TypeTool:TD_ToolName; VIRTUAL;
END;

O_MotorTrsp     = OBJECT(O_Transport)
    PROCEDURE       GenCode(EditWnd:PFileWindow); VIRTUAL;
    FUNCTION        GetClassName:PChar; VIRTUAL;
    FUNCTION        TypeTool:TD_ToolName; VIRTUAL;
END;

O_Union         = OBJECT(O_BlockChk)
    XY                          :TA_Ptos;
    ToDelete,Back,EditLine      :BOOLEAN;
    OldColLine,ColLine          :TColorRef;
    WidthLine                   :INTEGER;
    LineData                    :TPenData;

    CONSTRUCTOR     Init(AState:P_State; ShowUnion: BOOLEAN);
    CONSTRUCTOR     InitComp(AState:P_State;Id1,Id2:INTEGER);
    PROCEDURE       Deselect; VIRTUAL;
    PROCEDURE       CheckUnions;VIRTUAL;
    FUNCTION        GetClassName:PChar; VIRTUAL;
    FUNCTION        IsIn(X,Y:INTEGER):BOOLEAN;      VIRTUAL;
    FUNCTION        IsInRgn(X,Y:INTEGER; P: TPoint): BOOLEAN; VIRTUAL;
    FUNCTION        IsLinkable:BOOLEAN; VIRTUAL;
    PROCEDURE       KeyChar(Key, Count, lParamHi: WORD); VIRTUAL;
    PROCEDURE       Load(VAR S:TStream); VIRTUAL;
    PROCEDURE       MouseDown(AWindow: HWnd; X, Y: INTEGER; AState: P_State);
      VIRTUAL;
    PROCEDURE       MouseMove(X, Y: INTEGER); VIRTUAL;
    PROCEDURE       MouseUp; VIRTUAL;
    FUNCTION        NeedToBeDeleted(IdDel:INTEGER):BOOLEAN; VIRTUAL;
```

```
    PROCEDURE       Paint(PaintDC: HDC; VAR PaintInfo: TPaintStruct);VIRTUAL;
    PROCEDURE       Recalc;
    PROCEDURE       Select; VIRTUAL;
    PROCEDURE       SetCure(X,Y:INTEGER); VIRTUAL;
    PROCEDURE       Store(VAR S:TStream);VIRTUAL;
    FUNCTION        TypeTool:TD_ToolName;       VIRTUAL;
    PROCEDURE       UnlinkBike;
END;

O_Translate     = OBJECT( TObject )
    Parent                      : PWindowsObject;
    State                       : P_State;
    Edit                        : PEdit;

    CONSTRUCTOR     Init(aParent:PWindowsObject;aState:P_State;anEdit:PEdit);
    PROCEDURE       Run; VIRTUAL;
END;
```

f. Comunicación con PASION

La comunicación se realiza por medio de un archivo con registros que contienen la siguiente información por bloque:

Campo	Tipo	Descripción	Valores válidos
No	Integer	Descripción de un bloque	1..32767
Typ	Char	Tipo de bloque	I=Generador,P=Split, Q=Cola,T=Fin,S=Servidor, A=Ensamble
Fl	Char	Fifo/Lifo si T=Q	F=Fifo, L=Lifo, R=Random
Dst	Str80	Distribución	
Func	Str80	Regla de selección, parámetro de grupo si es un generador	
Tin	Str80	Tiempo de comienzo si es un generador, retraso para un servidor o un ensamblador	
Suc	A[1..10] of Integer	Número de identificación de los elementos sucesores	

Pred	A[1..10] of Integer	Número de identificación de los elementos predecesores	
Max	Str10	No. a generar, terminar o máxima longitud de la cola	
Pa	Char	Tipo de regla	S=salida por probabilidad, P= salida por prioridades, I= Si longitud inicial de la cola es > 0, 0= si es servidor sólo tiene una entrada
Pxx	A[1..10] of Real	Prioridades o probabilidades de salida	
Ca	Real	Costo por unidad de tiempo	
Cb	Real	Costo por operación	

La configuración anteriormente expuesta, es totalmente compatible con la establecida por el programa PSI, por lo que su conversión es bastante sencilla.

El archivo de conversión a PASION tiene una extensión QMG.

V. PROGRAMACIÓN

Se utilizó la siguiente convención en la codificación del programa:

Palabras reservadas de PASCAL con mayúsculas.
Las minúsculas y mayúsculas para código propio.
Los tipos tendrán la siguiente notación:

B	= Byte
OO	= Boolean
R	= Real
C	= Char
L	= Longint
O_	= Clase
E_	= Estructura (Registro)
S	= Conjuntos
K	= Constante
KMN	= Constante Número Máximo
KLM	= Constante Longitud Máxima
KStr	= Constante de caracteres
D_	= Tipo definido por el programador
A_	= Vector
P_	= Apuntador
PO	= Apuntador a una clase
Str	= Arreglo de caracteres
F	= Archivo

Se respetará toda la definición y notación previa de la librería de PASCAL OWL y Windows.
Los archivos declarados para este desarrollo son los siguientes:

BasicPSI.PAS	= Contiene las definiciones principales del sistema.
Bezier.PAS	= Rutinas para el manejo de curvas Bezier.
Canvas.PAS	= Area de edición gráfica.
Convert.PAS	= Diálogo de convertidores instalados.
EditGrap.PAS	= Ventana de edición gráfica.
EditWin.PAS	= Ventana de edición de código.
I_Abst.PAS	= Contiene las definiciones de TypePsi con referencia a O_Block.
I_Collec.PAS	= Contiene las definiciones de TypePsi con referencia a colecciones.
Pen.PAS	= Plumillas del sistema.
Preview.PAS	= Diálogo de archivos formato BMP.
PrnUnit.PAS	= Clases de impresión.
PSI.PAS	= Programa principal.
RoutPSI.PAS	= Rutinas generales para el sistema.
Servers.PAS	= En esta unidad existe la definición de los bloques.
Toolbar.PAS	= Barras de utilería y herramientas.
TypePSI.PAS	= Contiene la defincón básica de O_Block y las colecciones.
UsrBMP.PAS	= Clase para el manejo de mapas de bits.
Verify.PAS	= Clase para verificación.
ViewBMP	= Ventanas de información.

1. Índice de unidades por Clases:

Clase/Estructura	Archivo
E_State	TypePSI
O_AGV	Servers
O_Assem	Servers
O_Block	TypePSI, I_Abst
O_BlockChk	Servers
O_BMPBLK	Servers
O_Canvas	Canvas
O_CanvasScroller	Canvas
O_CharCollection	PrnUnit
O_Collecblks	TypePSI, I_Collec
O_Convey	Servers
O_Crane	Servers
O_DlgAttr	TypePSI, I_Abst
O_DlgConvert	Convert
O_DlgGetInfoAssem	Servers
O_DlgGetInfoAttrRes	Servers
O_DlgGetInfoDiv	Servers
O_DlgGetInfoFont	Servers

Clase/Estructura	Archivo
O_EditLangWin	EditWin
O_EditTB	Toolbar
O_FrameWin	PSI
O_GraphTB	Toolbar
O_LogDefWin	EditGrap
O_MotorTrsp	Servers
O_Pen	Pen
O_PSIApp	PSI
O_PSIEditor	EditWin
O_Renum	Verify
O_Resource	Servers
O_Rule	TypePSI, I_Collec
O_Select	Servers
O_Server	Servers
O_SimObj	TypePSI, I_Abst
O_SingleOp	Servers
O_StBar	TypePSI, I_Abst
O_SumFlow	Servers

O_DlgGetInfoGen	Servers
O_DlgGetInfoQueue	Servers
O_DlgGetInfoSrv	Servers
O_DlgGetInfoTer	Servers
O_DlgInfo	TypePSI, I_Abst
O_DlgMoveGraph	TypePSI, I_Abst
O_DlgPasion	EditGrap
O_DlgPen	Pen
O_DlgPreview	Preview
O_DlgPrn	TypePSI, I_Abst
O_DlgSize	TypePSI, I_Abst

O_TextBlk	Servers
O_TextPrint	PrnUnit
O_Tool	Toolbar
O_ToolButton	Toolbar
O_ToolSpacer	Toolbar
O_Translate	Servers
O_Transport	Servers
O_UsrBMP	UsrBMP
O_Verify	Verify
O_ViewBMP	ViewBMP

CONCLUSIONES

Al plantearnos como objetivo de este desarrollo la realización de una metodología para la definición de procesos de manufactura, junto con el diseño de la interfaz gráfica que la soportara, nos dimos cuenta de que era un trabajo extenso y laborioso. Sin embargo, los resultados nos muestran el potencial que tiene el desarrollo de esta aplicación.

Inicialmente, el trabajo se concretó a la búsqueda de información sobre paquetería existente; fue en este momento cuando reafirmamos la necesidad de crear un sistema que nos permitiera definir rápida y eficazmente una simulación de procesos de manufactura. El programa tenía que cumplir con los elementos básicos; pero a la vez, necesitaba tener la posibilidad de crecer ampliamente; es decir, ser extensible.

Gracias al progreso en plataformas de desarrollo, pudimos realizar un programa amigable en *Windows*; ya que dicha plataforma nos permitiría emigrar fácilmente, en un futuro no muy lejano, a otras arquitecturas de *hardware*.

La metodología desarrollada, junto con el metalenguaje definido, nos dan los primeros pasos de comunicación con múltiples lenguajes de simulación actualmente en el mercado. Es lógico pensar, que en esta primera versión del paquete quedan algunas partes por desarrollar; pero todo esto, excede a los objetivos del trabajo. Algunas ideas a desarrollar en este proyecto podrían ser : OLE 2.0 (*Object Linking and Embedding*), activación de macros, definición de submodelos, convertidor de GPSS y SLAM, desarrollo de una versión de 32 *bits* y posteriormente una para *Windows* NT, etcétera. Sin embargo, estamos ofreciendo la posibilidad de que se siga investigando y desarrollando en esta rama. No es difícil imaginar que se podría proponer una versión mejorada como estándar en la comunicación entre aplicaciones del mismo tipo.

El desarrollo del programa no fue una tarea sencilla, debido a que era necesario conocer ampliamente la programación dentro de *Windows* (bastante distinta a la programación de sistemas en otras plataformas no dirigidas por medio de eventos); además, aunque *Turbo Pascal for Windows* es orientado a objetos, no contempla un soporte amplio a distintos elementos existentes en *AT&T C++ 3.0* o *Smalltalk*; lo que nos limitó un poco, en cuanto a diseño y codificación.

Una vez terminado el programa PSI (*Process Simulation Interface*), se sometió a pruebas con distintos usuarios para retroalimentar versiones futuras y corregir detalles existentes en la versión actual. Los resultados fueron halagadores; así como la comunicación con PASION fue un éxito.

En este momento, son tantos los paquetes de definición de procesos de manufactura y normalmente a un precio tan elevado, que nos complace el brindar una opción sencilla, barata y con mucho futuro a la industria en México; a fin de poder lograr el óptimo nivel en la productividad, mediante la simulación de sus operaciones y, por ende, en la economía de la fabricación. Lo anterior, es clave en un mundo tan competitivo donde los países luchan por ganar cada vez mejores segmentos de mercado, ofreciendo máxima calidad al menor precio en productos y/o servicios.

BIBLIOGRAFÍA

ABELLANAS M. y LODARES D., Análisis de Algoritmos y Teoría de Grafos, México, Macrobit, 1991.

BERGER Marc, (trad. LOZANO Jorge), Graficación por Computador con Pascal, EUA, Addison-Wesley Iberoamericana, 1991.

CLARK Jeffrey D., Windows Programmer's Guide to OLE/DDE, EUA, Sams, 1992.

DALE Nell y LILLY Susan, (trad. TROYA José), Pascal y Estructura de Datos, México, McGraw-Hill, 1988.

DOYLE Lawrence E. et al ,(trad. FOURNIER Julio), Materiales y Procesos de Manufactura para Ingenieros, México, Prentice Hall, 1988.

ENTSMINGER Gary, Turbo Pascal for Windows Bible, EUA, Sams, 1992, (2a ed).

FELSINGER Richard, Análisis y Diseño Orientado a Objetos, México, Technology Training, 1992.

FOWLER Martin, "OO Methods: A comparative overview", Object-Oriented Analysis and Design, EUA, volumen 6, número 4, 1993, julio-agosto.

KENDALL E. Kenneth et al, Análisis y diseño de sistemas, (trad. Héctor López Hernández), México, Prentice-Hall, 1991.

FICHMAN Robert et al, "Object - Oriented and Conventional Analysis and Design Methodologies", Computer IEEE, EUA, volumen 25, número 10, 1992, octubre.

(trad. MADISETTI Vijay), Modeling and Simulation on Microcomputers, EUA, SCS, 1990.

MURRAY Kathy, The Graphics Coach, EUA, New Riders Publishing, 1993.

(trad. NELSON Barry L. et al), 1991 Winter Simulation Conference Proceedings, EUA, Ohio State University, 1991.

(trad. NILSEN Ragnar), Tools For The Simulation Profession, EUA, SCS, 1989.

ObjectWindows Programming Guide, EUA, Borland, 1992.

OGATA Katsuhiko,(trad. PECINA José), Dinámica de Sistemas, México, Prentice Hall, 1987.

_____, (trad. FRANKEL Bartolomé), Ingeniería de Control Moderna, México, Prentice Hall,1980.

PETZOLD Charles, Programming Windows, EUA, Microsoft Press, 1991.

RACZYNSKI Stanislaw, "Graphical description and a program generator for queuing models", Simulation SCS, EUA, volumen 57 ,número 3, 1990, septiembre.

_____, PASION User's Manual, México, Universidad Panamericana, 1993.

RINE David et al, "Object - Oriented Computing", Computer IEEE, EUA, volumen 25, número 10, 1992, octubre.

(trad. SPENCER Susan et al), System Dynamics, EUA, SCS, 1989.

SWAM Tom, "Algorithm Alley, inside BMP's", Dr. Dobb's Journal, EUA, Número 204, 1993, septiembre.

TENENBAUM Aarón y AUGENSTEIN Moshe,(trad. LOPEZ Luis), Estructura de Datos en Pascal, México, Prentice Hall, 1983.

Turbo Pascal for Windows Windows Reference Guide, EUA, Borland, 1991.

VARHOL Peter, "Extending a visual language for Simulation", Dr. Dobb's Journal, EUA, Número 201, 1993, junio.

VOSS Greg, Object Oriented Programming, EUA, Osborne McGraw-Hill, 1991.

APÉNDICE A
MANUAL DEL USUARIO

PROCESS SIMULATION INTERFACE

Manual del Usuario

Realizado por Omar Aguirre Suárez y Gerardo Bárcena Ruiz
Universidad Panamericana, 1993

CONTENIDO

I. INTRODUCCIÓN

1. Introducción a la primera versión

PSI (*Process Simulation Interface*) está diseñado para usuarios que deseen definir gráficamente simulaciones discretas en el ambiente *Windows*, para posteriormente convertir dicha definición a algún lenguaje de simulación.

Es decir, PSI es un paquete para definir simulaciones de eventos discretos; y no realiza directamente algún tipo de simulación, sino que se sirve a paquetes ya diseñados para este fin.

Actualmente PSI se comunica con PASION (*PAScal SImulatiON*) para el desarrollo de la simulación (posteriormente se contará con el convertidor para GPSS).

Si usted tiene algún tipo de problema en la ejecución del programa (actualmente en la primera versión), le pedimos que por favor apunte la dirección en donde ocurrió dicho error y se comunique con las siguientes personas:

Omar Aguirre Suárez
Tel. 6-74-30-50

Gerardo Bárcena Ruiz
Tel. 6-72-82-77

En los siguientes capítulos mostramos los principales elementos del programa y algunos ejemplos.

2. Instalación

a. Instalación

El programa *Process Simulation Interface* (PSI) Ver 1.0 para *Windows* requiere para instalarse (una vez ingresado el disco de instalación en la unidad A) de la siguiente instrucción escrita en el administrador de programas, menú de archivos, apartado de ejecución:

A:INSTALL (ENTER)

El instalador se encargará de generar los directorios necesarios para la ejecución del programa.
Si usted desea parar la instalación deberá oprimir Alt+F4.
Es posible cambiar de aplicación mientras se instalal programa, esto se realiza mediante Alt+*Tab*.

b. Requerimientos

Los requerimientos de PSI son:

Disco duro con espacio de 2.5 *Megabytes* libres.
Microsoft Windows Ver 3.0 ó Ver 3.1 .
Display VGA.
Ratón.
PASION.

NOTA: Es importante mencionar que el directorio de PASION debe estar en la ruta de directorios (*PATH*) de sistema operativo para la correcta ejecución del programa.

3. Introducción al ambiente

El ambiente de PSI, por haberse realizado en *Windows*, maneja los mismos estándares.
Existen tres tipos de archivos en PSI: los archivos que contienen el código que genera una salida gráfica (formato ASCII, con la extensión .COD), los archivos que contienen la definición gráfica del modelo (formato binario, con la extensión .PSI), y los archivos de generación gráfica inmediata (formato ASCII, con la extensión .LNK). Todo esto se explicará posteriormente.
PSI fue realizado con la posibilidad de manejar varios documentos simultáneamente. Para cada proyecto (se entiende por proyecto al conjunto de archivos que definen un proceso) existen dos tipos de ventanas: un editor de texto (ASCII), donde se realizará la definición del

modelo mediante un metalenguaje (archivos con extensión .COD); y el editor gráfico del modelo, donde se define el proceso mediante imágenes que se despliegan en pantalla.

Para ingresar al programa, sólo seleccione el icono y dé doble clic con el ratón. En la pantalla aparecerá un menú con dos opciones: archivos y ayuda; elija la opción de Nuevo.

Observe que en este momento se crearon dos ventanas (la de edición de código y la de edición gráfica). En la parte inferior contará con una línea de seguimiento que le permitirá ver información de ayuda para la edición.

Si usted abre un archivo existente dentro de una ventana previamente abierta, automáticamente se inicializará la ventana y se cargará la nueva información. Para abrir un proyecto, deseando que las demás ventanas queden abiertas, tendrá que abrir un nuevo proyecto y en esa ventana cargar el archivo ya existente.

II. ELEMENTOS

1) Generadores

Estos bloques se encargan de generar partes, transacciones, clientes, pacientes, etcétera. Los generadores tienen como características principales los siguientes parámetros:

- Tiempo inicial. Se refiere al tiempo para la generación de la primera entidad.
- Distribución. La distribución aplicada para la generación de partes. Dentro de estas funciones se cuentan con todas las definibles en el lenguaje de Pascal y PASION:

NEGEXP(a)
NORM(a,s)
RANDOM
ERLANG(k,s)
SAMPLE(m,x), ... para mayor información consulte el manual del usuario de PASION.

Número máximo de entidades.

Entidades por grupo. Se refiere al número de entidades que forman un grupo; es decir, el generador creará grupos de n entidades.

Sólo genera un flujo de salida.

2) Colas

Este tipo de bloque corresponde a la representación de una cola en la línea de un proceso. Contiene los siguientes parámetros:

- Longitud inicial. Se refiere al número de entidades existentes en el tiempo cero de la simulación.
- Costo por unidad.
- Máximo número de entidades.
- Tipo de la cola. Puede tener los valores *LIFO, FIFO, RAND.*

Este bloque tiene una salida y una entrada.

3) Servidores

Un bloque servidor, es aquel que realiza algún tipo de operación sobre la entidad en proceso. Contiene los siguientes parámetros:

- Tiempo adicional de salida.
- Distribución.
- Costo por unidad de tiempo.
- Costo por operario ó recursos empleados.

El número de flujos que entran en el servidor puede ser múltiple (en el caso de PASION, actualmente existe una restricción de 10; los demás serán ignorados por el convertidor), tomando uno de los siguientes comportamientos de selección: por probabilidad, por prioridad, o por alguna regla definida por el usuario (para mayor información sobre este aspecto consulte el manual del usuario de PASION).

Este bloque tiene entradas múltiples y una salida.

4) Ensambladores

Un bloque ensamblador es aquel que requiere de cierto número de entidades de cada línea para generar una entidad nueva. Tiene los siguientes atributos:

- Tiempo adicional de salida.
- Distribución.
- Costo por unidad de tiempo.
- Costo por operario o recursos empleados.

Además, es necesario definir el número de entidades por cada línea de entrada.

Este bloque tiene entradas múltiples y una salida.

5) Divisores de flujo

Un bloque divisor se encarga de elegir un camino para la entidad en cuestión. Maneja, al igual que el servidor, tres tipos de regla de selección: por probabilidad, por prioridad, o por una regla de decisión definida por el usuario.

Este bloque tiene una entrada y múltiples salidas.

6) Terminadores

Un bloque terminador finalizal flujo del proceso. Tiene los siguientes atributos:

- Número de entidades a restar.
- Número máximo de partes terminadas.

Este bloque tiene únicamente una entrada.

7) Recursos

Los recursos son aquellos elementos necesarios paral funcionamiento de cualquier bloque. Las características de un recurso se definen por medio de tipo y de atributos.

III. COMANDOS

1. Metalenguaje

a. Parámetros generales

Dimensions *ResHor, ResVer*

Esto define las dimensiones del área del usuario con resolución en *pixels*, siendo el máximo valor ingresable 32546; en cualquiera de las direcciones.
El área de edición tiene origen en la parte superior izquierda; se incrementa a la derecha en el eje X y hacia abajo en el eje Y.

b. Atributos adicionales

```
Parts
  Type
    NameType1= PascalTypeDefinition
    NameType2= PascalTypeDefinition
    ...
  Attributes
    Atr1: NameType1;
    Atr2: PascalTypeDefinition;
    ...
End;
```

c. Generador

Create Generator Named *IdNum*
  At *PosX, PosY*
  Bitmap= *PathBmpFile*
  Distribution= *PasionFunction*
  Init_Time= *PascalExpression*
  Gen_Groups= *PascalExpression*
  Max_Number= *PascalExpression*
  Linked with *IdNxt*
  Use resources: *IdRes*
End;

d. Colas (FIFO, LIFO, RAND)

Create Queue Named *IdNum*
  At *PosX, PosY*
  Bitmap= *PathBmpFile*
  Type= *[Fifo, Lifo, Rand]*
  Init_Len= *PascalExpression*
  Max_Number= *PascalExpression*
  Cost_Per_Unit= *RealNumber*
  Use resources: *IdRes*
End;

e. Servidor

Create Server Named *IdNum*
  At *PosX, PosY*
  Bitmap= *PathBmpFile*
  Distribution= *PascalExpression*
  Add_Delay= *PascalExpression*
  Cost_Per_Time= *RealNumber*
  Cost_Per_Oper= *RealNumber*
  Linked with *IdNxt*
  Being linked by *[ProbabilityRule, PriorityRule, UserRule= DefOfRule]* with:
  (IdPrv1, WPrv1) , (IdPrv2, WPrv2) , ...
  Use resources: *IdRes*
End;

f. Ensamblado

Create Assembler Named *IdNum*
  At *PosX, PosY*
  Bitmap= *PathBmpFile*
  Distribution= *PascalExpression*

Add_Delay= *PascalExpression*
Cost_Per_Time= *RealNumber*
Cost_Per_Oper= *RealNumber*
Linked with *IdNxt*
Being linked with:
(*IdPrv1*, *NoParts1ForAssem*) , (*IdPrv2*, *NoParts2ForAssem*) , ...
Use resources: *IdRes*
End;

g. División del flujo

Create Split Named *IdNum*
At *PosX*, *PosY*
Bitmap= *PathBmpFile*
Linked by [*ProbabilityRule*, *PriorityRule*, *UserRule= DefOfRule*] with:
(*IdNxt1*, *WNxt1*) , (*IdNxt2*, *WNxt2*) , ...
Use resources: *IdRes*
End;

h. Terminador

Create Terminator Named *IdNum*
At *PosX*, *PosY*
Bitmap= *PathBmpFile*
No_Parts_Dec= *RealNumber*
No_Max_Parts= *PascalExpression*
Use resources: *IdRes*
End;

i. Recursos

Create Resource Named *IdNum*
At *PosX*, *PosY*
Bitmap= *PathBmpFile*
Type
NameType1= PascalTypeDefinition
NameType2= PascalTypeDefinition
...
Attributes
Atr1: NameType1;
Atr2: PascalTypeDefinition;
...
End;

j. Referencias de Cursivas

ResXXX:

Define el tamaño en la dirección especificada, debe ser menor a 32546.

NameTypeX:

Establece el nombre del tipo definido por el usuario.

PascalTypeDefinition:

Define el tipo de Pascal utilizado.

AtrX:

Determinal nombre del atributo (variable) que se está definiendo.

IdXXX:

Determinal número que identifica al bloque en su creación.

Determinal número del bloque que apunta al actual.

Determinal número del bloque que es apuntado por el actual.

Determinal número de recurso que se utiliza.

PosX,Y:

Determina la posición (según el caso: X, Y), en la que se encuentral bloque.

PathBmpFile:

Dermina la ruta de acceso donde se encuentral *BitMap* que representa al bloque.

PascalExpression:

Define una función o expresión de PASION o Pascal, según seal caso. Recurra a los manuales tanto de PASION como de Pascal para determinar las expresiones válidas.

Fifo,Lifo,Rand:

Define el tipo de cola que se usa:

FIFO: First In First Out, una cola normal.

LIFO: Last In First Out, una pila normal.

RAND: una distribución aleatoria.

RealNumber:

Establece un número real; para Pascal el rango es el siguiente: 2.9E-39 .. 1.7E38, negativos también, con 11 o 12 dígitos y 6 bytes de tamaño.

XXXRule:

Define el tipo de regla de asociación que se usará en el momento de simulación.

ProbabilityRule : Regla de probabilidad; verifique que la suma siempre sea igual a la unidad.

PriorityRule : Regla de prioridad; verifique que cada rama presenta prioridades distintas.

UserRule= DefOfRule : Regla definida por el usuario; recuerde que en su programa, en PASION, debe existir la definición de la rutina que aquí llame.

WXXX:

Define en sí la cantidad usada en la regla de asociación.

NoPartsXForAssem:

Determinal número de partes que llegan por esa rama al ensamblador.

2. Palabras reservadas

Add_Delay=	Establece un retardo
At	Establece la posición del bloque
Attributes	Establece las características de la parte
Being linked by ... with: ...	Establece los bloques que lo apuntan
Being linked with:	Establece los bloques que apuntan a un ensamblador
Bitmap=	Establece la imagen que representa al bloque
Cost_Per_Oper=	Costo de la operación
Cost_Per_Time=	Costo por tiempo empleado
Cost_Per_Unit=	Costo por unidad
Create ... Named ...	Crea un bloque de un tipo específico con su identificador
Distribution=	Establece la manera en la que se comportal generador
End;	Finaliza bloques de creación
End_Processes	Termina la definición de bloques en archivos LNK
End_Links	Termina la definición de uniones en archivos LNK
Gen_Groups=	Establece la manera de generación de grupos
Init_Len=	Establece la longitud inicial de la cola
Init_Time=	Establece el tiempo inicial de espera
Links	Empieza la definición de uniones en archivos LNK
Linked by ... with: ...	Determina los bloques que apunta (para salidas múltiples)
Linked with	Determinal bloque al que apunta
Max_Number	Determinal número máximo de elementos
No_Max_Parts	Determinal número máximo de partes
No_Parts_Dec	Determinal número de partes a ser restadas
Parts	Empieza la definición de las características de la parte
Processes	Empieza la definición de bloques en archivos LNK
Type	Empieza la definición de tipos
Type=	Determinal tipo de cola
Use resource:	Determinal identificador del recurso

3. Archivos LNK

Un archivo con extensión LNK tiene por objetivo servir como una definición rápida de los elementos que componen un modelo. Este archivo tiene formato ASCII, con lo que se facilita su creación en cualquier editor de archivos de texto, como podrían ser: Notas de *Windows*, *Edit* del sistema operativo y, por supuesto, el editor de código de PSI.
Para comprender la codificación de este archivo, es necesario observar la siguiente tabla:

Descripción del tipo de bloque	Número de identificación	Nombre en código
Generador	103	*Generator*
Cola	105	*Queue*
Servidor	106	*Server,*
Ensamblador	107	*Assembler*
Suma de flujos	108	*SumFlow*
Divisor de flujos	109	*Split*
Procesado por lotes	110	*Batch*
Terminador	111	*Terminator*
Transportador	113	*Transport*
Bandas	114	*Convey*
AGV's	115	*AGV*
Transportador motorizado	116	*Motor*
Grúas	117	*Crane*

Aquí se muestran los identificadores y el nombre utilizado en código para cada uno de los tipos de bloque soportados en PSI.
En este momento, es posible observar la sintaxis de un archivo LNK:

PROCESSES
Identificador del bloque,[Nombre en código/Número de identificación],[Camino del bitmap/NOBITMAP]
...
END_PROCESSES
LINKS
Identificador del bloque 1, Identificador del bloque 2
...
END_LINKS

IV. PROCEDIMIENTOS

Refiérase al capítulo MENÚS para determinar el comportamiento de estos comandos.

COMPILAR

EDITAR
- Copiar
- Copiar imagen
- Selecciona todo
- Deselecciona todo
- Edita los bloques seleccionados...
- Atributos de la parte...

OPCIONES
- Dims. de edición...
- Mover grafo ...
- Acomoda grafo
- Alinear bloques
- Verificar flujo
- Reenumerar
- Completa los flujos
- Esconder identificadores

CÓDIGO
- Convierte a ...
- Genera código ...

V. EJEMPLOS

1. De archivos LNK

Ahora un pequeño ejemplo: Supongamos que queremos el siguiente flujo, 1 generador, 1 cola, 1 servidor y 1 terminador, enlazados secuencialmente; entonces el archivo (llamémoslo Prueba.LNK) tendría la siguiente forma:

```
PROCESSES
1,Generator
2,Queue
3,Server
4,Terminator
END_PROCESSES
LINKS
1,2
2,3
3,4
END_LINKS
```

Si quisiéramos poner una serie de imágenes que tenemos en cada uno de los bloques, entonces el archivo tomaría la siguiente forma:

```
PROCESSES
1,Generator,C:\PSI\IMAGES\Gen.BMP
2,105,C:\PSI\IMAGES\Queue.BMP
3,Server,C:\PSI\IMAGES\Server.BMP
4,Terminator,C:\PSI\IMAGES\Term.BMP
```

```
END_PROCESSES
LINKS
1,2
2,3
3,4
END_LINKS
```

Nótese que también es válido colocar el número de identificación del tipo de bloque, como se hizo en el bloque número dos.

Ahora lo único a realizar es:

a) Cargar el programa de PSI.
b) Abrir un proyecto nuevo.
c) Abrir el archivo Prueba.LNK
d) En este momento se acomodan los bloques automáticamente y los despliega en la pantalla gráfica.

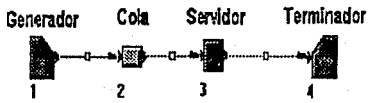
2. De archivos COD

a. Ejemplo de una línea sencilla

La definición del problema es el siguiente:

En un sistema de manufactura, elementos provenientes de una fundición son pasados a una taladradora, donde a cada pieza se le hará un agujero. El tiempo de llegada de las piezas de fundición es una distribución uniforme en un intervalo de 15.0±4.5 minutos. El tiempo requerido por el taladro es de 13.5±3.0 minutos, uniformemente distribuido. Las piezas de fundición son pasadas al taladro como: primera que llega, primera que se taladra.

Este problema se puede modelar con una línea conformada por los siguientes elementos:



Las características de cada elemento serán:

1.*Generador*. Tiempo inicial = 0, distribución = NORM(15.0,4.5), máximo número de entidades = 0 (no tiene límite), y número de entidades por grupo= 1.
2.*Cola*. Longitud inicial = 0, costo por unidad = 1, tipo = *FIFO*.

3.Servidor. Tiempo adicional de salida = 0, distribución = NORM(13.5.3.0), costo por unidad de tiempo = 1, costo por operación = 1.
4. Terminador. Número de partes a restar = 0, número máximo de partes = 0 (ilimitado).

Existen diversos métodos para definir este simple flujo, pero nosotros sólo veremos dos:

El primer método consiste en definir los bloques de la siguiente forma:

a) Coloque el ratón en la barra de utilería donde se encuentra la imagen que representa un generador. Presione el botón izquierdo.

b) Mueval ratón al área gráfica y decida dónde ponerlo.

c) Presione nuevamente el botón izquierdo (si no le gustó la posición del bloque, puede seleccionarla mediante el ratón, oprimiendo nuevamente el botón izquierdo dentro del mismo, moviéndolo a la posición deseada soltando el botón). Puede cambiar el tamaño de la imagen que representa este generador colocando el ratón en la esquina inferior derecha del rectángulo que delimital bloque; presione el botón izquierdo cuando el cursor cambie de figura y mueval ratón a su nuevo tamaño; finalmente, suelte el botón.

d) En este momento, el primer elemento se encuentra en la pantalla; para los demás elementos se repiten los pasos anteriores; pero seleccionando el elemento correspondiente en la barra de utilería .

e) A continuación se definirá el flujo del proceso. Hay que seleccionar el bloque número 1 y el bloque número 2. Posteriormente, es necesario oprimir el segundo botón de la barra de utilería (también se puede realizar la conexión oprimiendo la letra C en el teclado). Automáticamente generará el enlace entre dichos bloques. Para las siguientes conexiones aprovecharemos el seleccionado múltiple y conectaremos todos los bloques seleccionados. Esto se realiza tomando la herramienta de selección, oprimiendo el primer punto del cuadro de selección por encima y hacia la izquierda del bloque número 2, moviendo el ratón hasta que quede por debajo y hacia la derecha del bloque 4; y soltando el botón del ratón. De esta forma, estarán seleccionados los bloques 2,3,4. Oprimamos la letra C en el teclado y observemos como se generan las uniones correspondientes 2-3, 3-4. Las conexiones fueron realizadas de acuerdo a la creación de los bloques (primero se creó el bloque número 2 y en seguida se creó el bloque número 3; por lo que la unión se generó de 2 hacia 3). Para deseleccionar los bloques podemos oprimir el botón derecho del ratón.

f) Para ingresar cada uno de los parámetros del ejemplo, es necesario que se seleccionen todos los bloques (ya sea mediante la herramienta de selección, manualmente o mediante la letra S).

g) Para ingresar cada una de las características de los bloques, nos vamos al menú de edición y seleccionamos el apartado de Editar los bloques seleccionados.

h) Hemos finalizado la declaración de nuestra línea de manufactura.

i) Podemos ahora, grabar este archivo mediante la selección del apartado de Guardar dentro de Archivos, donde le pondremos un nombre al archivo (con extensión .PSI) y oprimiremos el botón de Aceptar.

j) Si queremos observar el código que representa esta imagen, podemos seleccionar la opción Genera código del menú de Código. Se generará el lenguaje correspondiente a la imagen en la ventana editora de código. Lo puede observar, cambiando de ventana mediante Ctrl+Tab o escogiéndola en el menú de ventanas. Aquí mostramos el conjunto de instrucciones generadas:

```
DIMENSIONS 1000,1000

CREATE Generator NAMED 1
  AT 67,34
  DISTRIBUTION=NORM(15,4.5)
  INIT_TIME=0
  GEN_GROUPS=1
  MAX_NUMBER=0
  LINKED WITH 2
END;
CREATE Queue NAMED 2
  AT 131,34
  TYPE=FIFO
  COST_PER_UNIT=1.0
  INIT_LEN=0
  MAX_NUMBER=0
  LINKED WITH 3
END;
CREATE Server NAMED 3
  AT 195,34
  DISTRIBUTION=NORM(13.5,3.0)
  ADD_DELAY=0
  COST_PER_TIME=1.0
  COST_PER_OPER=1.0
  LINKED WITH 4
  BEING LINKED BY Priority
  WITH:
  (2,0.00)
END;
CREATE Terminator NAMED 4
  AT 259,34
  NO_PARTS_DEC=0
  NO_MAX_PARTS=0
END;
```

k) Este archivo también se puede guardar similarmente (generalmente con extensión .COD) mediante el menú de archivos y guardar.

l) Suponga que desea alinear todos los bloques hacia arriba. Esto lo puede realizar cambiándose de ventana, seleccionando todos los bloques, oprimiendo el botón de alineación en la barra de utilería y seleccionando en el diálogo la opción Arriba.

m) Si queremos imprimir la imagen podemos especificar la impresora en el menú de archivos, y posteriormente escoger alguno de los tres formatos:

- Ajustar el tamaño de hoja a una hoja de impresión. Para esta opción, podríamos reducir el tamaño de la imagen mediante el menú de opciones y tamaño, cambiándolo a 1000 por 1000.
- Imprimir sólo lo que está visible. En este momento se imprime escalado a la hoja de impresión sólo lo que se encuentra visible en la pantalla de edición gráfica.

- Imprimir con la resolución de la impresora. Esta opción toma 1 *pixel* = 1 punto en la impresora.

n) Si usted desea cambiar la imagen que representa un generador, primero deseleccione todos los bloques, seleccione el bloque del generador, oprima Enter, oprimal botón de BMP y escoja alguna imagen que le guste (recuerde que tiene una biblioteca de imágenes en el directorio de IMAGES). En este diálogo usted puede ver el BMP escalado en una ventana que aparece en la parte derecha (si usted presional botón izquierdo encima de la imagen, ésta cambiará al tamaño original).

o) En algún proceso, los bloques y las líneas de unión se cruzan, por lo que es necesario editar la línea para que esto no ocurra. La edición de una línea consiste en seleccionar el cuadro pequeño que se encuentra en la mitad de la línea y mover los puntos que representan la línea (estos puntos definen una curva Bezier de 3er grado). Para seleccionar el color y grosor de las líneas, se vuelve a oprimir el cuadro central, apareciendo un diálogo donde se capturan estos datos. Recuerde que si usted mueve alguno de los bloques, la línea se recalculará; por lo que se recomienda, primero se muevan todos los bloques a los lugares deseados y posteriormente se editen las líneas.

p) Una vez realizado todo esto, podemos verificar el modelo en el menú de opciones y tratar de simular nuestro ejemplo en PASION (para lo que es necesario que PASION exista en el disco duro y su directorio sea \PASION o se encuentre en el PATH de sistema operativo).

q) Para simularlo en PASION se elegirá el menú de código en el apartado de Convertir a..., oprimiendo el botón de Aceptar. En este momento se abrirá una ventana de DOS, donde correrá la simulación del archivo (para mayor información sobre los archivos generados es necesario consultar el manual de PASION, así como el manual técnico de PSI).

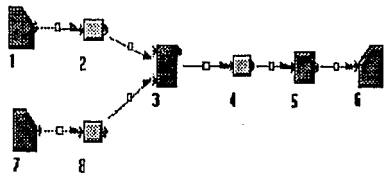
r) Al terminar nuestra sesión podemos salir del programa oprimiendo Alt+F4.

El segundo método consiste en una forma más rápida de creación:

a) Ponga un bloque generador en la pantalla de edición.
b) Seleccione la opción de completar flujos en el menú de opciones. Automáticamente el generador al no tener un bloque siguiente, crea un bloque de tipo cola, éste a su vez genera un servidor y, finalmente, éste a su vez genera un terminador. Esta opción puede ahorrarnos la creación de múltiples objetos en la definición de modelos más grandes.
c) Edite las características de cada uno de los bloques.
d) Continúe realizando las opciones del otro método, en caso de que así lo desee.

b. Ejemplo de un proceso de ensamblado

Para este ejemplo tenemos el siguiente flujo de proceso:



Todo se realiza similar al ejemplo anterior y el código generado da como resultado:

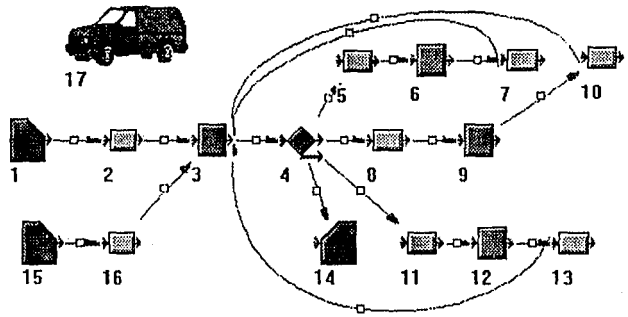
```
;Se pueden colocar
;comentarios diversos entre
;bloque y bloque mediante una
;coma.
DIMENSIONS 30000,30000

CREATE Generator NAMED 1
  AT 99,53
  DISTRIBUTION=NEGEXP(1.0)
  INIT_TIME=0
  GEN_GROUPS=1
  MAX_NUMBER=0
  LINKED WITH 2
END;
CREATE Queue NAMED 2
  AT 177,55
  TYPE=LIFO
  COST_PER_UNIT=1.0
  INIT_LEN=0
  MAX_NUMBER=0
  LINKED WITH 3
END;
; El ensamblador define el
;número de partes necesarias
;en la línea Being linked with
;necesitando una entidad del
;bloque 2, y tres entidades del
;bloque 8.
CREATE Assembler NAMED 3
  AT 260,82
  DISTRIBUTION=NEGEXP(3.0)
  ADD_DELAY=0
  COST_PER_TIME=1.0
  COST_PER_OPER=1.0
  LINKED WITH 4
  BEING LINKED WITH:
  (2,1),(8,3)
END;
CREATE Queue NAMED 4
  AT 342,82
  TYPE=LIFO
  COST_PER_UNIT=1.0
  INIT_LEN=0
  MAX_NUMBER=0
  LINKED WITH 5
END;
CREATE Server NAMED 5
  AT 413,83
  DISTRIBUTION=NEGEXP(3.0)
  ADD_DELAY=0
  COST_PER_TIME=1.0
  COST_PER_OPER=1.0
  LINKED WITH 6
  BEING LINKED BY Priority
  WITH:
  (4,0.00)
END;
CREATE Terminator NAMED 6
  AT 485,82
  NO_PARTS_DEC=0
  NO_MAX_PARTS=0
END;
CREATE Generator NAMED 7
  AT 103,130
  DISTRIBUTION=NEGEXP(1.0)
  INIT_TIME=0
  GEN_GROUPS=1
  MAX_NUMBER=0
  LINKED WITH 8
END;
CREATE Queue NAMED 8
  AT 176,131
  TYPE=LIFO
  COST_PER_UNIT=1.0
  INIT_LEN=0
  MAX_NUMBER=0
  LINKED WITH 3
END;
```

c. Ejemplo de un proceso de manufactura

En este ejemplo contemplamos el siguiente flujo:

En la siguiente figura el bloque número 17 representa un recurso utilizado por el servidor número tres. El bloque número 4 representa un divisor de flujos por probabilidad. Además, necesitamos ciertas características de las partes generadas.

El código generado por esta representación es el siguiente:




```
DIMENSIONS 1000,1000

; Observe cómo se definen
;tipos y atributos de la parte
;generada
PARTS
  TYPES
    TypeAttr1=Integer;
  ATRIBUTES
    AttrPart1:TypeAttr1
END;

CREATE Generator NAMED 1
  AT 10,119
  DISTRIBUTION=NEGEXP(1.0)
  INIT_TIME=0
  GEN_GROUPS=1
  MAX_NUMBER=0
  LINKED WITH 2
END;

CREATE Queue NAMED 2
  AT 76,119
  TYPE=LIFO
  COST_PER_UNIT=1.0
  INIT_LEN=0
  MAX_NUMBER=0
  LINKED WITH 3
END;
; Observe como se ingresan
;las prioridades por línea y la
;utilización del recurso 17
CREATE Server NAMED 3
  AT 142,119
  DISTRIBUTION=NEGEXP(3.0)
  ADD_DELAY=0
  COST_PER_TIME=1.0
  COST_PER_OPER=1.0
  LINKED WITH 4
  BEING LINKED BY Priority
  WITH:
    (2,1),(7,2),(10,3),(13,5),(16,4)
  USE RESOURCES: 17
END;
; Este es un bloque divisor por
;probabilidad
CREATE Split NAMED 4
  AT 208,119
  LINK BY Probability WITH:
  (5,0.30),(8,0.20),(11,0.10),(14,0.40)
END;
CREATE Queue NAMED 5
  AT 250,61
  TYPE=LIFO
  COST_PER_UNIT=1.0
  INIT_LEN=0
  MAX_NUMBER=0
  LINKED WITH 6
END;
CREATE Server NAMED 6
  AT 304,60
  DISTRIBUTION=NEGEXP(3.0)
```

```
  ADD_DELAY=0
  COST_PER_TIME=1.0
  COST_PER_OPER=1.0
  LINKED WITH 7
  BEING LINKED BY Priority
  WITH:
  (5,0.00)
END;
CREATE Queue NAMED 7
  AT 370,60
  TYPE=LIFO
  COST_PER_UNIT=1.0
  INIT_LEN=0
  MAX_NUMBER=0
  LINKED WITH 3
END;
CREATE Queue NAMED 8
  AT 272,119
  TYPE=LIFO
  COST_PER_UNIT=1.0
  INIT_LEN=0
  MAX_NUMBER=0
  LINKED WITH 9
END;
CREATE Server NAMED 9
  AT 338,119
  DISTRIBUTION=NEGEXP(3.0)
  ADD_DELAY=0
  COST_PER_TIME=1.0
  COST_PER_OPER=1.0
  LINKED WITH 10
  BEING LINKED BY Priority
  WITH:
  (8,0.00)
END;
CREATE Queue NAMED 10
  AT 428,58
  TYPE=LIFO
  COST_PER_UNIT=1.0
  INIT_LEN=0
  MAX_NUMBER=0
  LINKED WITH 3
END;
CREATE Queue NAMED 11
  AT 297,194
  TYPE=LIFO
  COST_PER_UNIT=1.0
  INIT_LEN=0
  MAX_NUMBER=0
  LINKED WITH 12
END;
CREATE Server NAMED 12
  AT 348,194
  DISTRIBUTION=NEGEXP(3.0)
  ADD_DELAY=0
  COST_PER_TIME=1.0
  COST_PER_OPER=1.0
  LINKED WITH 13
  BEING LINKED BY Priority
  WITH:
  (11,0.00)
END;
CREATE Queue NAMED 13
  AT 407,194
  TYPE=LIFO
  COST_PER_UNIT=1.0
  INIT_LEN=0
  MAX_NUMBER=0
  LINKED WITH 3
END;
CREATE Terminator NAMED 14
  AT 233,194
  NO_PARTS_DEC=0
  NO_MAX_PARTS=0
END;
CREATE Generator NAMED 15
  AT 19,194
  DISTRIBUTION=NEGEXP(1.0)
  INIT_TIME=0
  GEN_GROUPS=1
  MAX_NUMBER=0
  LINKED WITH 16
END;
CREATE Queue NAMED 16
  AT 75,194
  TYPE=LIFO
  COST_PER_UNIT=1.0
  INIT_LEN=0
  MAX_NUMBER=0
  LINKED WITH 3
END;
; Este tipo de bloque aún no
;se encuentra definido por
;QMG de pasion.
CREATE Resource NAMED 17
  AT 51,35
  BITMAP=
  c:\pe\images\wagon1.bmp
  TYPES
    TypeRes1=String;
    TypeRes2=Char;
  ATRIBUTES
    AttrRes1:TypeRes1
    AttrRes2:TypeRes2
END;
```

VI. MENÚS

1. Menú ventana inicial

Archivos
- Nuevo
- Abrir...
- Especificar impresora ...
- Salida

Ayuda
- Índice
- Teclado
- Comandos
- Procedimientos
- Utilizando Ayuda
- Acerca...

2. Menú ventana de edición de código

Archivos
- Nuevo
- Abrir...
- Guardar
- Guardar como...
- Imprimir ...

Especificar impresora ...
Salida

Editar
Deshacer
Cortar
Copiar
Insertar
Borrar
Borrar todo

Buscar
Encontrar...
Reemplazar...
Siguiente

Compilar

Ventanas
Mosaico
Cascada
Arreglar Iconos
Cerrar todo

Ayuda
Índice
Teclado
Comandos
Procedimientos
Utilizando Ayuda
Acerca...

3. Menú ventana de edición de gráficos

Archivos
Nuevo
Abrir...
Guardar
Guardar como...
Imprimir ...
Especificar impresora ...
Salida

Editar
- Cortar
- Copiar
- Copiar imagen
- Pegar
- Borrar
- Borrar todo
- Selecciona todo
- Deselecciona todo
- Edita los bloques seleccionados...
- Atributos de la parte...

Opciones
- Dims. de edición...
- Mover grafo ...
- Acomoda grafo
- Alinear bloques
- Verificar flujo
- Reenumerar
- Completa los flujos
- Esconder identificadores

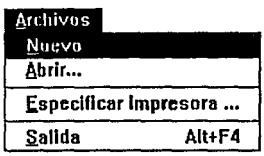
Código
- Convierte a ...
- Genera código ...

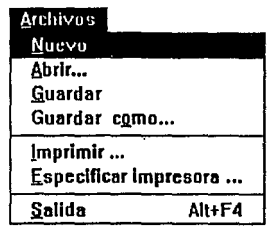
Ventanas
- Mosaico
- Cascada
- Arreglar Iconos
- Cerrar todo

Ayuda
- Índice
- Teclado
- Comandos
- Procedimientos
- Utilizando Ayuda
- Acerca...

4. Submenús por orden alfabético

a. Archivo





1) Abrir...
El menú de la izquierda corresponde a la ventana inicial, y el de la derecha a cualesquiera de las ventanas de edición.
Se encarga de cargar en la ventana activa el archivo que se desea editar (normalmente, se editan los archivos de programa; es decir, con extensión .COD, o los de creación rápida con extensión .LNK; aunque se puede utilizar como editor de cualquier archivo con formato ASCII).
Si se tienen un par de ventanas sin nombre (es decir, que se crearon por medio de ARCHIVO | Nuevo), y se ejecuta este comando, éstas tomarán la información del archivo que se está cargando.

2) Especificar impresora...
El menú de la izquierda corresponde a la ventana inicial y el de la derecha a cualesquiera de las ventanas de edición.
Se especifica el tipo de impresora disponible para la impresión. En caso de no aparecer el tipo de impresora que usted maneja, es necesario ir al Panel de Control e instalar la impresora correspondiente (para mayor información ver el manual de instalación de impresoras en el manual de usuario para Windows).

3) Guardar
Esta opción sólo se encuentra en las ventanas de edición.
Graba en la unidad de disco los cambios realizados al archivo. Este comando asume que el archivo a guardar ya tiene nombre y ruta de grabación; en caso de no tenerlo, se ejecutará el comando ARCHIVO | Guardar Como.

4) Guardar como...
Esta opción sólo se encuentra en las ventanas de edición.
Si es la primera vez que se graba el archivo, se ejecuta este comando para asignar el nombre, así como la ruta de acceso o grabación; es decir, el lugar o subdirectorio donde se alojará el archivo.

Si el archivo ya tiene nombre lo graba con otro nombre; esto sirve para hacer respaldos si existen modificaciones.

5) Imprimir...
Esta opción sólo se encuentra en las ventanas de edición.
Para la ventana de código: imprime el archivo activo; se necesita escoger las páginas que serán impresas, así como el número de copias que se desean imprimir.
Para la ventana gráfica, tenemos las siguientes opciones:

Ajustar el tamaño de hoja a una hoja de impresión
Toda la hoja de trabajo será impresa en una hoja del impresor. Tenga cuidado, pues las dimensiones de la hoja determinarán el escalado del dibujo.
Imprimir sólo lo que está visible en la pantalla
Imprime sólo la imagen que se muestra en el área cliente de la ventana gráfica.
Imprimir con la resolución de la impresora
Si usted cuenta con una impresora de muy alta resolución, y su grafo es bastante grande, la impresión será de muy buena calidad; pues se tomará un punto de impresión (*dot*) por cada *pixel*.

6) Nuevo
El menú de la izquierda es de la ventana inicial, y el de la derecha de cualesquiera de las ventanas de edición.
Crea ventanas de edición vacías, sin nombre de archivo.
El nombre del archivo se asigna al grabarlo. (Nota: si es un archivo de código es conveniente que tenga extensión .COD)
Recuerde que si desea establecer el nombre, así como la ruta de acceso, debe ejecutar el comando ARCHIVO | Guardar Como.

7) Salida
El menú de la izquierda corresponde a la ventana inicial, y el de la derecha a cualesquiera de las ventanas de edición.
Salida del programa PSI. Si por alguna razón no se ha grabado la información modificada, PSI preguntará si se desea guardar la información; es responsabilidad del usuario contestar afirmativa o negativamente.

b. Buscar

1) Encontrar...
Esta opción sólo se encuentra en la ventana de edición de código.

Se utiliza para buscar palabras en el texto editado. Se establece(n) la(s) palabra(s) a buscar y el modo de búsqueda.

2) Reemplazar...
Esta opción sólo se encuentra en la ventana de edición de código.
Se utiliza para reemplazar palabras en el texto editado. Se establece(n) la(s) palabra(s) a reemplazar y el modo de búsqueda.

3) Siguiente
Esta opción sólo se encuentra en la ventana de edición de código.
Repite la acción de encontrar o reemplazar según sea el caso.

c. Compilar

Esta opción sólo se encuentra en la ventana de edición de código.

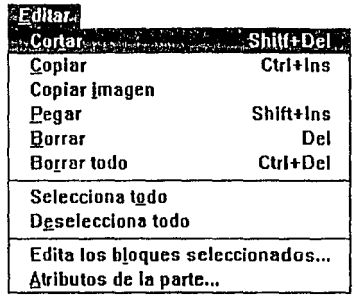
Se realiza la traducción del código realizado y se despliegan los resultados en la ventana gráfica correspondiente al proyecto. Si existía algo previamente en la ventana gráfica, se inicializa con el código traducido.

d. Código

1) Convierte a...
Esta opción sólo se encuentra en la ventana de edición gráfica.
Se encarga de convertir a alguno de los lenguajes instalados. Para esta opción, es necesario tener la librería de enlace dinámico (*DLL, Dynamic Link Library*) correspondiente al convertidor asociado.

2) Genera código...
Esta opción sólo se encuentra en la ventana de edición gráfica.
General código asociado a la representación gráfica en la ventana de edición de código.

e. Editar



1) Atributos de la parte...
Esta opción sólo se encuentra en la ventana de edición gráfica.
Edita los tipos y atributos de las entidades.

2) Borrar
Para la ventana de código: borra el carácter o la parte de texto seleccionados.
Para la ventana gráfica: borra los bloques seleccionados. Todas las uniones que tengan relación con el bloque a borrar, también desaparecerán. Si una unión es seleccionada se puede borrar por medio de este comando.

3) Borrar todo
Para la ventana de código: borra todo el contenido de la ventana de edición de código.
Para la ventana gráfica: borra todos los bloques de la definición, tanto los que tienen que ver con el flujo, como los bloques libres. Tenga cuidado con esta opción.

4) Copiar imagen
Esta opción sólo se encuentra en la ventana de edición gráfica.
Copia únicamente la imagen desplegada en pantalla al portapapeles con formato BMP.

5) Copiar
Para la ventana de código: copia el texto seleccionado al Portapapeles de *Windows*.
Para la ventana gráfica: copia los bloques seleccionados al portapapeles, junto con la imagen de la ventana actual con formato BMP.

6) Cortar
Para la ventana de código: borra el texto seleccionado.
Para la ventana gráfica: borra e inserta en el portapapeles los bloques seleccionados.

7) Deselecciona todo
Esta opción sólo se encuentra en la ventana de edición gráfica.

Deselecciona todos los bloques definidos. También se puede realizar esta operación oprimiendo el botón derecho del ratón o mediante la tecla *D*.

8) Deshacer
Esta opción sólo se encuentra en la ventana de edición de código.
Se encarga de deshacer el último cambio realizado al texto editado.

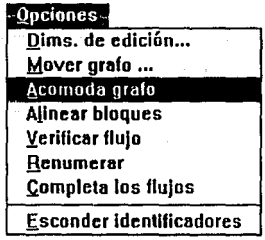
9) Edita los bloques seleccionados...
Esta opción sólo se encuentra en la ventana de edición gráfica.
Edita los atributos de los bloques seleccionados.
Esta opción también se puede ejecutar en la ventana de edición gráfica oprimiendo *ENTER*; pero sólo se podrán ingresar los atributos del primer bloque seleccionado.

10) Insertar
Esta opción sólo se encuentra en la ventana de edición de código.
Copia del Portapapeles de *Windows* al texto editado.

11) Pegar
Esta opción sólo se encuentra en la ventana de edición gráfica.
Pega los bloques que se encuentran en el portapapeles en la posición del ratón (ya sea en coordenadas absolutas o relativas).

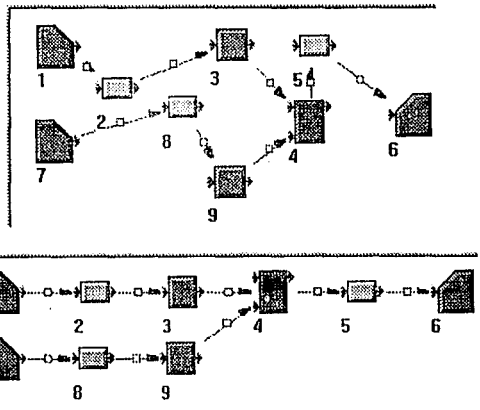
12) Selecciona todo
Esta opción sólo se encuentra en la ventana de edición gráfica.
Selecciona todos los bloques definidos. También se puede realizar esta operación mediante la tecla *S* en la pantalla de edición gráfica.

f. Opciones



1) Acomoda grafo
Esta opción sólo se encuentra en la ventana de edición gráfica.
Acomoda los bloques. Internamente este comando ejecuta a OPCIONES | Reenumerar.

Cuando en el momento de la edición, no se tiene el tiempo suficiente de acomodar los bloques, y se desea de manera rápida, es muy conveniente esta opción. A continuación se muestra un ejemplo de antes y después del acomodo de bloques.



2) Alinear bloques

Esta opción sólo se encuentra en la ventana de edición gráfica.

Alinea los bloques seleccionados en las siguientes direcciones: Derecha, Izquierda, Arriba y Abajo. De todos los bloques seleccionados, el que se toma como base es el que se encuentra más cercano a la dirección de alineamiento.

3) Completa flujos

Esta opción sólo se encuentra en la ventana de edición gráfica.

Verifica e ingresa bloques donde lo cree necesario. Ingresa bloques, en caso de no existir bloques anteriores o siguientes, de acuerdo a la siguiente lógica:

Bloque	Bloque anterior por omisión	Bloque siguiente por omisión
Generador	Nada	Cola
Cola	Generador	Servidor
Servidor	Cola	Terminador
Terminador	Cola	Nada
Ensamblador	Cola	Cola
Divisor	Server	Cola
Recurso	Nada	Nada

En caso de existir algún enlace no válido, lo deshace y sigue la misma lógica con los bloques que intervinieron en el enlace.

4) Dims. de edición...
Esta opción sólo se encuentra en la ventana de edición gráfica.
Se encarga de definir las dimensiones del área gráfica de edición. Los valores deben ser menores a 32546.

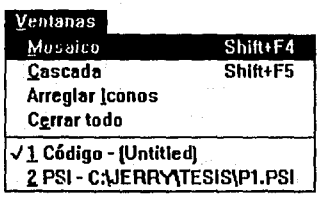
5) Esconder identificadores
(Mostrar identificadores)
Esta opción sólo se encuentra en la ventana de edición gráfica.
Permite mostrar o esconder los identificadores de los bloques de flujo. Los bloques de texto y de imagen no presentan nunca el identificador.

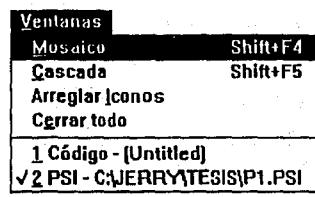
6) Mover grafo...
Esta opción sólo se encuentra en la ventana de edición gráfica.
Mueve los bloques seleccionados en forma relativa o absoluta cierto número de pixels.
La forma relativa suma o resta el valor asignado en la caja de diálogo a las coordenadas de los bloques.
La forma absoluta asigna los valores de la caja de diálogo a las coordenadas del (de los) bloque(s) seleccionado(s).
Si existen bloques seleccionados, éstos son los que se moverán. Si no existen seleccionados, moverá todo el grafo.

7) Renumerar
Esta opción sólo se encuentra en la ventana de edición gráfica.
Reasigna los identificadores según el flujo del proceso. Siempre busca el flujo principal, y posteriormente, las ramas secundarias hacia abajo.

8) Verificar flujo
Esta opción sólo se encuentra en la ventana de edición gráfica.
Verifica que el flujo tenga sentido. Muestra mensajes especificando los errores de flujo.

g. Ventana





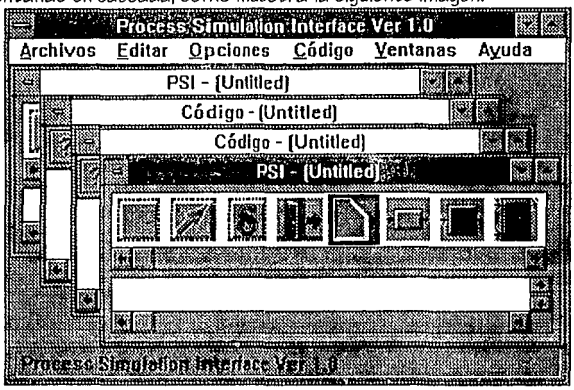
1) Arreglar iconos
Esta opción sólo se encuentra en las ventanas de edición.

Todas las ventanas que se encuentran minimizadas, son arregladas dentro del área cliente de la ventana inicial.

2) Cascada

Esta opción sólo se encuentra en las ventanas de edición.

Acomoda las ventanas en cascada, como muestra la siguiente imagen:



3) Cerrar todo

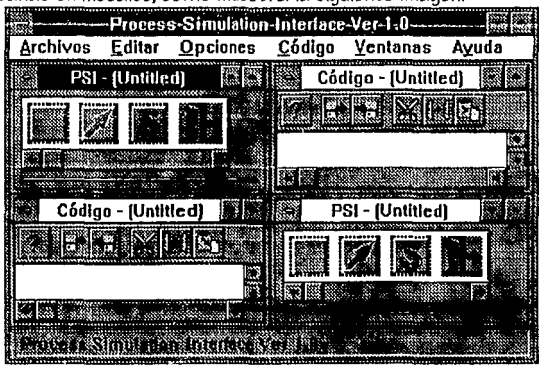
Esta opción sólo se encuentra en las ventanas de edición.

Cierra todas las ventanas dentro de PSI. Si en alguna los cambios no han sido grabados, pregunta si es necesario guardarlos. Recuerde que es responsabilidad del usuario contestar afirmativa o negativamente.

4) Mosaico

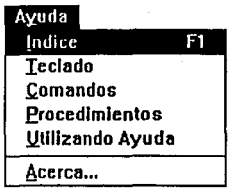
Esta opción sólo se encuentra en las ventanas de edición.

Acomoda las ventanas en mosaico, como muestra la siguiente imagen:



5) Acerca...
Esta opción sólo se encuentra en las ventanas de edición.
Muestra los créditos del programa.

h. Ayuda



1) Comandos
Esta opción se encuentra cualesquiera de las ventanas.
Muestra los comandos del lenguaje de PSI.

2) Índice
Esta opción se encuentra cualesquiera de las ventanas.
Muestra el índice de PSI.

3) Procedimientos
Esta opción se encuentra cualesquiera de las ventanas.
Muestra los procedimientos de PSI.

4) Teclado
Esta opción se encuentra cualesquiera de las ventanas.
Muestra el uso del teclado, tanto para PSI, como para Windows.

5) Utilizando ayuda
Esta opción se encuentra cualesquiera de las ventanas.
Muestra cómo se usa la ayuda en Windows, es importante saber que la presentación de esta parte de la ayuda se presentará en el idioma en el que se encuentre Windows.

5. Barra de herramientas de código

El editor de código consta de una barra de utilidades que se encuentra en la parte superior de la ventana. Esta barra consta de las siguientes funciones:

a) Ayuda.
b) Abrir archivo existente.
c) Guardar archivo en disco.
d) Borrar lo seleccionado (esto se utiliza cuando el usuario selecciona una parte del texto oprimiendo el botón izquierdo del ratón).
e) Copiar lo seleccionado al portapapeles.
f) Copiar del portapapeles al editor.

Para abrir otro proyecto, lo único necesario es seleccionar Nuevo dentro del menú de archivos (creándose dos nuevas ventanas), y seleccionando el archivo que deseamos se encuentre dentro de esa ventana (en el menú de Archivos, apartado de abrir).

6. Barra de herramientas de gráficos

El editor gráfico es una ventana diseñada para la definición gráfica de procesos discretos. En esta ventana se cuenta con una barra de utilería, y una área de definición. Cada uno de los elementos desplegados representa un bloque.
La barra de utilería cuenta con los bloques típicos de definición en un proceso, además de elementos de edición y conexión. Los elementos existentes en dicha barra son:

a) *Selección de bloques.* Este botón permite seleccionar un conjunto de bloques.
b) *Unión de bloques.* Une los bloques válidos seleccionados en orden de creación. Esta opción se puede obtener también oprimiendo una *C* en la ventana de edición gráfica.
c) *Alinear bloques.* Alineación de los bloques seleccionados en las cuatro direcciones básicas: arriba, abajo, izquierda, derecha.
d) *Generador de partes.* Es un bloque generador de entidades.
e) *Generador de partes.* Bloque generador de entidades con distinta representación gráfica.
f) *Cola.*
g) *Servidor.*
h) *Ensamble.*
i) *Sumador de flujo.*

j) *Divisor de flujo.*
k) *Procesado por lotes.* Es un servidor con distinta representación gráfica.
l) *Terminador.*
m) *Terminador.* Bloque terminador con distinta representación gráfica.
n) *Transporte.* Servidor con distinta representación gráfica.
o) *Transportador de banda.* Servidor con distinta representación gráfica.
p) *Transportador automatizado.* Servidor con distinta representación gráfica.
q) *Transportador motorizado.* Servidor con distinta representación gráfica.
r) *Grúa.* Servidor con distinta representación gráfica.
s) *Cola.*
t) *Recurso.*
u) *Texto.* Etiqueta con cualquier tipo de fuente definida en *Windows.*
v) *Gráfico.* Bloque de representación gráfica, sin ningún tipo de atributo para el proceso.

APÉNDICE B
MANUAL TÉCNICO

Ψ PROCESS SIMULATION INTERFACE

Manual Técnico

Realizado por Omar Aguirre Suárez y Gerardo Bárcena Ruiz
Universidad Panamericana, 1993

CONTENIDO

MANUAL TÉCNICO

A continuación se presentarán las unidades que fueron realizadas para el desarrollo de PSI, se entrega la documentación para cada rutina, objeto, tipo, constante o variable. Cada apartado cuenta con los parámetros de entrada y salida, su tipo, y una breve explicación de su uso. Este manual está destinado a aquellas personas que desean desarrollar una aplicación, ya sea sobre PSI, o una idea propia.

1. BASICPSI

UNIT_BasicPSI
Esta unidad es la declaración básica de los tipos y constantes que son usados en las demás unidades de PSI.
Todo aquel tipo o constante que no requiriera de las unidades creadas para PSI, fue declarado aquí. Por lo que sólo necesitó de la utilería propia de Pascal.

TYPE_Array
TA_Coef = ARRAY[1..4] Of REAL;
{Estructura para guardar los coeficientes de las ecuaciones paramétricas de las curvas Bezier}

TA_Pts = ARRAY[0..3] OF TPoint;
{Estructura que guarda los cuatro puntos de control de las curvas Bezier}

TYPE_Definidos
TD_Queue = (LIFO,FIFO,RAND);
{Tipos de colas}

TD_Direc = (Rt, UR, Up, UL, Lt, DL, Dn, DR, Null);
{Direcciones de búsqueda para el acomodo de bloques}

TD_IOMode = (NotIO, InOut);
{Tipo de entrada / salida para la escritura de archivos}

TD_MvSzMode = (Nothing_M, Move_M, SizeX_M, SizeY_M, SizeXY_M);
{Tipos de movimientos con el ratón: nada que hacer, mover el bitmap de lugar, reasignar tamaño horizontalmente, reasignar tamaño verticalmente, reasignar tamaño en diagonal}

```
TD_PrtType = (PrtScaled, PrtOnlyDisp,
PrtByPrnRes );
{Tipo de impresión: escalada, sólo lo de
pantalla y por resolución de la impresora}

TD_Rule = (Probability , Priority , ByUser);
{Tipo de regla de asociación entre
bloques}

TD_SourceDs = (byProm, byOverlap);
{Tipo de cálculo de Ds y Ds_, ya sea por
promedio o traslape}

TD_ToolName =
(SelTool      {100},
UnionTool     {101},
AlignTool     {102},
BeginTool1    {103},
BeginTool2    {104},
QueueTool     {105},
SingleOpTool  {106},
AssemTool     {107},
SumFlow       {108},
DivTool       {109},
BatchTool     {110},
EndTool1      {111},
EndTool2      {112},
TransTool     {113},
ConveyTool    {114},
AGVTool       {115},
MotorTool     {116},
CraneTool     {117},
QueueTool2    {118},
ResourceTool  {119},
TextTool      {120},
BMPTool       {121});
{Nombres definidos para los bloques o
herramientas gráficas, junto con su
identificador}
```

```
TYPE_Varios
TSetToolN = SET OF TD_ToolName;
{Conjunto de nombres de herramientas
gráficas}

TMenuState = (msEnable, msDisable);
{Estado de alguna opción del menú}

PInteger = ^INTEGER;
{Puntero a entero, diversas aplicaciones}

CONST_Varias
ToolNames : ARRAY[TD_ToolName] OF
PChar= (
'SelTool',      'UnionTool',
'AlignTool',    'Generator',
'Generator',    'Queue',
'Server',       'Assembler',
'SumFlow',      'Split',
'Batch',        'Terminator',
'Terminator',   'Transport',
'Convey',       'AGV',
'Motor',        'Crane',
'Queue',        'Resource',
'Text',         'BMPTool');
{Nombres de los bloques}

IOMode : TD_IOMode= NotIO;
{Modo incial de lectura / escritura}

MinTool = SelTool;
{Herramienta inicial, para las búsquedas
secuenciales}

MaxTool = BMPTool;
{Herramienta final, para las búsquedas
secuenciales}

EditorCount : INTEGER = 0;
{Número de editores de código}

IsPrinting : BOOLEAN= False;
```

{Identifica si UsrBmp escribe la imagen a pantalla o a la impresora}

IdClipBrdFormat : WORD = 0;
{Identificador para el tipo de protocolo del Portapapeles}

CONST_CMs
cm_PrnRes = 102;
{Archivo | Imprimir}
cm_PrnSetUp = 103;
{Archivo | Especificar impresora}
cm_EditSelAll = 207;
{Edición | Selecciona todos}
cm_EditDesel = 208;
{Edición | Deselecciona todos}
cm_EditBlk = 210;
{Edición | Edita bloques seleccionados}
cm_EditCopyImg = 211;
{Edición | Copia imagen}
cm_Compile = 501;
{Compila}
cm_OptionsSize = 444;
{Opciones | Dims. de la hoja}
cm_GenCode = 445;
{Código | Genera código}
cm_Arrange = 446;
{Opciones | Acomoda grafo}
cm_Convert = 447;
{Código | Convierta a}
cm_Attr = 449;
{Opciones | Atributos}
cm_MoveGraph = 450;
{Opciones | Mover grafo}
cm_Verify = 451;
{Opciones | Verifica flujo}
cm_Renum = 452;
{Opciones | Reenumerar}
cm_CompleteFlux= 453;
{Opciones | Completa flujo}
cm_ShowIds = 454;
{Opciones | Esconder / mostrar identificadores}
cm_Align = 455;
{Opciones | Alinear bloques}
cm_HelpIndex = 901;
{Ayuda | Índice}
cm_HelpKey = 902;
{Ayuda | Teclado}
cm_HelpComm = 903;
{Ayuda | Comandos}
cm_HelpProcs = 904;
{Ayuda | Procedimientos}
cm_HelpUtil = 905;
{Ayuda | Utilizando ayuda}
cm_About = 999;
{Ayuda | Acerca}

CONST_Ayuda
hid_A_Abri = 01; {Archivo | Abrir}
hid_A_Espe = 02; {Archivo | Especificar impresora}
hid_A_Guar = 03; {Archivo | Guardar}
hid_A_GuCo = 04; {Archivo | Guardar como}
hid_A_Impr = 05; {Archivo | Imprimir}
hid_A_Nuev = 06; {Archivo | Nuevo}
hid_A_Sali = 07; {Archivo | Salida}
hid_B_Enco = 08; {Buscar | Encontrar}
hid_B_Reem = 09; {Buscar | Reemplazar}
hid_B_Sigu = 10; {Buscar | Siguiente}
hid_C_Comp = 11; {Compilar}
hid_C_Conv = 12; {Código | Convierte a}
hid_C_Gene = 13; {Código | Genera código}
hid_E_Atri = 14; {Editar | Atributos}
hid_E_Borr = 15; {Editar | Borrar}
hid_E_BoTo = 16; {Editar | Borrar todo}
hid_E_Colm = 17; {Editar | Copiar imagen}
hid_E_Copi = 18; {Editar | Copiar}
hid_E_Cort = 19; {Editar | Cortar}
hid_E_Dese = 20; {Editar | Deseleccionar todo}
hid_E_Desh = 21; {Editar | Deshacer}

```
    hid_E_Edit   = 22;  {Editar | Edita
bloques seleccionados}
    hid_E_Inse   = 23;  {Editar | Insertar}
    hid_E_Pega   = 24;  {Editar | Pegar}
    hid_E_Sele   = 25;  {Editar | Selecciona
todo}
    hid_O_Acom = 26;  {Opciones | Acomoda
grafo}
    hid_O_Alin   = 27;  {Opciones | Alinear
bloques}
    hid_O_Comp = 28;  {Opciones |
Completar flujo}
    hid_O_Dims   = 29;  {Opciones | Dims. de
la hoja}
    hid_O_Esco   = 30;  {Opciones | Esconder
/ Mostrar identificadores}
    hid_O_Move = 31;  {Opciones | Mover
grafo}
    hid_O_Renu   = 32;  {Opciones |
Renumerar}
    hid_O_Veri   = 33;  {Opciones | Verifica
flujo}
    hid_V_Arre   = 34;  {Ventana | Arreglar
iconos}
    hid_V_Casc   = 35;  {Ventana | Cascada}
    hid_V_Cerr   = 36;  {Ventana | Cerrar
todas}
    hid_V_Mosa = 37;  {Ventana | Mosaico}
    hid_Y_Acer   = 38;  {Ayuda | Acerca}
    hid_Y_Coma = 39;  {Ayuda | Comandos}
    hid_Y_Indi   = 40;  {Ayuda | Índice}
    hid_Y_Proc   = 41;  {Ayuda |
Procedimientos}
    hid_Y_Tecl   = 42;  {Ayuda | Teclado}
    hid_Y_Util   = 43;  {Ayuda | Utilizando
Ayuda}
    hid_I_Intr   = 46;  {Introducción}
    hid_M_BarC = 50;  {Barra de
herramientas de código}
    hid_M_BarG = 51;  {Barra de
herramientas gráfica}
    hid_E_Desc   = 52;  {Descripción de
bloques}

CONST_String
    KStr_ClipBrdFormat: PChar = 'PSICF';
    {Formato para el Portapapeles}

    KStr_HelpFile: PChar = 'PSI.HLP';
    {Nombre archivo ayuda}

    KStr_AboutBMP: PChar = 'PSI_AB1.BMP';
    {Nombre archivo bitmap de Acerca...}

    KStr_MenuInit: PChar = 'MENUINIT';
    {Nombre menú ventana inicial}

    KStr_MenuGraf: PChar = 'MENUGRAPH';
    {Nombre menú ventana gráfica}

    KStr_MenuEdit: PChar = 'MENUEDIT';
    {Nombre menú ventana código}

    KStr_Title: PChar = 'Process Simulation
Interface Ver 1.0';
    { Nombre programa}

CONST_IDs
    id_BmpBut      = 10;    {Dlg: Id del botón
BMP}
    id_ResDef      = 15;    {Dlg: Id de la
definición de recursos}
    id_HorRes      = 102;   {Dlg: Id tamaño
horizontal del Canvas}
    id_VerRes      = 103;   {Dlg: Id tamaño
vertical del Canvas}
    id_PrtScaled   = 102;   {Dlg: Id impresión
a escala}
    id_PrtOnlyDisp = 103;   {Dlg: Id impresión
de sólo lo visible}
    id_PrtByPrnRes = 104;   {Dlg: Id
impresión con la resolución de la impresora}
    id_Type        = 101;   {Dlg: Id tipos de
las partes o recursos}
    id_Attr        = 102;   {Dlg: Id atributos
o variables de las partes o recursos}
```

```
  id_Abs        = 1100;  {Dlg: Id mover
grafo absolutamente}
  id_Rel        = 1101;  {Dlg: Id mover
grafo relativamente}
  id_EdX        = 1098;  {Dlg: Id mover
grafo incremento horizontal}
  id_EdY        = 1099;  {Dlg: Id mover
grafo incremento vertical}
```

2. BEZIER

UNIT_Bezier
Esta unidad tiene por finalidad la visualización y edición de curvas BEZIER de tercer grado con cuatro puntos de control. Con ellas se pueden realizar líneas rectas, curvas, buclés, etc.

FUNCTION_ASN
Parámetros
    X: REAL; {Seno del ángulo}
Salidas
    Arco Seno: REAL; {Ángulo}

PROCEDURE_BezArrow
Parámetros
    DC_: HDC; {Display Context}
    aState: P_State; {Estado general}
    P: TA_Ptos; {Ptos de Control}
    Wid: INTEGER; {Ancho de la línea}
    Col: TColorRef; {Color de la línea}
    VAR Pos: TPoint {Coors. de la caja de Selección}

Crea una flecha Bezier, manda llamar a *BezLine* y *PutArrow*, la punta de la flecha se encontrará en el punto P[3].

Ver también:
    BEZLINE
    PUTARROW

PROCEDURE_BezLine
Parámetros
    DC_: HDC; {Display Context}
    P: TA_Ptos; {Ptos de Control}
    Wid: INTEGER; {Ancho de la línea}
    Col: TColorRef; {Color de la línea}
    VAR Pos: TPoint; {Coors. de la caja de Selección}

Crea una curva Bezier, manda la curva de P[0] a P[3].

Ver también:
BEZARROW
BEZPTOS

PROCEDURE_BezPtos
Parámetros
DC_: HDC; {Display Context}
P: TA_Ptos; {Ptos de Control}
Wid: INTEGER; {Ancho de la línea}
Col: TColorRef; {Color de la línea}

Pinta los puntos de control en forma de cuadrados, unidos por líneas rectas, esta rutina es necesaria para la edición de la curva.

PROCEDURE_CalcCoef
Parámetros
P: TA_Ptos; {Ptos de Control}
VAR Cx, {ARRAY de factores para X}
Cy: TA_Coef; {ARRAY de factores para Y}

Calcula los valores de las ecuaciones paramétricas, para X y para Y, dados los puntos de control.

PROCEDURE_ChgBezLine
Parámetros
DC_: HDC; {Display Context}
Pc, Pe, {Coors. de los bloques}
{inicial y final}
Dc, De: TPoint; {Anchos y largos de los}
{bloques inicial y final}
CBack: TColorRef; {Color libre de fondo}
VAR P: TA_Ptos; {Ptos de Control}

Esta rutina cambia las coordenadas de los puntos de control de las curvas bezier, buscando una ruta libre, en la que no cruce otras líneas o bloques. Si se busca *pixel* por *pixel* y se encuentra el color *CBack*, entonces existe vía libre, de lo contrario, no la hay.

FUNCTION_Dist
Parámetros
X1, X2: REAL; {Coordenadas lineales}
Salidas
Distancia: REAL; {Distancia cuadrática}

Calcula la distancia cuadrática entre esos puntos.

FUNCTION_GetAng
Parámetros
Pc, Pe: TPoint; {Coors. bloques inicial y final}
Salidas
Ángulo: REAL; {Ángulo dirigido entre ellos}

Calcula el ángulo entre los bloques, pero es dirigido; es decir, el ángulo de Pc a Pe es 180 ° desplazado del ángulo calculado entre Pe y Pc. El ángulo siempre es normalizado.

Ver también:
NANG

FUNCTION_GetDir
Parámetros
Ang: REAL; {Ángulo dirigido entre bloques}
Salida
Direc.: TD_Direc; {Dirección entre ellos}

Dado el ángulo calculado por NANG, se calcula la dirección dirigida que existe entre los bloques, que son las siguientes:

TD_Direc = (Rt, UR, Up, UL, Lt, DL, Dn, DR, Null);

Derecha, Arriba derecha, Arriba, Arriba izquierda, Izquierda, Abajo izquierda, Abajo, Abajo derecha, Ninguno.

PROCEDURE_GetFreePos
Parámetros

DC_: HDC;	{Display Context}
Pc, Pe,	{Coors. de los bloques}
	{inicial y final}
Dc, De: TPoint;	{Anchos y largos de los}
	{bloques inicial y final}
CBack: TColorRef;	{Color libre de fondo}
VAR P: TA_Ptos;	{Ptos. de Control}

Rutina que calcula la primera posición libre alrededor del bloque.

PROCEDURE_GetMinPos
Parámetros

Pc, Pe,	{Coors. de los bloques}
	{inicial y final}
Dc, De: TPoint;	{Anchos y largos de los}
	{bloques inicial y final}
VAR P: TA_Ptos;	{Ptos. de Control}

Una de las rutinas más importantes para el Arrange; calcula la distancia mínima entre dos bloques, se basa en las coordenadas de los bloques y se obtienen las coordenadas de los puntos de control de la curva Bezier.
Ver también:
GETNEWPOS

PROCEDURE_GetNewPos
Parámetros

aState: P_State;	{Estado general}
Pc,	{Coors. bloque inicial}
Dc, De: TPoint;	{Anchos y largos de los bloques inicial y final}
N, I: BYTE;	{# de bloques a acomodar y num. de que se acomoda ahora}
VAR Pe: TPoint;	{Coors. bloque final}
VAR P: TA_Ptos;	{Ptos de control de la curva bezier}
PeNew: BOOLEAN;	{Si ya fue acomodado el bloque final}

La más importante de las rutinas de Arrange. Calcula el abanico de bloques que apunta el primero (Pc), uno a uno, según se le pida en el parámetro " I ", y en " N " se le manda cuántos son con el fin de proporcionarles las colocaciones pertinentes. Si el bloque final ya tiene posicióna asignada (NOT PeNew), entonces sólo se calculan las coordenadas de los puntos de control de las curvas bezier.

Ver también:
GETMINPOS

PROCEDURE_GetPoint
Parámetros

Pc: TPoint;	{Coors. del centro}
Long: INTEGER;	{Longitud}

Theta: REAL; {Ángulo}
VAR Pf: TPoint; {Coors. punto final}

Calcula las coordenadas del punto final de un vector polar.

FUNCTION_GoodWay
Parámetros
DC_: HDC; {Display Context}
P: TA_Ptos; {Ptos. de Control}
CBack: TColorRef;{Color libre de fondo}
Salidas
Way: BOOLEAN;{Si hay ruta libre o no}

Identifica si el camino no cruza una línea bezier, un bloque o cualquier otro elemento.

PROCEDURE_Incr
Parámetros
VAR X: INTEGER; {Variable a incrementar}
N: REAL; {valor a incrementar}

Incrementa una variable en un valor real.

FUNCTION_LongC
Parámetros
Ang_: REAL; {Ángulo entre bloques}
D_: TPoint; {Ancho y largo del bloque}
Salidas
Long: INTEGER; {Longitud correcta}

Calcula la longitud de un punto colocado alrededor de los bloques.
Da la longitud dos Pixeles fuera del bloque.

FUNCTION_NAng
Parámetros
Ang: REAL; {Ángulo inicial}
Sx, Sy: REAL {Cos y Sin del ángulo}
Salidas
Ángulo: REAL; {Ángulo final}

Normaliza un ángulo; es decir, si es negativo o el valor no está definido a la parte positiva del eje X, lo recalcula para que sea positivo.

PROCEDURE_NormPtos
Parámetros
VAR P: TA_Ptos; {Ptos de Control}

Normaliza los puntos de control de una curva bezier, recalcula las posiciones de los puntos (P[1] y P[2]) de tal manera que el segmento definido por ellos se parte en tres subsegmentos iguales.

PROCEDURE_PutArrow
Parámetros
DC_: HDC; {Display Context}
aState: P_State; {Estado General}
P: TA_Ptos; {Ptos de Control}
Wid: INTEGER; {Ancho de la línea}
Col: TColorRef; {Color de la línea}
Pinta la punta de la flecha bezier, en el punto P[3]; si el ancho de la línea es de un pixel, la pintará con dos pixels de ancho; si es mayor a 1, la pinta del ancho establecido.
Ver también:
BEZARROW

3. CANVAS

UNIT_Canvas
Esta unidad se encarga de manejar todo lo referente al área de edición gráfica. Responde, a los eventos del ratón; llamando a los bloques o herramientas correspondientes.

Se utiliza en la ventana gráfica O_LogDefWin. Existe una referencia de O_Canvas en State, de esta forma, muchos objetos tiene acceso al área de edición gráfica.

```
TYPE_O_Canvas
  {Punteros}
  PO_Canvas = ^O_Canvas;
  PO_CanvasScroller = ^O_CanvasScroller;

  O_Canvas = OBJECT(TWindow)
    PRIVATE
      BlockSel : POINTER;
      {Bloque seleccionado}
      MoveSel : BOOLEAN;
      {Indicador de movimiento}
      State : P_State;
      {Puntero al estado de la ventana gráfica}
      ToolsCrs : ARRAY[TD_ToolName] OF HCursor;
      {Crs de las herramientas}
      OtherState : E_State;

    PUBLIC
      Constructor INIT
      Destructor DONE
      Function CANCLOSE
      Procedure CLEARALL
      Procedure COPY
      Procedure COPYTOCLIPBOARD
      Procedure CUT
      Procedure DELETE
      Procedure DISABLECCD
      Procedure ENABLECCD
      Procedure ENABLECCDMENU
      Procedure MOVESELF
      Procedure PAINT
      Procedure PASTE
      Procedure READPSICF
      Procedure RELEASESELECTION
      Procedure VERIFY
      Procedure WMLBUTTONDOWN
      Procedure WMLBUTTONUP
      Procedure WMMOUSEMOVE
      Procedure WMRBUTTONRIGHT
      Procedure WMSETCURSOR
      Procedure WRITEPSICF
  END;
  O_CanvasScroller = OBJECT(TScroller)
    Procedure BEGINVIEW
  END;
```

CONSTRUCTOR_Init
Parámetros

AParent: PWindowsObj	{Ventana padre}
anState: P_State	{Puntero al estado de la ventana gráfica}

Se construyen las variables correspondientes a la ventana.

DESTRUCTOR_Done
Se destruyen las variables correspondientes a la ventana.

FUNCTION_CanClose
Salida
Si puede cerrar:BOOLEAN

Función que se manda llamar antes de cerrar una ventana. (Para mayor información consultar el manual de ObjectWindows).

PROCEDURE_ClearAll
Borra todos los bloques del área de edición.

Ver también:
CUT
DELETE
PASTE

PROCEDURE_Copy
Copia los elementos seleccionados al portapapeles. Estos elementos se copian en dos formatos:

a) El primer formato es el PSICF, que se refiere a un tipo de grabación especial.
b) El segundo formato es cf_Bitmap.

Para mayor información sobre PSICF, consultar el apartado de formatos para grabación de archivos.

Ver también:
DELETE
CUT
PASTE

PROCEDURE_CopyToClipBoard
Parámetros

DC: HDC	Contexto a copiar al portapapeles
Left,	Coordenada izquierda (X).
Top,	Coordenada superior (Y)
Width,	Ancho
Height: INTEGER	Alto

Copia una imagen delimitada por un rectángulo al portapapeles.

PROCEDURE_Cut
Copia y borra.

Ver también:
COPY
DELETE

PROCEDURE_Delete
Borra los elementos seleccionados.

Ver también:
COPY
CUT

PROCEDURE_DisableCCD
Deshabilitar las opciones de copiado, borrado y cortado.

Ver también:
ENABLECCD

PROCEDURE_EnableCCD
Habilita las opciones de copiado, cortado y borrado.

Ver también:
DISABLECCD

PROCEDURE_EnableCCDMenu
Parámetros
mf_Flag: INTEGER {Estado del elemento en el menú}

Pone el estado ingresado en Copiado, Borrado y Cortado.

Ver también:
ENABLECCD

PROCEDURE_MoveSelf
Parámetros

WX,	{Coordenada izquierda}
WY,	{Coordenada superior}
WW,	{Ancho}
WH: INTEGER	{Alto}

Repaint: BOOLEAN {Si se repinta la ventana}

Mover la ventana a las nuevas coordenadas.

PROCEDURE_Paint
Parámetros
PaintDC: HDC {Contexto del pintado}
VAR PaintInfo: TPaintStruct {Información para pintar}

Se pintan los bloques definidos y la ventana.

PROCEDURE_Paste
Pegar información del portapapeles. Primero se observa si el formato del portapapeles es el de la aplicación: *PSICF* (para mayor información sobre este formato consulte el apartado de formato de grabación); en caso de no serlo se revisa si es *cf_Bitmap*. Cuando el formato es *cf_Bitmap* se revisa si existe un bloque seleccionado, copiando la representación gráfica al bloque. Si no existe un bloque seleccionado, se crea uno nuevo y se ingresa el gráfico en el mismo.
Si el formato es *PSICF*, entonces se pregunta por el tipo de copiado y se ingresan los bloques, junto con las conexiones definidas.

Ver también:
COPY
CUT
DELETE

PROCEDURE_ReadPSICF
Parámetros
aFile:PChar {Nombre del archivo}

Lee el archivo formato *PSICF*.

Ver también:
WRITEPSICF

PROCEDURE_ReleaseSelection
Deselecciona todos los bloques.

PROCEDURE_Verify
Manda verificar la variable de estado de la ventana gráfica.

PROCEDURE_WMLButtonDown
Parámetros
VAR Msg: TMessage {Mensaje de *Windows*}

Responde al mensaje del ratón. Se utiliza ya sea para seleccionar un bloque existente o para definir uno nuevo.

Ver también:
WMMOUSEMOVE
WMLBUTTONUP
WMRBUTTONRIGHT

PROCEDURE_WMLButtonUp
Parámetros
VAR Msg:TMessage {Mensaje de *Windows*}

Responde al mensaje del ratón. Se utiliza para finalizar alguna acción sobre un bloque.

Ver también:
WMMOUSEMOVE
WMLBUTTONDOWN
WMRBUTTONRIGHT

PROCEDURE_WMMouseMove
Parámetros
VAR Msg: TMessage {Mensaje de *Windows*}

Responde al mensaje del ratón. Se utiliza para continuar la acción sobre un bloque.

Ver también:
WMLBUTTONDOWN
WMLBUTTONUP
WMRBUTTONRIGHT

PROCEDURE_WMRButtonRight
Parámetros
VAR Msg: TMessage {Mensaje de Windows}

Responde al mensaje del ratón. Se utiliza para deseleccionar todos los bloques.

Ver también:
WMLBUTTONDOWN
WMLBUTTONUP
WMLBUTTONMOVE

PROCEDURE_WMSetCursor
Parámetros
VAR Msg: TMessage {Mensaje de "Windows"}

Responde al mensaje para cambiar el aspecto del ratón, de acuerdo a la herramienta seleccionada.

PROCEDURE_WritePSICF
Escribe un archivo con el formato PSICF.

Ver también:
READPSICF

PROCEDURE_BeginView
Rutina utilizada para llevar el rastreo de la variable Offset.

4. CONVERT

UNIT_Convert
Unidad que contiene el diálogo de conversión a lenguajes de simulación.

```
TYPE_Types
  {Constantes}
  id_LBCnv = 103;
  KStr_PSI = 'PSI';

  TOpenCnvProc = Procedure(Path:PChar);
  {Tipo procedimiento para las llamadas al
  convertidor dentro del archivo DLL}
  TCnvProc = Procedure;
  {Tipo procedimiento para las llamadas al
  convertidor dentro del archivo DLL}

  PO_DlgConvert= ^O_DlgConvert;
  O_DlgConvert = OBJECT( TDialog )
    PRIVATE
      NomCnv : PChar;

    PUBLIC
      Constructor INIT
      Procedure FILLNAME
      Procedure IDLB1
      Procedure WMINITDIALOG
  END;
```

PROCEDURE_RunDllCnv
Parámetros

aDll,	{Nombre del archivo DLL}
aFile,	{Archivo que se convertirá}
aCommLine:PChar	{Línea de comandos}

La línea de comandos corresponde a una serie de parámetros capturados por el diálogo del convertidor.

CONSTRUCTOR_Init
Parámetros

AParent:PWindowsObject	{Ventana padre}
CnvSel:PChar	{Convertidor seleccionado}

Constructor para el diálogo que captura la información del convertidor seleccionado. Los convertidores instalados se encuentran en el archivo de configuración WIN.INI, en el apartado de PSI.

PROCEDURE_FillName
Copia la información seleccionada a la variable CnvSel.

PROCEDURE_IDLb1
Parámetros

VAR Msg:TMessage	{Mensaje de Windows}

Manejador de la lista de opciones.

PROCEDURE_WMInitDialog
Parámetros

VAR Msg:TMessage	{Mensaje de Windows}

Se inicializa la información del diálogo con los convertidores instalados.
Los convertidores instalados se encuentran en el archivo WIN.INI, bajo el apartado de PSI.

5. EDITGRAP

UNIT_EditGrap
Esta unidad es de las más importantes por dos razones. La primera porque aquí se unen todas las rutinas de las demás unidades (es donde se realiza la edición gráfica, que es la finalidad de PSI); y la segunda porque aquí es donde se inicializa la variable de estado (State) que contiene la información de la instancia global de PSI.

```
CONST_Varias
  id_WidthField       = 101;
  id_HeightField      = 102;
  id_StretchBM        = 104;
  id_PadBM            = 105;
  id_CurrentBMGroup       = 106;

  id_QMGFile          = 108;
  id_RunVaran         = 109;
  id_Queues           = 110;
  id_PASFile          = 111;
  id_ParentP          = 112;
  id_PredefP          = 113;
```

Identificadores para el diálogo de unión con PASION.

```
TYPE_LogDefWin
  PO_LogDefWin = ^O_LogDefWin;
  O_LogDefWin = OBJECT (TWindow)
    State : E_State;
    {Estado del sistema, variable original}
    ToolBar : PO_GraphTB;
    {Barra de herramientas}
    Canvas : PO_Canvas;
    {Espacio de edición gráfica}
    FileName : ARRAY [0..fsPathName] OF
CHAR;
    {Nombre del archivo a editar}
    CBChainNext : HWnd;
```

{Ventana de conexión con el Portapapeles}

Constructor INIT
Destructor DONE
Procedure ARRANGE
Procedure CALCMINMAX
Function CANCLOSE
Procedure CMALIGN
Procedure CMARRANGE
Procedure CMATTR
Procedure CMCOMPLETEFLUX
Procedure CMCONVERT
Procedure CMEDITBLK
Procedure CMEDITCLEAR
Procedure CMEDITCOPY
Procedure CMEDITCOPYIMG
Procedure CMEDITCUT
Procedure CMEDITDELETE
Procedure CMEDITDESEL
Procedure CMEDITPASTE
Procedure CMEDITSELALL
Procedure CMFILEOPEN
Procedure CMFILESAVE
Procedure CMFILESAVEAS
Procedure CMGENCODE
Procedure CMMOVEGRAPH
Procedure CMOPTIONSSIZE
Procedure CMPRNRES
Procedure CMRENUM
Procedure CMSHOWIDS
Procedure CMVERIFY
Procedure CREATEGRAPH
Procedure DELETECOOR
Function GETCLASSNAME
Procedure GETWINDOWCLASS
Procedure MODIFYVERTPOS
Procedure MOVEGRAPH
Procedure READFILE
Procedure READLNK
Procedure SETDS
Procedure SETNAMES
Procedure SETUPWINDOW
Procedure UPDATECHILDREN
Procedure WMCHANGECBCHAIN
Procedure WMCHAR
Procedure WMDESTROY
Procedure WMDRAWCLIPBOARD
Procedure WMMDIACTIVATE
Procedure WMSIZE
Procedure WRITEFILE
END;

PO_DlgPasion= ^O_DlgPasion;
O_DlgPasion = OBJECT(TDialog)
PPathPASFile,
{Ruta al archivo de atributos}
PRunVaran,
{Si de desea correr Varan}
PQueues,
{(1,2,3,4,5, ... Colas)}
PParentProcs,
{Si se desea correr procesos padres}
PPredefProcs : PChar;
{Procesos predefinidos}

Constructor INIT_
Procedure OK_
Procedure SETSTR_
Procedure WMINITDIALOG_
END;

Function PREGDLGPASION
{Controlador del diálogo anterior}

CONSTRUCTOR_Init
Parámetros
AParent : PWindowsObject
{Ventana padre}
ATitle : PChar
{Archivo a editar}

Se asigna el nombre del archivo que se va a editar y se crea la ventana.
Se incrementa el contador de editores.

Se dan de alta los campos de la variable de estado general State, que es la variable a la que hacen referencia las demás; mediante un puntero.
Se crea la pluma que cambia de color y ancho las curvas bezier.
Se crea el objeto que sirve de unión con los simuladores.
Se asigna el menú gráfico.
Se crea el Canvas. Esto es el espacio lógico para la edición gráfica, es otra ventana, donde se insertan las imágenes que identifican a los bloques.
Se crea la barra de herramientas.
Se inicializa la unión con el Portapapeles.

DESTRUCTOR_Done
Se decrementa el contador de editores.
Se manda el mensaje de destrucción para la ventana de edición de código.
Se destruyen:

- el objeto de unión para los simuladores.
- el bloque por omisión.
- las colecciones de bloques (flujo de proceso), y la de los identificadores de los bloques seleccionados.
- el tipo de letra lógico.
- la pluma de cambio de color y ancho de las curvas bezier.
- la barra de herramientas.
- el Canvas.
- la ventana.

PROCEDURE_Arrange
Esta rutina es mandada llamar por READFILE y CMARRANGE, y tiene por finalidad acomodar los bloques de manera ordenada y evitando los traslapes entre bloques.
Renumera los bloques.
Establece las distancias de espacio vital para los bloques, por medio de un promedio de alto y ancho de todos los bloques insertados en el Canvas, y posteriormente crea el grafo.
Verifica que no exista un traslape que el creador no pudo resolver. Si esto no ocurre, en esta ocasión se calculan las distancias del espacio vital para los bloques por las distancias de los traslapes más grandes que ocurran en todo el grafo.
Si existió traslape, se borran las coordenadas (sin recalcular el número de bloques que se apuntan hacia atrás), y se crea el grafo con las distancias vitales nuevas. Ahora, se procederá a resolver los traslapes verticales (si ocurrieron), como en el caso de líneas paralelas.
Se calculan los máximos y mínimos del grafo, se mueve todo el grafo hacia la esquina superior izquierda, y se deja un margen de 10 Pixels.

PROCEDURE_CalcMinMax
Esta rutina calcula los mínimos y máximos del grafo, es decir las posiciones extremas de los bloques, primero para los bloques, y luego para las uniones o curvas bezier. Esta rutina es auxiliar en el acomodo del grafo.

FUNCTION_CanClose
Salidas
  ¿Se puede cerrar? : BOOLEAN

Verifica si se puede cerrar la ventana. Si el campo Change de State se encuentra activado, indica de una modificación en el grafo, si es así se pregunta si se desea guardar el archivo, la respuesta afirmativa o negativa es responsabilidad de usuario.

PROCEDURE_CMAlign
Parámetros
  VAR Msg : TMessage
Responde a:
  Opciones | Alinear

Esta rutina alinea los bloques, tomando como base el bloque que se encuentre más extremo en la dirección de alineamiento, a este bloque se le designa como el pivote. Por ejemplo: si se desean alinear a la derecha, se busca el bloque que está más a la derecha, y su posición en X es asignada a los demás bloques. Debe haber bloques seleccionados.

NOTA:

1) Cuando deseé enviar datos entre rutinas que son llamadas por los servicios de las colecciones, que deben ser FAR, es posible realizarlo por medio de constantes con tipo (constantes-variables), para que los datos lleguen sin basura, de lo contrario suceden errores graves.

2) El número máximo de llamadas de servicios de las colecciones es igual a tres (3). Por ejemplo: una rutina que posee un *ForEach* (manda llamar a otra que posee un *ForEach* (que llama a otra, que no llama ningún servicio de colecciones)). Si no se cuida la recomendación anterior, pueden suceder errores graves.

PROCEDURE_CMArrange
Parámetros
VAR Msg : TMessage
Responde a:
Opciones | Alinear bloques

Borra las coordenadas, calculando el número de bloques apuntados hacia atrás, se llama la rutina ARRANGE, y se limpia la pantalla.

PROCEDURE_CMAttr
Parámetros
VAR Msg : TMessage
Responde a:
Editar | Atributos de la parte...

Edita los atributos de la parte, que circulan por el flujo, y son generados lógicamente por los generadores, para que los simuladores puedan tomar sus características.
Debe referirse a los manuales de PASION para hacer uso conveniente de esta opción.

PROCEDURE_CMCompleteFlux
Parámetros
VAR Msg : TMessage
Responde a:
Opciones | Completa los flujos

Verifica que el flujo sea consistente, de lo contrario muestra los mensajes de error correspondientes.

PROCEDURE_CMConvert
Parámetros
VAR Msg : TMessage
Responde a:
Código | Convierte a...

Corre el DLL de conversión, es decir, traduce el metalenguaje de PSI, a un formato reconocible, por ejemplo para PASION. Y si se encuentra PASION instalado, se ejecutará la simulación correspondiente. Previamente graba el archivo a disco para la traducción.

PROCEDURE_CMEditBlk
Parámetros
VAR Msg : TMessage
Responde a:
Editar | Edita los bloques seleccionados...

Edita el bloque seccionado, donde se asignan las características para la simulación, se cambia el identificador visual (*BitMap*), el tipo letra, etc. Si se encuentra más de un bloque seleccionado, se presentarán los diálogos en el orden de selección.

PROCEDURE_CMEditClear
Parámetros
VAR Msg : TMessage
Responde a:
Editar | Borrar todo

Llama la rutina del *Canvas* que borra todos los bloques insertados.

PROCEDURE_CMEditCopy
Parámetros
VAR Msg : TMessage
Responde a:
Editar | Copiar

Llama la rutina del *Canvas* que copia todos los bloques seleccionados al Portapapeles.

PROCEDURE_CMEditCopyImg
Parámetros
VAR Msg : TMessage
Responde a:
Editar | Copiar imagen

Llama la rutina del *Canvas* que copia la parte de la ventana que se encuentra visualizada (como imagen) y la manda al Portapapeles.

PROCEDURE_CMEditCut
Parámetros
VAR Msg : TMessage
Responde a:
Editar | Cortar

Llama la rutina del *Canvas* que corta todos los bloques seleccionados.

PROCEDURE_CMEditDelete
Parámetros
VAR Msg : TMessage
Responde a:
Editar | Borrar

Llama la rutina del *Canvas* que borra todos los bloques seleccionados.

PROCEDURE_CMEditDesel
Parámetros
VAR Msg : TMessage
Responde a:
Editar | Deselecciona todo

Llama la rutina del *Canvas* que deselecciona todos los bloques insertados.

PROCEDURE_CMEditPaste
Parámetros
VAR Msg : TMessage
Responde a:
Editar | Pegar

Llama la rutina del *Canvas* que pega los bloques copiados.

PROCEDURE_CMEditSelAll
Parámetros
VAR Msg : TMessage
Responde a:
Editar | Selecciona todo

Selecciona todos los bloques insertados.

PROCEDURE_CMFileOpen
Parámetros
VAR Msg : TMessage
Responde a:
Archivo | Abrir...

Si el archivo actual ha sido modificado, y no ha sido guardado, pregunta si lo desea hacer. Posteriormente se llama al diálogo estándar de apertura de archivos y se buscan aquellos con extensión .PSI (que es la propia del formato gráfico).

PROCEDURE_CMFileSave
Parámetros
VAR Msg : TMessage
Responde a:
Archivo | Guardar

Si el archivo no tiene nombre manda llamar a CMFILESAVEAS, de lo contrario guarda el archivo.

PROCEDURE_CMFileSaveAs
Parámetros
VAR Msg : TMessage
Responde a:
Archivo | Guardar como...

Llama al diálogo estándar para salvar archivos, sugiriendo la extensión .PSI que es la propia del formato gráfico, y guarda el archivo.

PROCEDURE_CMGenCode
Parámetros
VAR Msg : TMessage
Responde a:
Código | Genera código...

Traduce de la configuración gráfica del flujo o grafo, a una definición en palabras que describe las características, relaciones y posiciones de los bloques insertados. Este código o metalenguaje se escribe en la ventana de edición de código, que es la pareja de la de edición gráfica. Se manda llamar el generador de código de cada bloque para que escriba la información correspondiente.

PROCEDURE_CMMoveGraph
Parámetros
VAR Msg : TMessage
Responde a:
Opciones | Mover grafo...

Esta rutina mueve todo el grafo o los bloques seleccionados. Los modos de movimiento pueden ser absolutos o relativos; es decir, si es absoluto se le asignan a todos los bloques la misma posición. Por el contrario; si es relativo, se suma o resta el factor para cada dirección. Las posiciones de las uniones se recalculan , por lo que es necesario editar de nuevo las curvas de formas elaboradas.
Manda llamar a MOVEGRAPH

PROCEDURE_CMOptionsSize
Parámetros
VAR Msg : TMessage
Responde a:
Opciones | Dims. de edición...

El tamaño del espacio de edición gráfica o *Canvas*, originalmente es de 30000 para cada dirección, pero el usuario puede reasignar el tamaño con esta rutina.
Cuidado: los bloques que hayan quedado fuera del espacio pueden perderse, por lo que es conveniente reacomodar el grafo después de esta operación.

PROCEDURE_CMPrnRes
Parámetros
VAR Msg : TMessage
Responde a:
Archivo | Imprimir

Pregunta el tipo de impresión que se desea realizar: escalado, de marco visible, o con la resolución del impresor.
El escalado imprime todo el *Canvas* en una hoja de impresión, por lo que si es muy grande, la imagen impresa será bastante pequeña.
La impresión del marco visible es aquélla que manda a impresión sólo la parte de la pantalla de se ve en ese instante.

La impresión por resolución manda un *pixel* de pantalla a un *dot* del impresor; por lo que si se cuenta con una impresora de muy baja resolución es probable que el grafo sobrepase las dimensiones del papel, por el contrario, si se cuenta con mayor resolución puede quedar una impresión de muy buena calidad.

PROCEDURE_CMRenum
Parámetros
VAR Msg : TMessage
Responde a:
Opciones | Reenumerar

Reenumera los bloques, reasigna los identificadores y posiciones en la colección, según su posición en el flujo de bloques, siempre en orden a los flujos principales de izquierda a derecha y de arriba hacia abajo.

PROCEDURE_CMShowIds
Parámetros
VAR Msg : TMessage
Responde a:
Opciones | Mostrar/Esconder identificadores

Muestra o esconde los identificadores de los bloques.

PROCEDURE_CMVerify
Parámetros
VAR Msg : TMessage
Responde a:
Opciones | Verificar flujo

Verifica que el flujo sea coherente, según las políticas para cada tipo de bloque. Si existe alguna irregularidad se presentan los mensajes correspondientes.

PROCEDURE_CreateGraph
Esta rutina crea el grafo, se ocupa cuando se lee un archivo LNK o cuando se acomoda el grafo. Primero se calcula el número de bloques apuntados hacia atrás y después se asignan las posiciones.
Se pretende que el grafo quede ordenado en forma de abanico que abre hacia la derecha, esto se logra con la rutina GETNEWPOS; pero cuando se apunta a un bloque que se encuentra atrás del que se acomoda ahora, los restantes deben de tomar la posición correcta en el abanico. Por eso se calculan los que son apuntados hacia atrás primero, para asignar a la curva bezier su posición correcta con GETMINPOS.
Para correr este procedimiento es necesario que las posiciones de los bloques dentro de la colección sea exactamente como las del flujo a crear; por ejemplo, el primer bloque (*IdNum* = 1) debe ser un generador.

NOTA:

1) Cuando deseé enviar datos entre rutinas que son llamadas por los servicios de las colecciones, que deben ser FAR, es posible realizarlo por medio de constantes con tipo (constantes-variables), para que los datos lleguen sin basura, de lo contrario suceden errores graves.

2) El número máximo de llamadas de servicios de las colecciones es igual a tres (3). Por ejemplo: una rutina que posee un *ForEach*(manda llamar a otra que posee un *ForEach*(que llama a otra, que no llama ningún servicio de colecciones)). Si no se cuida la recomendación anterior, pueden suceder errores graves.

PROCEDURE_DeleteCoor
Parámetros
CheckApunt : BOOLEAN
{Si se calculan los bloques apuntados}

Borra las coordenadas de todos los bloques y curvas bezier, asignando un -1 a sus coordenadas, excepto al bloque con *IdNum* = 1, que se asigna el cero, pues este es el pivote de inicio para crear al grafo. Cuando el parámetro *CheckApunt* se encuentra activo se recalcula el número de bloques que apunta el bloque activo, esto para que cuando se crea el grafo, se puedan asignar las coordenadas precisas para formar el abanico.

FUNCTION_GetClassName
Salidas
'O_LogDefWin' : PChar {Nombre de la clase}

Entrega el nombre de la clase de ventana.

PROCEDURE_GetWindowClass
Parámetros
VAR WndClase : TWndClass {Datos de la clase}

Investiga los datos de la clase heredada. Asigna el icono que identifica a los editores gráficos, y asigna el color de fondo.

PROCEDURE_ModifyVertPos
Acomoda los bloques de forma vertical. Cuando existe un traslape y las rutinas de asignación de distancias vitales y creación de grafo fallaron, existen bloques que se encuentran en líneas paralelas. Para remediar esto, se cambian de posición los bloques traslapados, poniendo más abajo el de mayor *IdNum*; por lo que el grafo debe estar acomodado numéricamente para no fallar. La distancia que se mueve hacia abajo está determinada por el alto del grafo de menor *IdNum* y el tamaño del identificador.

PROCEDURE_MoveGraph
Parámetros
dX, dY : INTEGER {Factor en X, Y}

Mueve todo el grafo, sumando o restando el factor que se manda como parámetro para cada dirección. Si existen bloques seleccionados, se hace caso omiso de esta característica y se modifica la posición de todos los bloques insertados (además de sus uniones).

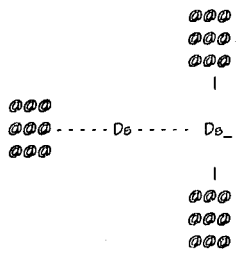
PROCEDURE_ReadFile
Lee un archivo para presentarlo en la pantalla. Si el formato del archivo es .PSI, se procesa la información. Si por el contrario el formato .LNK, se llama a la rutina READLNK. El formato .PSI es el característico del editor gráfico.

PROCEDURE_ReadLNK
Si el archivo que se está leyendo es .LNK, se ejecuta esta rutina, aquí se dan de alta todos los bloques y uniones según el formato establecido (Ver Manual de Usuario). El archivo debe ser de texto plano, y esta rutina la manda llamar READFILE.

PROCEDURE_SetDs
Parámetros
Source : TD_SourceDs
{Tipo de cálculo a realizar}
VAR ExistOverlap : BOOLEAN
{Si existe traslape}

En esta rutina se pueden hacer dos tipos de cálculos, por Promedio y por Traslape.
En el primer caso se investigan los altos y anchos de todos los bloques insertados y se promedian para obtener las distancias vitales de los bloques. Si los bloques son de tamaño constante, es más que suficiente

para que no existan traslapes. La variable *ExistOverlap* en este caso no se usa.
Para el segundo caso, se investiga si hay traslape, y se escoge el de mayor distancia como base de cálculo; pues es el más crítico. Se reasignan los valores de las distancias vitales, y se entera a la rutina de acomodo -mediante *ExistOverlap*- que existió traslape y que se debe de recrear el grafo.
Las distancias vitales se guardan en los campos Ds y Ds_ de la variable general de estado (*State*). Abajo se muestra que indican los campos anteriores.



NOTA:

1) Cuando deseé enviar datos entre rutinas que son llamadas por los servicios de las colecciones, que deben ser FAR, es posible realizarlo por medio de constantes con tipo (constantes-variables), para que los datos lleguen sin basura, de lo contrario suceden errores graves.

2) El número máximo de llamadas de servicios de las colecciones es igual a tres (3). Por ejemplo: una rutina que posee un *ForEach(* manda llamar a otra que posee un *ForEach(* que llama a otra, que no llama ningún servicio de colecciones)). Si no se cuida la recomendación anterior, pueden suceder errores graves.

PROCEDURE_SetNames
Parámetros
NewName : PChar {Nuevo nombre de archivo}

Cada vez que cambia el nombre de un archivo, se ocupa esta rutina para reasignar el título de la ventana.

PROCEDURE_SetupWindow
Se da de alta la el objeto padre, y se investiga si existen datos en el Portapapeles que puedan ser susceptibles de ser copiados en esta ventana.

PROCEDURE_UpdateChildren
Cada vez que ocurre un cambio en la ventana de edición gráfica, se informa al *Canvas* para que se acomode según el nuevo tamaño.

PROCEDURE_WMChangeCBChain
Parámetros
VAR Msg : TMessage

Reasigna la unión con el Portapapeles.
PROCEDURE_WMChar
Parámetros
VAR Msg : TMessage
Esta rutina atrapa y ejecuta las órdenes de las macros para PSI.
Las macros corresponden a las letras:
'S' = CMEDITSELALL
'C' = DoUnion (*Ver Servers*)
'D' = CMEDITDESEL

PROCEDURE_WMDestroy
Parámetros
VAR Msg : TMessage

Destruye la unión con el Portapapeles.
Envía el mensaje de destrucción de la ventana.

PROCEDURE_WMDrawClipBoard
Parámetros
VAR Msg : TMessage

Verifica si el formato de los datos contenidos en el Portapapeles pueden ser pegados en esta ventana, para activar o no los comandos pertinentes del menú Editar.

PROCEDURE_WMMdiActivate
Parámetros
VAR Msg : TMessage

Verifica si la ventana de edición gráfica se encuentra activa para asignar el menú correspondiente. De lo contrario asigna el menú de la ventana inicial o principal. Manda también pintar la barra de herramientas según sea el caso.

PROCEDURE_WMSize
Parámetros
VAR Msg : TMessage

Modifica el tamaño o posición de la ventana, y notifica a las ventanas hijas (en este caso sólo al *Canvas*) de este cambio.

PROCEDURE_WriteFile
Escribe en disco el archivo activo con el formato de la ventana de edición gráfica (.PSI).

CONSTRUCTOR_Init_
Parámetros
AParent : PWindowsObject
{Ventana padre}
aPASFile,
{Nombre del archivo en PASCAL}
aRunVaran,
{Si se desea correr VARAN de PASION}
aQueue,
{Tipos de colas}
aParentProcs,
{Si se desea correr procesos de padres}
aPredefProcs : PChar
{Si se desea correr procesos predefinidos}

Inicializa un diálogo con los datos para unión con PASION.

PROCEDURE_SetStr_
Parámetros
id : INTEGER
{Identificador del editor a leer}
VAR Str_ : PChar
{Cadena a investigar}

Obtiene una línea de texto de un editor específico. Recuerde que este editor es de un diálogo.

PROCEDURE_Ok_
Parámetros
VAR Msg : TMessage

Si el diálogo fue aceptado, se leen los datos para la unión con PASION.

PROCEDURE_WMInitDialog_
Parámetros
VAR Msg : TMessage

Inicializa el diálogo con las opciones que se asignaron en el constructor.

FUNCTION_PregDlgPasion
Parámetros
anState : P_State
{Estado general del sistema}
Parent : PWindowsObject
{Ventana padre}
FN,
{Ruta de acceso}
CommLine : PChar
{Línea de comandos}

Salidas
Estatus : INTEGER;
{Si fueron aceptadas las opciones}

Controlador del diálogo de PASION.

6. EDITWIN

UNIT_EditWin
Esta es la ventana de edición de código, básicamente es un editor estándar de Windows. Guarda estrecha relación con la ventana de edición gráfica, pues si se crea o se destruye una, sucede por igual a la otra, siempre deben trabajar en parejas, de lo contrario puede haber problemas graves.

TYPE_EditLangWin
PO_EditLangWin = ^O_EditLangWin;
O_EditLangWin = OBJECT(TFileWindow)
GraphWnd : PO_LogDefWin;
{Puntero a la ventana de edición gráfica}
ToolBar : PO_EditTB;
{Barra de herramientas de edición de código}

Constructor INIT
Destructor DONE
Procedure CMCOMPILE
Procedure CMMDIFILEOPEN
Procedure CMPRNRES
Function GETCLASSNAME
Procedure GETWINDOWCLASS
Procedure WMMDIACTIVATE
Procedure WMSIZE
END;

PO_PSIEditor= ^O_PSIEditor;
O_PSIEditor = OBJECT(TEdit)
Procedure CMEDITDELETE
END;

CONSTRUCTOR_Init
Parámetros
AParent : PWindowsObject
{Ventana padre}
AFileName : PChar
{Nombre del archivo a contener}

Copia el nombre del archivo a contener en la variable interna del editor, inicializa el editor, da de alta la barra de herramientas e incrementa el contador de editores. Si no viene el nombre del archivo en los parámetros, se asigna uno, que será cambiado por el usuario cuando se ejecuta el comando para guardar el archivo.

DESTRUCTOR_Done
Decrementa el contador de editores, manda el mensaje a la ventana de edición gráfica de que debe cerrarse (pues siempre deben de trabajar en parejas). Destruye la barra de herramientas, y destruye la ventana de edición de código.

PROCEDURE_CMCompile
Parámetros
  VAR Msg : TMessage
Responde a:
  Compilar | *vacío*

Convierte la información en código (archivo ASCII) a una presentación gráfica en su ventana hermana gráfica. Todas las uniones se recalculan , por lo que si se tenía una configuración especial de curvas bezier, se pierde y debe de volverse a realizar.

PROCEDURE_CMMDIFileOpen
Parámetros
  VAR Msg : TMessage
Responde a:
  Archivo | Abrir...

Manda llamar al diálogo de apertura de archivos estándar.

PROCEDURE_CMPrnRes
Parámetros
  VAR Msg : TMessage
Responde a:
  Archivo | Imprimir

Manda llamar al objeto de impresión, e imprime si es posible.

FUNCTION_GetClassName
Salidas
  'FileEditor' : PChar;

Entrega el nombre de la clase de ventana.

PROCEDURE_GetWindowClass
Parámetros
  VAR AWndClass : TWndClass {Datos de la clase}

Manda llamar los datos de la clase del objeto padre, y asigna el icono que identifica a las ventana de edición de código.

PROCEDURE_WMMdiActivate
Parámetros
  VAR Msg : TMessage

Si la ventana de código se encuentra activa,se cambia al menú del editor de código. De lo contrario se establece el menú de la ventana inicial o principal.

PROCEDURE_WMSize
Parámetros
  VAR Msg : TMessage

Si cambió la posición o el tamaño de la ventana, se recalculan las propias del editor y de la barra de herramientas.

PROCEDURE_CMEditDelete
Parámetros
  VAR Msg : TMessage

Esta rutina borra caracteres, el editor original sólo podía suprimirlos por medio de la

orden de un *backspace*; pero esta rutina permite borrarlos por medio de un suprimir (*delete*) también.

7. PEN

UNIT_Pen
Este diálogo es utilizado en la edición de las curvas bezier, para el cambio de color y ancho de la curva. Para accionar este diálogo es necesario hacer un clic en el cuadro central del la curva bezier, y una vez que aparecen los otros cuatro cuadros (para la edición de la posición de los puntos de control), se repite la operación (clic en el cuadro central) y entonces aparecerá la ventana del diálogo.

```
CONST_Colores
   ColorAttr: ARRAY[0..15] OF Longint = (
      {Black}     $000000,
      {Blue}      $800000,
      {Green}     $008000,
      {Cyan}      $808000,
      {Red}       $000080,
      {Magenta}   $800080,
      {Brown}     $008080,
      {L Gray}    $C0C0C0,
      {D Gray}    $808080,
      {L Blue}    $FF0000,
      {L Gree}    $00FF00,
      {L Cyan}    $FFFF00,
      {L Red}     $0000FF,
      {L Magenta} $FF00FF,
      {Yellow}    $00FFFF,
      {Write}     $FFFFFF);
```

Esta es la lista de los 16 colores sólidos en VGA, que se utiliza para la asignación de los mismos en el diálogo. La asignación está en el formato RGB; pero en este caso se encuentra en orden inverso:es decir, BGR. Verifique esto con los tres colores básicos.

```
TYPE_O_Pen
   PO_DlgPen = ^O_DlgPen;
   PO_Pen = ^O_Pen;
```

```
{Punteroes a los objetos}

E_PenData = RECORD
   XWidth : ARRAY[0..6] OF CHAR;
   ColorArray : ARRAY[0..15] OF WORD
END;
{Registro de transferencia de}
{datos del diálogo al programa}

O_DlgPen = OBJECT(TDialog)
   Constructor INIT
END;
{Controlador del diálogo}

O_Pen = OBJECT(TObject)
   Procedure CHANGEPEN
END;
{Cambio de color y ancho}
```

FUNCTION_GetColorAttr
Parámetros
ARec : E_PenData
Salidas
Color de cambio : LONGINT

Entrega el color que el usuario asignó en la ventana de diálogo.

PROCEDURE_SetColorAttr
Parámetros
VAR ARec : E_PenData{Buffer de datos}
AColor : LONGINT {Color actual}

Establece el color en el buffer de datos según el color que viene en la variable AColor.

CONSTRUCTOR_Init
Parámetros
AParent : PWindowsObject
{Padre al que reportar}
AName : PChar
{Nombre del diálogo}

Este es controlador del diálogo, que es mandado llamar por CHANGEPEN.

PROCEDURE_ChangePen
Parámetros
VAR PenData : E_PenData {Buffer de datos}
VAR Width : INTEGER {Ancho actual}
VAR Color : LONGINT {Color actual}

Esta rutina manda construir el diálogo, y si la respuesta fue afirmativa se procede a la traducción de los datos provenientes del usuario.

8. PREVIEW

UNIT_Preview
Esta unidad tiene por objetivo dar una presentación preliminar de la librería de imágenes o BMPs con que cuenta PSI. Si se da un clic sobre la imagen cambiará a una presentación del tamaño real del BMP. Si se repite la operación cambiará nuevamente a un tamaño aumentado. Si se acepta la imagen el diálogo entrega el nombre y ruta del archivo que la contiene.

```
CONST_Varias
   sd_FileOpen  = $7F00; {Nombre del
diálogo}
   id_FName     = 100;   {Dlg: Id nombre y
ruta del archivo}
   id_FPath     = 101;   {Dlg: Id ruta del
archivo}
   id_FList     = 102;   {Dlg: Id caja de
archivos}
   id_DList     = 103;   {Dlg: Id caja de
directorios}
   fsFileSpec   = fsFileName +fsExtension;
                         {Tamaño de la
ruta}

TYPE_O_DlgPreview
   PO_DlgPreview = ^O_DlgPreview;
   O_DlgPreview = OBJECT(TDialog)
      XIniBMP,
         {Pos. horizontal}
      YIniBMP,
         {Pos. vetical}
      HBmpV,
         {Alto del área a pintar}
      WBmpV     : INTEGER;
         {Ancho del área a pintar}
      BitMap    : PO_UsrBMP;
         {Manejador del BitMap}
      Caption   : PChar;
         {Título}
      FilePath  : PChar;
         {Ruta del archivo}
      PathName : ARRAY[0..fsPathName]
OF CHAR;
         {Ruta del archivo}
      Extension  : ARRAY[0..fsExtension] OF
CHAR;
         {Estensión del archivo}
      FileSpec   : ARRAY[0..fsFileSpec] OF
CHAR;
         {Nombre del archivo}

      Construnctor INIT
      Destructor DONE
      Function CANCLOSE
      Procedure DRAWBMP
      Procedure HANDLEDLIST
      Procedure HANDLEFLIST
      Procedure HANDLEFNAME
      Procedure SETUPWINDOW
      Procedure WMLBUTTONDOWN
   PRIVATE
      Procedure SELECTFILENAME
      Procedure UPDATEFILENAME
      Function UPDATELISTBOXES
   END;

CONSTRUCTOR_Init
Parámetros
   AParent : PWindowsObject      {Padre
al cual reportar}
   AName,                        {Nombre del
diálogo}
   AFilePath : PChar             {Nombre y ruta
del archivo}

Inicializa el diálogo.

DESTRUCTOR_Done
Destruye el diálogo, y el objeto UsrBmp que
maneja las imágenes.
```

FUNCTION_CanClose
Salida
Si se puede cerrar : BOOLEAN

Verifica que el nombre y la ruta sean válidos, de lo contrario no destruye el diálogo, a menos que se presione el botón de cancelación. Aquí es donde se calcula y entrega en el *puntero* la ruta final del archivo.

PROCEDURE_DrawBMP
Pinta el bitmap sobre el área del diálogo. No se hace uso del DC del diálogo, ni se pinta con las rutinas de pintado del propio diálogo; pues son incompatibles, por lo que se investigan las coordenadas y se pintan según éstas.

De lo contrario causa un error bastante GRAVE.

PROCEDURE_HandleDList
Parámetros
VAR Msg : TMessage

Controla la caja de directorios.

PROCEDURE_HandleFList
Parámetros
VAR Msg : TMessage

Controla la caja de archivos.

PROCEDURE_HandleFName
Parámetros
VAR Msg : TMessage

Controla el desplegado de la ruta y nombres del directorio activo.

PROCEDURE_SetUpWindow
Inicializa las listas y despliega el directorio actual.

Aquí también se dan de alta las posiciones iniciales para el pintado del *bitmap* y, alto y ancho del área del diálogo donde se mostrará la imagen.

PROCEDURE_WMLButtonDown
Parámetros
VAR Msg : TMessage

Procedimiento para cambiar el modo de desplegado de la imagen, alternando entre tamaño normal y aumentado. Además de las funciones propias del control del diálogo para la búsqueda del archivo *bmp*.

PROCEDURE_SelectFileName
Investiga la ruta y nombre activos.

PROCEDURE_UpDateFileName
Actualiza el desplegado de ruta y nombre del archivo según los cambios de directorios.

FUNCTION_UpDateListBoxes
Salida
Si actualizó : BOOLEAN

Actualiza las cajas de archivos y directorios según los cambios de directorios.

9. PRNUNIT

UNIT_PrnUnit
Esta es la unidad de impresión. Puede imprimir de memoria o de disco, dependiendo de la construcción del objeto. Recuerde que la impresión en *Windows* es por medio del *Display Context* (DC), por lo que debe verificar su dispositivo de impresión se encuentra bien configurado para dar la salida que se desea.

```
TYPE_TextPrint
   PO_CharCollection = ^O_CharCollection;
   O_CharCollection = OBJECT(TCollection)
      Procedure FREEITEM
   END;
   {Colección de líneas de impresión}

   PO_TextPrint = ^O_TextPrint;
   O_TextPrint = OBJECT (TPrintOut)
      PRIVATE
         TextHeight,
            {Alto del texto}
         LinesPerPage,
            {Líneas por hoja impresa}
         FirstOnPage,
            {Primera línea en la hoja}
         LastOnPage : INTEGER;
            {Última línea en la hoja}
         TheLines : PO_CharCollection;
            {Colección de líneas}

      PUBLIC
         Constructor INIT
         Constructor INITEDIT
         Destructor DONE
         Function GETDIALOGINFO
         Function HASNEXTPAGE
         Procedure LEEARCHIVO
         Procedure PRINTPAGE
         Procedure READFROMEDIT
         Procedure SETPRINTPARAMS
   END;
   { Impresor }
```

PROCEDURE_FreeItem
Parámetros
   Item : Pointer {Línea}
Libera la memoria la una línea específica.

CONSTRUCTOR_Init
Parámetros
   ATitle : PChar {Título}
   FileName : PChar {Archivo que provee el texto}
Inicializa el impresor con las líneas de un archivo (de disco).

CONSTRUCTOR_InitEdit
Parámetros
   ATitle : PChar {Título}
   Edit : PEdit {Editor de texto}
Inicializa el impresor con las líneas de un editor (de memoria).

DESTRUCTOR_Done
Destruye la colección *TheLines*.

FUNCTION_GetDialogInfo
Parámetros
   VAR Pages : INTEGER {Páginas totales}
Salidas
   Siempre True : Boolean
Investiga cuántas hojas se obtienen, de la razón entre el número total de líneas en el documento y el número de líneas por hoja de impresión.

FUNCTION_HasNextPage
Parámetros
   Page : Word {Uso reservado}
Salidas
   ¿Más hojas? : Boolean
Investiga si existen más hojas.

PROCEDURE_ReadFile
Parámetros
NomArch : PChar {Nombre y ruta del archivo}

Toma las líneas de un archivo y las inserta en la colección *TheLines*, para posteriormente se impresas.

PROCEDURE_PrintPage
Parámetros
Page : Word {Número de página}
VAR Rect : TRect{Uso reservado}
Flags : Word {Uso reservado}

Toma las líneas que se insertaron el la colección *TheLines*, y las envía a la impresora; manda de hoja en hoja.

PROCEDURE_ReadFromEdit
Parámetros
anEdit : PEdit {*Puntero a un editor*}

Toma línea por línea de un editor de texto, y la inserta en la colección *TheLines*, posteriormente se imprimirá. En cada ocasión inserta sólo los caracteres usados por cada línea; pues no existe una longitud constante de la línea.

PROCEDURE_SetPrintParams
Parámetros
ADC : HDC {*Display Context*}
ASize : TPoint {Alto de la hoja}

Investiga los parámetros de impresión para el ADC y asigna los valores correspondientes a las variables *TextHeight* (Altura del carácter) y *LinesPerPage* (Líneas por hoja de impresión).

10. PSI

INTERFACE_PROG_PSI
Recursos Utilizados
PSI.RES
{Menús y barras de herramientas}

STRINGS.RES
{Mensajes de la barra de estado}

BMIcoCur.RES
{Bitmaps de los bloques}

Unidades utilizadas
BasicPsi, EditGrap, EditWin, Objects, OMemory, OPrinter, OStdDlgs, OWindows, RoutPsi, Strings, TypePsi, UsrBMP, ViewBMP, WinDos, WinProcs, WinTypes;

Este es el programa principal que controla a PSI. Aquí se cargan los recursos principales. Todos los recursos de cada unidad, se cargan en sus propias instancias.

```
TYPE_FrameWin
  PO_FrameWin = ^O_FrameWin;
  O_FrameWin = OBJECT (TMDIWindow)
    PRIVATE
      StBar : PO_StBar;
        {Bara de estado y ayuda}

    PUBLIC
      Constructor INIT
      Destructor DONE
      Procedure ABOUT
      Procedure GETWINDOWCLASS
      Procedure HELPCOMM
      Procedure HELPINDEX
      Procedure HELPKEY
      Procedure HELPPROC
      Procedure HELPUTIL
      Procedure NEWFILE
      Procedure OPENFILE
```

```
        Procedure PRNSETUP
        Procedure SETUPWINDOW
        Procedure SHIFTF1
        Procedure WMCOMMAND
        Procedure WMENTERIDLE
        Procedure WMLBUTTONDOWN
        Procedure WMMENUSELECT
        Procedure WMSETCURSOR
        Procedure WMSIZE
    END;
    {Ventana principal multi documentos}

    O_PSIApp = OBJECT(TApplication)
      Procedure INITMAINWINDOW
      Procedure INITINSTANCE
      Function CANCLOSE
    END;
    {Aplicación para Windows}
```

CONSTRUCTOR_Init
Parámetros
ATitle : PChar {Título}
aMenu : HMenu {Menú de la ventana inicial}

Inicializa la ventana principal, el *puntero de* impresión y la barra de estado o ayuda.

DESTRUCTOR_Done
Destruye el *puntero de* impresión.
Destruye el *puntero de* la barra de estado o ayuda.
Destruye las ventanas hijas.
Se destruye a sí misma.

PROCEDURE_About
Responde a:
Archivo | Acerca...

Manda llamar el objeto *O_ViewBMP* de la unidad *ViewBMP* para presentar el *bitmap* de créditos del programa. La ventana siempre se presenta centrada en la pantalla.

PROCEDURE_GetWindowClass
Parámetros
VAR AWndClass : TWndClass
{Datos de la clase de la ventana}

Manda llamar la clase heredada y asigna el icono del programa.

PROCEDURE_HelpComm
Parámetros
VAR Msg : TMessage
Responde a:
Ayuda | Comandos

Llama la ayuda de PSI sobre comandos.

PROCEDURE_HelpIndex
Parámetros
VAR Msg : TMessage
Responde a:
Ayuda | Índice

Presenta el índice de la Ayuda de PSI.

PROCEDURE_HelpKey
Parámetros
VAR Msg : TMessage
Responde a:
Ayuda | Teclado

Llama la ayuda de PSI sobre el manejo del teclado.

PROCEDURE_HelpProc
Parámetros
VAR Msg : TMessage
Responde a:
Ayuda | Procedimientos

Llama la ayuda de PSI sobre procedimientos.

PROCEDURE_HelpUtil
Parámetros
VAR Msg : TMessage
Responde a:
Ayuda | Utilizando Ayuda

Llama a la ayuda propia de Windows, recuerde que esta opción se presentará en el idioma base de la instalación de Windows, y no en el idioma de PSI.

PROCEDURE_NewFile
Parámetros
VAR Msg : TMessage
Responde a:
Archivo | Nuevo

Crea un par de ventanas de edición nuevas. No tienen nombre y se encuentran vacías.

PROCEDURE_OpenFile
Parámetros
VAR Msg : TMessage
Responde a:
Ayuda | Arbrir...

Manda llamar el diálogo para seleccionar el nombre y ruta de un archivo. Una vez realizado esto se crea la ventana hija que contendrá la información del archivo.

PROCEDURE_PrnSetUp
Parámetros
VAR Msg : TMessage
Responde a:
Archivo | Imprimir...

Inicializa las características para imprimir e imprime si es posible.

PROCEDURE_SetupWindow
Llama a la actualización de la ventana principal y asigna las opciones desactivadas del menú.

PROCEDURE_ShiftF1
Parámetros
VAR Msg : TMessage

Establece el cursor y la posibilidad de realizar una ayuda sensible; es decir, que cuando cambia el ratón a una forma de mano con un signo de interrogación presionando las teclas Shift + F1 y posicionándose sobre cualquier menú, la ayuda aparecerá sobre ese tema en particular.

PROCEDURE_WMCommand
Parámetros
VAR Msg : TMessage

Este es el traductor entre un mensaje del menú, y el tema de ayuda que se desea mostrar con el mecanismo de Shift + F1.

PROCEDURE_WMEnterIdle
Parámetros
VAR Msg: TMessage

Este procedimiento ingresa un Enter en la cola de eventos para que pueda tener lugar el desplegado de la ayuda sensible.

PROCEDURE_WMLButtonDown
Parámetros
VAR Msg : TMessage

Si se encuentra la ayuda sensible activa, y se da un clic en el área cliente de la ventana, se manda llamar la introducción del documento de ayuda de PSI.

PROCEDURE_WMMenuSelect
Parámetros
VAR Msg : TMessage

Muestra la descripción de cada submenú en la barra de estado o ayuda, según se mueve por los menús.

PROCEDURE_WMSetCursor
Parámetros
VAR Msg : TMessage

Establece el cursor en forma de mano y signo de interrogación, si se encuentra la ayuda sensible activa. De lo contrario se ejecuta el procedimiento de la ventana por omisión, que se encargará del establecer el cursor pertinente.

PROCEDURE_WMSize
Parámetros
VAR Msg : TMessage

Calcula el nuevo tamaño, y asigna la nueva posición a la barra de estado, y manda moverse a sus ventanas hijas.

PROCEDURE_PutPresentation
Parámetros
X, Y : INTEGER {Posición}
aFileBMP : PChar {Archivo a mostrar}
TimeLimit : LONGINT {Tiempo de desplegado}

Muestra un *bitmap* en pantalla, escribiéndolo sobre el *desktop* (sin ventana por supuesto). Después que el tiempo ha transcurrido continúa la ejecución del programa. Si el parámetro X toma el valor de -1 el *bitmap* se presentará centrado en la pantalla.

PROCEDURE_InitMainWindow
Muestra los *bitmaps* de presentación, registra el formato para el Portapapeles, carga el menú para la ventana inicial o principal e inicializa la ventana principal.

PROCEDURE_InitInstance
Inicializa la instancia del programa. Carga los aceleradores y los menús del editor de código y gráfico.

FUNCTION_CanClose
Salidas
¿Se puede cerrar? : Boolean

Verifica si se pueden cerrar tanto las ventanas hijas como la padre. De ser así destruye los menús de las ventanas inicial, de código y gráfica.

11. ROUTPSI

UNIT_RoutPsi
Esta unidad contiene una colección de rutinas que se usan en muchas otras y fue necesario almacenarlas en un solo sitio. Toda rutina que dependía de forma importante de otra unidad se dejó en aquellas unidades. Aquí están las que se les podría denominar rutinas independientes.

TYPE_Strings
Str20 = ARRAY[0..20] OF CHAR;
{String de 20 caracteres = STRING[20] }

Str80 = ARRAY[0..80] OF CHAR;
{String de 80 caracteres = STRING[80] }

FUNCTION_Ask
Parámetros
Quest : PChar { Pregunta }
Salidas
Sí o No : BOOLEAN

Rutina que pregunta, y la respuesta puede ser sólo Sí o No.

FUNCTION_AskCancel
Parámetros
Quest : PChar { Pregunta }
Salidas
Id de respuesta : INTEGER

Rutina que pregunta, y la respuesta puede ser afirmativa, negativa o cancelar la pregunta.

FUNCTION_Confirm
Parámetros
Msg : PChar { Sentencia }
Salidas
Confirmar o Cancelar : BOOLEAN

Diálogo que sirve para confirmar una orden, con la posibilidad de cancelar la operación siguiente, si no se debe de realizar.

PROCEDURE_DecEditors
Decrementa el contador de Editores (EditorCount), y si llega a cero desactiva los menús correspondientes.

FUNCTION_DoFontDlg
Parámetros
AParent : PWindowsObject
{Ventana padre}
LF : PLogFont
{Puntero a un tipo de letra lógico}
ATitle : PChar
{Título}
Salidas
Si hubo selección : BOOLEAN

Selecciona un tipo de letra lógico, con todas sus características, tipo, estilo(subrayado, cursiva, etc.), así como el tamaño de la fuente.

FUNCTION_FileDialog
Parámetros
VAR Path : PChar {Ruta de acceso inicial}
Which : PChar {Tipo de diálogo, Abrir o Guardar}
Salidas
Si hubo éxito : BOOLEAN

Esta rutina manda llamar al diálogo estándar de Windows para abrir o guardar archivos, la ruta con los comodines (* y ?) vienen en *Path*, y el tipo de diálogo en *Which*.

FUNCTION_FileOpenDialog
Parámetros
Path : PChar {Ruta de acceso inicial}
Salidas
Si hubo éxito : BOOLEAN

Manda llamar a FILEDIALOG, con tipo de diálogo para abrir archivos.

FUNCTION_FileSaveDialog
Parámetros
  Path : PChar {Ruta de acceso inicial}
Salidas
  Si hubo éxito : BOOLEAN

Manda llamar a FILEDIALOG, con tipo de diálogo para guardar archivos.

PROCEDURE_GetAtPos
Parámetros
  Line : PChar {Línea de código}
  VAR X , Y : INTEGER {Valores X, Y}

Dada una línea de código, investiga los valores X y Y, los convierte de cáracter a número y los devuelve en las variables.

PROCEDURE_GetBlkCom
Parámetros
  ChDef : PChar {Segmento a buscar}
  Line, {Línea de código}
  Attr, {Línea sin segmento}
  Expression : PChar {Segmento}

Convierte a mayúsculas Line. Busca a ChDef en Line, si la encuentra copia de Line a Expression pero en una posición después. Copia Line - ChDef en Attr. Quita espacios a Attr y Expression.

PROCEDURE_GetBlkComEqual
Parámetros
  Line, {Línea de código}
  Attr, {Segmento antes del =}
  Expression : PChar {Segmento después del =}

Manda llamar GETBLKCOMdonde el segmento a buscas es '='.

PROCEDURE_GetRule
Parámetros
  Line : PChar {Línea de código}
  VAR Rule : TD_Rule {Regla de asociación}
  ByUsr : PChar {Regla del usuario}

Dada una línea de código, se investiga el tipo de regla de asociación entre bloques.

PROCEDURE_IncEditors
Incrementa el contador de Editores (EditorCount), y si es cero, reestablece los menús.

PROCEDURE_InsEndIn
Parámetros
  EditWnd : PFileWindow {Ventana de edición}

Inserta un END; en una ventana de edición. Lama a INSSTR.

PROCEDURE_InsLineIn
Parámetros
  Wnd : PFileWindow {Ventana de edición}
  CmdLine : PChar {Línea de código}
  VAR Dat {Formato}

Inserta una línea completa de código en una ventana de edición, seguida de un enter. El formato se establece en Dat.

PROCEDURE_InsStr
Parámetros
  EditWnd : PFileWindow {Ventana de edición}
  StrToIns : PChar {Línea de código}

Inserta una línea completa de código en una ventana de edición, seguida de un enter.

PROCEDURE_InsStrLineIn
Parámetros
Wnd : PFileWindow {Ventana de edición}
CmdLine : PChar {Línea de código}
Param : PChar {Formato}

Inserta una línea completa de código en una ventana de edición. Manda llamar a INSSTRLINEIN.

PROCEDURE_MenuItems
Parámetros
State : TMenuState {Estado del menú}

Modifica el estado del menú, activándolo o desactivándolo según sea el caso. Los submenús que se ven afectados son aquellos que tienen que ver con edición y uso del Portapapeles; así como los procedimientos de almacenamiento.

PROCEDURE_QuitaSp
Parámetros
Line : PChar {Línea de código}

Esta rutina tiene como finalidad suprimir los espacios de una línea de código.

PROCEDURE_StrDispose_
Parámetros
VAR Str_ : PChar {Puntero a liberar}

Libera la memoria de una cadena de caracteres.

PROCEDURE_StrNew_
Parámetros
VAR StrSource : PChar
{Puntero a una cadena de caracteres}
Const StrCopy_ : PChar
{Cadena a copiar}

Entrega la memoria suficiente a la variable StrSource para contener a StrCopy_, y después copia StrCopy_ a StrSource.

PROCEDURE_Tell
Parámetros
Msg : PChar {Mensaje a mostrar}

Presenta un mensaje en pantalla y un botón para desaparecerlo.

PROCEDURE_TellError
Parámetros
Line : PChar {Línea de código con el error}
ErrorCode, {Código del error}
NumLine : INTEGER {Número de línea}

Muestra la línea, el error, y el número de línea donde existe un error en el código de un archivo PSI.

12. SERVERS

UNIT_Servers_
En esta unidad se define todos los bloques de aplicación, junto con sus diálogos; además, contiene el traductor de código. Esta unidad parte de las definiciones en *TypePSI*.

CONST_Ids
{Todos los identificadores abajo mostrados, representan los elementos en los diálogos}

{Identificadores del diálogo de división de flujos}
id_ByProb = 12;
id_ByPrio = 13;
id_ByUsr = 14;
id_EditByUsr = 102;

{Identificadores del diálogo de generadores}
id_GenInitVal = 102;
id_GenDist = 103;
id_GenMax = 104;
id_GenEntGpo = 105;

{Identificadores del diálogo de colas}
id_QueInitLon = 102;
id_QueCostPer = 103;
id_QueMax = 104;
id_QueLIFO = 12;
id_QueFIFO = 13;
id_QueRAND = 14;

{Identificadores del diálogo de servidores}
id_SrvTpoAdic = 102;
id_SrvDist = 103;
id_SrvCostPerT = 104;
id_SrvCostPerO = 105;
id_SrvInps = 11;
id_SrvOuts = 12;

{Identificadores del diálogo de terminadores}
id_TerNoRest = 102;
id_TerMaxPartes= 103;

{Identificadores del diálogo del bloque de texto}
id_Font = 13;
id_Text = 102;

TYPE_*Punteros*
PO_Assemble = ^O_Assemble;
PO_Batch = ^O_Batch;
PO_BMPBlk = ^O_BMPBlk;
PO_Divider = ^O_Divider;
PO_Generator = ^O_Generator;
PO_Group = ^O_Group;
PO_Queue = ^O_Queue;
PO_Resource = ^O_Resource;
PO_Select = ^O_Select;
PO_Server = ^O_Server;
PO_SingleOp = ^O_SingleOp;
PO_SumFlow = ^O_SumFlow;
PO_Terminator = ^O_Terminator;
PO_TextBlk = ^O_TextBlk;
PO_Transport = ^O_Transport;
PO_AGV = ^O_AGV;
PO_Convey = ^O_Convey;
PO_Crane = ^O_Crane;
PO_MotorTrsp = ^O_MotorTrsp;
PO_Union = ^O_Union;
PO_Translate = ^O_Translate;
PO_DlgGetAttrRes = ^O_DlgGetAttrRes;
PO_DlgGetInfoAss = ^O_DlgGetInfoAss;
PO_DlgGetInfoFont = ^O_DlgGetInfoFont;
PO_DlgGetInfoGen = ^O_DlgGetInfoGen;
PO_DlgGetInfoQueue=
^O_DlgGetInfoQueue;
PO_DlgGetInfoSrv = ^O_DlgGetInfoSrv;
PO_DlgGetInfoTer = ^O_DlgGetInfoTer;
PO_DlgTypeDiv = ^O_DlgTypeDiv;

TYPE_O_BlockChk
{Clase que da servicios de revisión de flujos utilizados en O_Verify}

```
O_BlockChk = OBJECT(O_Block)
  Procedure CheckWith(Prevs:BOOLEAN;
  WhichNOT:TSetToolN;TypeCompatible:T
  D_ToolName);
  Procedure BuildUnions;VIRTUAL;
END;
```

Ver también:
- O_DIVIDER
- O_GENERATOR
- O_QUEUE
- O_TERMINATOR
- O_UNION

TYPE_O_Server
{Clase que define el comportamiento básico de un servidor}

```
O_Server = OBJECT(O_BlockChk)
Distribution,          {Distribución
                       que define el
                       tiempo de
                       servicio}
AddDelay,              {Tiempo
                       añadido al salir
                       del servidor}
CostPerTime,           {Costo por
                       unidad de
                       tiempo}
CostPerOper : PChar;   {Costo por
                       operación}

  Constructor Init(anState:P_State);
  Destructor Done; VIRTUAL;
  Procedure CheckUnions; VIRTUAL;
  Procedure
  GenCode(EditWnd:PFileWindow);
  VIRTUAL;
  Function GetClassName:PChar;
  VIRTUAL;
  Procedure GetInfoFromUser; VIRTUAL;
  Function IsLinkableOut:BOOLEAN;
  VIRTUAL;
  Procedure Load(VAR S:TStream);
  VIRTUAL;
  Function ReadComm(aLine:PChar;VAR
  NumLine:INTEGER;Edit:PEdit):BOOLEA
  N; VIRTUAL;
  Procedure Store(VAR S:TStream);
  VIRTUAL;
END;
```

Ver también:
- O_BLOCKCHK
- O_ASSEMBLE
- O_BATCH
- O_SINGLEOP
- O_SUMFLOW
- O_TRANSPORT

TYPE_O_Assemble
{Clase que define el comportamiento de un ensamblador}

```
O_Assemble = OBJECT(O_Server)
  Procedure CheckUnions; VIRTUAL;
  Procedure
  GenCode(EditWnd:PFileWindow);
  VIRTUAL;
  Function GetClassName:PChar;
  VIRTUAL;
  Procedure GetInfoFromUser; VIRTUAL;
  Function TypeTool:TD_ToolName;
  VIRTUAL;
END;
```

Ver también:
- O_SERVER

TYPE_O_Batch
{Clase que define el comportamiento de un procesador por lotes}

```
O_Batch = OBJECT(O_Server)
  Procedure
  GenCode(EditWnd:PFileWindow);
  VIRTUAL;
  Function GetClassName:PChar;
  VIRTUAL;
  Function TypeTool:TD_ToolName;
  VIRTUAL;
END;
```

Ver también:
O_SERVER

TYPE_O_BmpBlk
{Clase que define el comportamiento de un bloque de representación gráfica}

```
O_BMPBlk = OBJECT(O_Block)
  Procedure
  GenCode(EditWnd:PFileWindow);
  VIRTUAL;
  Function GetClassName:PChar;
  VIRTUAL;
  Function IsLinkable:BOOLEAN; VIRTUAL;
  Procedure Paint(PaintDC: HDC; VAR
  PaintInfo: TPaintStruct);VIRTUAL;
  Function TypeTool:TD_ToolName;
  VIRTUAL;
END;
```

TYPE_O_Divider
{Clase que define el comportamiento de un divisor de flujos}

```
O_Divider = OBJECT(O_BlockChk)
  Constructor Init(anState:P_State);
  Procedure CheckUnions; VIRTUAL;
  Procedure
  GenCode(EditWnd:PFileWindow);
  VIRTUAL;
  Function GetClassName:PChar;
  VIRTUAL;
  Procedure GetInfoFromUser; VIRTUAL;
  Function IsLinkableIn:BOOLEAN;
  VIRTUAL;
  Function ReadComm(aLine:PChar;VAR
  NumLine:INTEGER;Edit:PEdit):BOOLEA
  N; VIRTUAL;
  Function TypeTool:TD_ToolName;
  VIRTUAL;
END;
```

Ver también:
O_SERVER

TYPE_O_Generator
{Clase que define el comportamiento de un generador de partes}

```
O_Generator = OBJECT(O_BlockChk)
  Distribution:PChar;     {Distribución
                           para generación
                           de partes}
  InitTime :PChar;        {Tiempo incial
                           necesario para
                           generar la
                           primera parte}
  GenGroups :PChar;       {Número de
                           partes para
                           formar un
                           grupo}
  MaxNumber :PChar;       {Número
                           máximo de
                           entidades a
                           generar}

  Constructor
  Init(anState:P_State;aBMP:PChar);
  Destructor Done; VIRTUAL;
  Procedure CheckUnions; VIRTUAL;
```

```
    Procedure
    GenCode(EditWnd:PFileWindow);
    VIRTUAL;
    Function GetClassName:PChar;
    VIRTUAL;
    Procedure GetInfoFromUser; VIRTUAL;
    Function IsLinkableIn:BOOLEAN;
    VIRTUAL;
    Function IsLinkableOut:BOOLEAN;
    VIRTUAL;
    Procedure Load(VAR S:TStream);
    VIRTUAL;
    Function ReadComm(aLine:PChar;VAR
    NumLine:INTEGER;Edit:PEdit):
    BOOLEAN; VIRTUAL;
    Procedure Store(VAR S:TStream);
    VIRTUAL;
    Function TypeTool:TD_ToolName;
    VIRTUAL;
  END;
```

Ver también:
O_BLOCKCHK

TYPE_O_Queue
{Clase que define el comportamiento de un bloque de colas}

```
  O_Queue = OBJECT(O_BlockChk)
  CostPerUnit : PChar;      {Costo por
                             unidad}
  InitLen : PChar;          {Longitud inicial
                             de la cola}
  MaxNumber : PChar;        {Máximo
                             número de
                             elementos en la
                             cola}
  Tipo : TD_Queue;          {Tipo de cola
                             FIFO,LIFO,RAND
                             }

    Constructor Init(anState:P_State);
    Destructor Done; VIRTUAL;
    Procedure CheckUnions; VIRTUAL;
    Procedure
    GenCode(EditWnd:PFileWindow);
    VIRTUAL;
    Function GetClassName:PChar;
    VIRTUAL;
    Procedure GetInfoFromUser; VIRTUAL;
    Function IsLinkableIn:BOOLEAN;
    VIRTUAL;
    Function IsLinkableOut:BOOLEAN;
    VIRTUAL;
    Procedure Load(VAR S:TStream);
    VIRTUAL;
    Function ReadComm(aLine:PChar;VAR
    NumLine:INTEGER;Edit:PEdit):
    BOOLEAN; VIRTUAL;
    Procedure Store(VAR S:TStream);
    VIRTUAL;
    Function TypeTool:TD_ToolName;
    VIRTUAL;
  END;
```

Ver también:
O_BLOCKCHK

TYPE_O_Resource
{Clase que define a los recursos}

```
  O_Resource = OBJECT(O_Block)
  Types,                    {Tipos del
                             recurso}
  Attributes: PStrCollection;
                            {Atributos del
                             recurso}

    Constructor Init(anState:P_State);
    Destructor Done; VIRTUAL;
    Procedure
    GenCode(EditWnd:PFileWindow);
    VIRTUAL;
    Function GetClassName:PChar;
    VIRTUAL;
    Procedure GetInfoFromUser; VIRTUAL;
```

```
    Function IsLinkable:BOOLEAN; VIRTUAL;
    Procedure Load(VAR S:TStream);
    VIRTUAL;
    Function ReadComm(aLine:PChar;VAR
    NumLine:INTEGER;Edit:PEdit):BOOLEA
    N; VIRTUAL;
    Procedure Store(VAR S:TStream);
    VIRTUAL;
    Function TypeTool:TD_ToolName;
    VIRTUAL;
  END;
```

TYPE_O_Select
{Clase que define la herramienta de selección}

```
  O_Select = OBJECT(O_Block)
  Pen:HPen;                {Plumilla
                           utilizada para
                           la definición}

    Constructor Init(anState:P_State);
    Destructor Done; VIRTUAL;
    Function GetClassName:PChar;
    VIRTUAL;
    Procedure MouseDown(AWindow: HWnd;
    X, Y: INTEGER; anState: P_State);
    VIRTUAL;
    Procedure MouseMove(X, Y: INTEGER);
    VIRTUAL;
    Procedure MouseUp; VIRTUAL;
    Function TypeTool:TD_ToolName;
    VIRTUAL;
  END;
```

TYPE_O_SingleOp
{Clase que define el servidor básico}

```
  O_SingleOp = OBJECT(O_Server)
    Procedure
    GenCode(EditWnd:PFileWindow);
    VIRTUAL;
    Function GetClassName:PChar;
    VIRTUAL;
    Function TypeTool:TD_ToolName;
    VIRTUAL;
  END;
```

Ver también:
O_SERVER

TYPE_O_SumFlow
{Sumador de flujos}

```
  O_SumFlow = OBJECT(O_Server)
    Procedure
    GenCode(EditWnd:PFileWindow);
    VIRTUAL;
    Function GetClassName:PChar;
    VIRTUAL;
    Function TypeTool:TD_ToolName;
    VIRTUAL;
  END;
```

Ver también:
O_SERVER

TYPE_O_Terminator
{Clase que define el bloque terminador de partes}

```
  O_Terminator= OBJECT(O_BlockChk)
  NoPartsDec,              {Número de
                           partes a
                           restar}
  NoMaxParts: PChar;       {Número
                           máximo de
                           partes}

    Constructor
    Init(anState:P_State;aBMP:PChar);
    Destructor Done; VIRTUAL;
    Procedure CheckUnions; VIRTUAL;
```

```
      Procedure
      GenCode(EditWnd:PFileWindow);
      VIRTUAL;
      Function GetClassName:PChar;
      VIRTUAL;
      Procedure GetInfoFromUser; VIRTUAL;
      Function IsLinkableIn:BOOLEAN;
      VIRTUAL;
      Function IsLinkableOut:BOOLEAN;
      VIRTUAL;
      Procedure Load(VAR S:TStream);
      VIRTUAL;
      Function ReadComm(aLine:PChar;VAR
      NumLine:INTEGER;Edit:PEdit):BOOLEA
      N; VIRTUAL;
      Procedure Store(VAR S:TStream);
      VIRTUAL;
      Function TypeTool:TD_ToolName;
      VIRTUAL;
   END;
Ver también:
   O_BLOCKCHK

TYPE_O_TextBlk
{Herramienta para texto}

   O_TextBlk = OBJECT(O_Block)
   Font:HFont;              {Tipo de letra}
   PText:PChar;             {Texto}

      Constructor Init(anState:P_State);
      Destructor Done; VIRTUAL;
      Procedure ChangeFont(NewOne:hFont);
      Procedure
      ChangeFontInd(NewOne:TLogFont);
      Procedure
      GenCode(EditWnd:PFileWindow);
      VIRTUAL;
      Function GetClassName:PChar;
      VIRTUAL;
      Procedure GetInfoFromUser; VIRTUAL;
      Function IsLinkable:BOOLEAN; VIRTUAL;
      Procedure Load(VAR S:TStream);
      VIRTUAL;
      Procedure MouseDown(AWindow: HWnd;
      X, Y: INTEGER; anState: P_State);
      VIRTUAL;
      Procedure MouseMove(X, Y: INTEGER);
      VIRTUAL;
      Procedure MouseUp; VIRTUAL;
      Procedure Paint(PaintDC: HDC; VAR
      PaintInfo: TPaintStruct);VIRTUAL;
      Procedure SetCurs(X,Y:INTEGER);
      VIRTUAL;
      Procedure Store(VAR S:TStream);
      VIRTUAL;
      Function TypeTool:TD_ToolName;
      VIRTUAL;
   END;

TYPE_O_Transport
{Transportadores, se definen como
servidores}

   O_Transport = OBJECT(O_Server)
      Procedure
      GenCode(EditWnd:PFileWindow);
      VIRTUAL;
      Function GetClassName:PChar;
      VIRTUAL;
      Function TypeTool:TD_ToolName;
      VIRTUAL;
   END;

   O_AGV = OBJECT(O_Transport)
      Procedure
      GenCode(EditWnd:PFileWindow);
      VIRTUAL;
      Function GetClassName:PChar;
      VIRTUAL;
      Function TypeTool:TD_ToolName;
      VIRTUAL;
   END;

   O_Convey = OBJECT(O_Transport)
```

```
      Function GetClassName:PChar;
      VIRTUAL;
      Function TypeTool:TD_ToolName;
      VIRTUAL;
      Procedure
      GenCode(EditWnd:PFileWindow);
      VIRTUAL;
   END;

   O_Crane = OBJECT(O_Transport)
      Procedure
      GenCode(EditWnd:PFileWindow);
      VIRTUAL;
      Function GetClassName:PChar;
      VIRTUAL;
      Function TypeTool:TD_ToolName;
      VIRTUAL;
   END;

   O_MotorTrsp = OBJECT(O_Transport)
      Procedure
      GenCode(EditWnd:PFileWindow);
      VIRTUAL;
      Function GetClassName:PChar;
      VIRTUAL;
      Function TypeTool:TD_ToolName;
      VIRTUAL;
   END;
```

Ver también:

O_SERVER

TYPE_O_Union

{Elemento que representa la unión de bloques}

```
   O_Union = OBJECT(O_BlockChk)
   XY : TA_Pts;              {Puntos que
                              definen la curva
                              Bezier}
   ToDelete,                 {Bandera de
                              invalidez}
   Back,                     {Si la unión
                              apunta hacia
                              atrás}
   EditLine : BOOLEAN;       {Modo de
                              edición}
   OldColLine,               {Color anterior}
   ColLine : TColorRef;      {Color de la
                              línea}
   WidthLine : INTEGER;      {Ancho de la
                              línea}
   LineData : E_PenData;     {Información de
                              la línea}

      Constructor Init(anState:P_State;
      ShowUnion: BOOLEAN);
      Constructor
      InitComp(anState:P_State;Id1,Id2:INTE
      GER);
      Procedure Deselect; VIRTUAL;
      Procedure CheckUnions; VIRTUAL;
      Function GetClassName:PChar;
      VIRTUAL;
      Function IsIn(X,Y:INTEGER):BOOLEAN;
      VIRTUAL;
      Function IsInRgn(X,Y:INTEGER; P:
      TPoint): BOOLEAN; VIRTUAL;
      Function IsLinkable:BOOLEAN; VIRTUAL;
      Procedure KeyChar(Key, Count,
      lParamHi: WORD); VIRTUAL;
      Procedure Load(VAR S:TStream);
      VIRTUAL;
      Procedure MouseDown(AWindow: HWnd;
      X, Y: INTEGER; anState: P_State);
      VIRTUAL;
      Procedure MouseMove(X, Y: INTEGER);
      VIRTUAL;
      Procedure MouseUp; VIRTUAL;
      Function
      NeedToBeDeleted(IdDel:INTEGER):BOO
      LEAN; VIRTUAL;
      Procedure Paint(PaintDC: HDC; VAR
      PaintInfo: TPaintStruct); VIRTUAL;
      Procedure Recalc;
```

```
      Procedure Select; VIRTUAL;
      Procedure SetCurs(X,Y:INTEGER);
      VIRTUAL;
      Procedure Store(VAR S:TStream);
      VIRTUAL;
      Function TypeTool:TD_ToolName;
      VIRTUAL;
      Procedure UnlinkBlks;
   END;

Ver también:
   O_BLOCKCHK

TYPE_O_Translate
{Traductor de código}

   O_Translate = OBJECT( TObject )
      PRIVATE
         Parent: PWindowsObject;
         State : P_State;
         Edit : PEdit;

      PUBLIC
         Constructor
         Init(aParent:PWindowsObject;anSta
         te:P_State;anEdit:PEdit);
         Procedure Run; VIRTUAL;
   END;

TYPE_O_Dlgs
{Diálogos de cada uno de los tipos de bloque}

{Diálogo de atributos}

   O_DlgGetAttrRes = OBJECT(
   O_DlgGetInfo )
      Constructor
      Init(AParent:PWindowsObject;aBlk:PO_
      Resource);
      Procedure Ok(VAR
      Msg:TMessage);VIRTUAL id_First +
      id_Ok;
      Procedure WMInitDialog(VAR
      Msg:TMessage); VIRTUAL wm_First +
      wm_InitDialog;
   END;

{Diálogo del servidor}

   O_DlgGetInfoSrv =
   OBJECT(O_DlgGetInfo)
      Constructor
      Init(AParent:PWindowsObject;aBlock:P
      O_Block);
      Procedure
      AskARule(WhatRule:PO_Rule);
      Procedure Inputs(VAR Msg:TMessage);
      VIRTUAL id_First + id_SrvInps;
      Procedure Ok(VAR
      Msg:TMessage);VIRTUAL id_First +
      id_Ok;
      Procedure Outputs(VAR
      Msg:TMessage); VIRTUAL id_First +
      id_SrvOuts;
      Procedure WMInitDialog(VAR
      Msg:TMessage); VIRTUAL wm_First +
      wm_InitDialog;
   END;

{Diálogo del ensamblador}

   O_DlgGetInfoAss =
   OBJECT(O_DlgGetInfoSrv)
      Constructor
      Init(AParent:PWindowsObject;aBlock:P
      O_Block);
      Procedure Inputs(VAR Msg:TMessage);
      VIRTUAL id_First + id_SrvInps;
      Procedure WMInitDialog(VAR
      Msg:TMessage); VIRTUAL wm_First +
      wm_InitDialog;
   END;

{Diálogo de la herramienta de texto}
```

```
O_DlgGetInfoFont =
OBJECT(O_DlgGetInfo)
   Constructor
   Init(AParent:PWindowsObject;aBlock:P
   O_Block);
   Procedure FontBtn(VAR
   Msg:TMessage); VIRTUAL id_First +
   id_Font;
   Procedure Ok(VAR
   Msg:TMessage);VIRTUAL id_First +
   id_Ok;
   Procedure WMInitDialog(VAR
   Msg:TMessage); VIRTUAL wm_First +
   wm_InitDialog;
END;

{Diálogo del generador}

O_DlgGetInfoGen =
OBJECT(O_DlgGetInfo)
   Constructor
   Init(AParent:PWindowsObject;aBlock:P
   O_Block);
   Procedure Ok(VAR
   Msg:TMessage);VIRTUAL id_First +
   id_Ok;
   Procedure WMInitDialog(VAR
   Msg:TMessage); VIRTUAL wm_First +
   wm_InitDialog;
END;

{Diálogo de la cola}

O_DlgGetInfoQueue =
OBJECT(O_DlgGetInfo)
   Constructor
   Init(AParent:PWindowsObject;aBlock:P
   O_Block);
   Procedure Ok(VAR
   Msg:TMessage);VIRTUAL id_First +
   id_Ok;
   Procedure WMInitDialog(VAR
   Msg:TMessage); VIRTUAL wm_First +
   wm_InitDialog;
END;

{Diálogo del terminador}

O_DlgGetInfoTer =
OBJECT(O_DlgGetInfo)
   Constructor
   Init(AParent:PWindowsObject;aBlock:P
   O_Block);
   Procedure Ok(VAR
   Msg:TMessage);VIRTUAL id_First +
   id_Ok;
   Procedure WMInitDialog(VAR
   Msg:TMessage); VIRTUAL wm_First +
   wm_InitDialog;
END;

{Diálogo del divisor}

O_DlgTypeDiv= OBJECT(O_DlgGetInfo)
   PRIVATE
      RuleApp : ^TD_Rule;
      DefByUsr : PChar;
      Id : INTEGER;

   PUBLIC
      Constructor
      Init(AParent:PWindowsObject;VAR
      aRule:TD_Rule;aDefByUsr:PChar;anId
      :INTEGER);
      Procedure Ok(VAR
      Msg:TMessage);VIRTUAL id_First +
      id_Ok;
      Procedure WMInitDialog(VAR
      Msg:TMessage); VIRTUAL wm_First
      + wm_InitDialog;
END;
```

PROCEDURE_GetBlkComTPts
Parámetros

Line,	{Línea a leer}
Attr,	{Atributos leídos}
Expression:PChar	{Expresión obtenida}

Devuelve el atributo y la expresión de una línea definida como:

Atributo:Expresión

PROCEDURE_GetIdsWeights
Parámetros

Line:PChar	{Línea}
Collec:PO_Rule	{Colección}

Devuelve los valores ingresados a una colección, en una línea definida como:

(Id1,W1),(Id2,W2),...

PROCEDURE_GetIdWeight
Parámetros

Elem:PChar	{Elemento}
VAR Id:INTEGER	{Identificador}
VAR W:REAL	{Peso}

Devuelve los valores de identificador y peso de un elemento definido como:

(Identificador,Peso)

PROCEDURE_GetLinkedWith
Parámetros

Line:PChar	{Línea de dónde se lee la información}
VAR Id:INTEGER	{Identificador}

Devuelve el identificador de una línea deinida como:

LINKED WITH Id

FUNCTION_GetPTool
Parámetros

Tool:TD_ToolName	{Tipo de herramienta}
anState:P_State	{Copia del estado del sistema}

Salida
Puntero del bloque creado:PO_Block

Crea un bloque del tipo especificado, dándole una copia del estado de la ventana gráfica.

PROCEDURE_GetTypeCreate
Parámetros

Line:PChar	{Línea de donde se lee la información}
VAR Tool:TD_ToolName	{Tipo de herramienta definido}
VAR IdNum:INTEGER	{Identificador de la herramienta}

Regresa el tipo de herramienta y el identificador, de una línea definida como:

CREATE TypeTool WITH IdNum

13. TOOLBAR

UNIT_ToolBar
Esta unidad tiene por objetivo el desplegar en pantalla la barra de herramientas de la ventana de código y edición gráfica. Sólo estos tipos de barras son las que se expondrán aquí.

```
CONST_Varias
   am_CalcParentClientRect = wm_User + 120;
                              {Mensaje del usuario}

   tbHorizontal   = $01; {Posición horizontal}
   tbLeftVertical = $02; {Posición vertical izquierda}
   tbRightVertical        = $04;
   {Posición vertical derecha}
   DenyRepaint  = 0;     {Uso reservado}
   AllowRepaint  = 1;    {Uso reservado}

TYPE_Varios
   PO_EditTB = ^O_EditTB;
   PO_GraphTB = ^O_GraphTB;

   O_EditTB = OBJECT(TWindow)
      PRIVATE
         ResName : PChar;
         {Nombre del recurso}
         Tools : TCollection;
         {Colección de iconos}
         Capture : PO_Tool;
         {Uso reservado}
         Orientation : WORD;
         {Tipo de desplegado}

      PUBLIC
         Constructor INIT
         Destructor DONE
         Procedure AMCALCPARENTCLIENTRECT
         Function CREATETOOL
         Procedure ENABLETOOL
         Procedure FREERESNAME
         Function GETCLASSNAME
         Function GETORIENTATION
         Procedure GETWINDOWCLASS
         Constructor LOAD
         Procedure NEXTTOOLORIGIN
         Procedure PAINT
         Procedure READRESOURCE
         Procedure SETORIENTATION
         Procedure SETRESNAME
         Procedure STORE
         Procedure SWITCHTO
         Procedure WMLBUTTONDOWN
         Procedure WMLBUTTONUP
         Procedure WMMOUSEMOVE
   END;

   O_GraphTB = OBJECT(TWindow)
      PRIVATE
         AnState : P_State;
         {Estado del sistema}
         ToolsIcon : ARRAY[TD_ToolName] OF HIcon;
         {Iconos de herramientas}
         NoIconsTB : INTEGER;
         {Número de herramientas}

      PUBLIC
         Constructor INIT_
         Destructor DONE_
         Procedure GETR_
         Procedure PAINT_
         Procedure TOOLSELECT_
         Procedure WMLBUTTONDOWN_
   END;

CONSTRUCTOR_Init
Parámetros
   AParent : PWindowsObject
   {Ventana padre}
```

AName : PChar {Nombre del Toolbar}
Orient : WORD {Tipo de desplegado}

Inicializa la barra de herramientas, dando de alta la ventana que la contendrá y leyendo los iconos o bitmaps del archivo de recursos.

DESTRUCTOR_Done
Destruye la ventana y la colección de herramientas.

PROCEDURE_AMCalcParentClientRect
Parámetros
VAR Msg : TMessage

Si existen cambios en la ventana padre, se recalculan las posiciones de las herramientas, dependiendo de la orientación que tenga la barra de herramientas. Esta rutina responde a un mensaje generado por el poseedor, no por un mensaje estándar de Windows.

FUNCTION_CreateTool
Parámetros
Num : INTEGER {Número de herramienta}
Origin : TPoint {Punto de origen}
Command : WORD {Orden a la que responde el botón}
BitmapName : PChar {Nombre del identificador visual}
Salidas
Puntero del botón : PO_Tool;

Asigna el puntero de una nueva herramienta, dado su punto de origen, la imagen que lo define, y sobre todo, el comando al que responde.

PROCEDURE_EnableTool
Parámetros
Command : WORD {Comando del botón}
NewState : BOOLEAN {Nuevo estado}

Le cambia el estado al primer botón que cumple con el comando de búsqueda.

PROCEDURE_FreeResName
Libera la memoria donde se almacena el nombre del archivo de recursos.

FUNCTION_GetClassName
Salidas
'O_EditTB' : PChar

Entrega el nombre de la clase de ventana.

FUNCTION_GetOrientation
Salidas
Orientación : WORD

Entrega el tipo de orientación; es decir, el modo de desplegado de la barra de herramientas, ya sea horizontal o vertical.

PROCEDURE_GetWindowClass
Parámetros
VAR WC : TWndClass {Datos de la clase}

Investiga la clase de la ventana heredada, y asigna al fondo de la ventana un color gris por medio de la brocha LtGray_Brush.

PROCEDURE_NextToolOrigin
Parámetros
Num : INTEGER {Número de herramienta}
VAR Origin : TPoint{Nuevo origen}
P : PO_Tool {Botón de herramienta}

Calcula el punto de origen del siguiente botón, dado el anterior y su ancho; o su altura si el modo de desplegado es vertical.

PROCEDURE_Paint
Parámetros
DC : HDC {Display Context}
VAR PS : TPaintStruct {Datos de pintado}

Pinta el fondo de la ventana, la barra de herramientas y posteriormente botón por botón con el uso de las rutinas de colecciones.

PROCEDURE_ReadResource
Lee el archivos de recursos, para cargar los bitmaps o iconos que identifican visualmente a cada botón de cada una de las herramientas. Aquí es donde se usan las rutinas CREATETOOL, NEXTTOOLORIGIN, para crear las herramientas y asignar posiciones, y una vez que están los datos completos se insertan en la colección de herramientas.

PROCEDURE_SetOrientation
Parámetros
NewOrient : WORD {Nuevo modo de desplegado}

Cambia el modo de desplegado de la barra de herramientas, y se recalculan las posiciones de cada botón.

PROCEDURE_SetResName
Parámetros
NewName : PChar {Nombre del archivo}

Establece el nuevo nombre del recurso que contiene las imágenes que identifican a las herramientas.

PROCEDURE_SwitchTo
Parámetros
NewName : PChar {Nombre del archivo}

Destruye la barra de herramientas actuales, y cambia por otra, asignando el nombre del nuevo archivo de recursos.

PROCEDURE_WMLButtonDown
Paráemtros
VAR Msg : TMessage

Investiga si hubo un clic dentro del área de uno de los botones para ejecutar el comando que le corresponde.

PROCEDURE_WMLButtonUp
Parámetros
VAR Msg : TMessage

Cuando el botón del ratón es soltado, se manda el mensaje a la ventana padre sobre el botón que fue presionado.

PROCEDURE_WMMouseMove
Parámetros
VAR Msg : TMessage

Investiga si durante un clic existe movimiento, si es así, no se ejecuta el comando correspondiente.

CONSTRUCTOR_Init_
Parámetros
AParent : PWindowsObject
{Ventana padre}
TheState : P_State {Estado del sistema}

Crea la ventana que contendrá la barra de herramientas, se crea un barra de recorrido, se cargan todos los iconos necesarios como las imágenes de las herramientas.

DESTRUCTOR_Done_
Destruye el bloque activo por omisión, destruye los iconos, y al final la ventana.

PROCEDURE_GetR_
Parámetros
I : WORD {Número de herramientas}
VAR R : TRect {Área de la barra}
MinusOffs : BOOLEAN {Si se ve el lado izquierdo}

Calcula del área de la barra de herramientas que se despliega en pantalla. El número 40 que aparece aquí es el ancho y alto de cada icono. Aunque en realidad sus dimensiones son de 32 x 32.

PROCEDURE_Paint_
Parámetros
PaintDC : HDC {Display Context}
VAR PaintInfo : TPaintStruct {Datos de pintado}

Pinta las herramientas. Recordar que el botón mide 32 x 32, y aquí se presentan como de 40 x 40, para que tuvieran un margen considerable y no se confundiesen entre sí.
PROCEDURE_ToolSelect_
Parámetros
Tool : TD_ToolName {Nombre del bloque}

Al hacer un clic sobre alguno de los botones, se destruye del bloque por omisión anterior, se crea el nuevo bloque por omisión, y se despliega en estado invertido para señalizar que se encuentra seleccionado.
El bloque por omisión es el bloque que se encuentra activo para ser insertado en el Canvas. Este bloque vive en la variable general de estado, en el campo Block.

PROCEDURE_WMLButtonDown_
Parámetros
VAR Msg : TMessage

Si se realiza el evento de un clic sobre la barra de herramientas, se calcula la posición del bloque al que corresponde el clic. Dependiendo del tipo de herramienta seleccionada, es la acción a tomar. Esta rutina manda llamar a TOOLSELECT_.

PROCEDURE_DeallocateResources
Libera los recursos que fueron cargados desde TypePsi.

14. TYPEPSI

UNIT_TypePSI
Unidad básica de tipos para PSI. En esta unidad se declara la clase de O_Block, junto con las colecciones de identificadores. También se encuentra la clase de O_SimObj; además de algunos diálogos.
Prácticamente todas las unidades necesitan de TypePSI.

TYPE_Varios

P_State = ^E_State;
{Puntero al estado del sistema}

P_RuleItem = ^E_RuleItem;
{Puntero al elemento de una regla de asociación}

PO_Block = ^O_Block;
{Puntero a un bloque}

PO_CollecBlks = ^O_CollecBlks;
{Puntero a una colección de bloques}

PO_Rule = ^O_Rule;
{Puntero a una regla de asociación}

PO_SimObj = ^O_SimObj;
{Puntero a una clase que identifica a las partes en el proceso}

PO_StBar = ^O_StBar;
{Puntero a una barra de estado}

PO_DlgGetAttr = ^O_DlgGetAttr;
{Puntero a un diálogo que captura los atributos de la parte en proceso}

PO_DlgGetInfo = ^O_DlgGetInfo;
{Puntero a un diálogo que obtiene información en general}

PO_DlgMoveGraph= ^O_DlgMoveGraph;
{Puntero a un diálogo de movimiento de bloques}

O_DlgPrn = ^O_DlgPrn;
{Puntero a un diálogo de impresión}

O_DlgSize = ^O_DlgSize;
{Puntero a un diálogo que cambia el tamaño del área de definición}

E_State = RECORD

PenSize,	{Tamaño de la plumilla}
ItemCount,	{Contador de elementos}
Ds,Ds_,Dx_,Dy_,	
L_Arrow,	{Longitud de la flecha}
MinX,	{Mínima X del grafo}
MinY,	{Mínima Y del grafo}
MaxX,	{Máxima X del grafo}
MaxY : INTEGER;	{Máxima Y del grafo}
Change : BOOLEAN;	{Verificador de cambios en el diagrama}
Font : HFont;	{Tipo de letra tomada por omisión}
ToolSel : TD_ToolName;	{Herramienta seleccionada}
Offset,	{Offset del área de edición gráfica}
XYScale : TPoint;	{Tamaño del área de edición gráfica}

```
Block : PO_Block;          {Bloque actualmente utilizado}
SimObj : PO_SimObj;        {Parte en el proceso}
Blocks : PO_CollecBlks;    {Bloques existentes en el proceso}
SelBlocks : PO_Rule;       {Bloques seleccionados}
BezierPen : PO_Pen;        {Plumilla utilizada en Bezier}
Canvas : PWindow;          {Area de edición gráfica}
EditWin : PFileWindow      {Area de edición de código}
END;

{ Esta estructura es la más importante dentro del sistema, pues determina el estado del sistema: Flujo definido, bloques seleccionados, parte simulada, área de edición de código, etcétera. }

E_RuleItem = RECORD
Id : INTEGER;
Weight : REAL;
END;

{Estructura que define un elemento dentro de la colección de identificadores O_Rule}

O_Block= OBJECT(TObject)
DC:HDC;
    {Área de contexto compatible con el CANVAS}
Mode : TD_MvSzMode;
    {Tipo de selección}
IdNum,NApunt,NApOr : INTEGER;
    {Identificador del bloque}
FirstTime,
    {Indicador booleano que determina si es la primera vez que es seleccionado}
Selected,
    {Indicador de selección}
Flag : BOOLEAN;
    {Bandera para múltiples propósitos}
Pos,
    {Posición del bloque en el CANVAS}
WH : TPoint;
    {Ancho y alto de la representación gráfica}
State : P_State;
    {Copia de la variable de estado}
BitMap : PO_UsrBMP;
    {Representación gráfica}
ResBlocks,
    {Colección de identificadores de los recursos utilizados por el bloque}
NxtBlocks,
    {Colección de identificadores de los bloques siguientes}
PrvBlocks : PO_Rule;
    {Colección de identificadores de los bloques anteriores}

  Constructor INIT
  Destructor DONE
  Procedure BUILDFROMCOPY
  Procedure BUILDUNIONS
  Procedure CHANGEIDTO
  Procedure CHECKUNIONS
  Procedure DESELECT
  Procedure GENCODE
  Procedure GENCODELINKS
  Function GETCLASSNAME
  Procedure GETINFOFROMUSER
  Function HASMULTINPS
  Function HASMULTOUTS
  Function ISIN
  Function ISINSIZERGN
```

```
    Function ISLINKABLE
    Function ISLINKABLEIN
    Function ISLINKABLEOUT
    Procedure KEYCHAR
    Procedure LINKWITH
    Procedure LOAD
    Procedure MOUSEDOWN
    Procedure MOUSEMOVE
    Procedure MOUSEUP
    Function NEEDTOBEDELETED
    Procedure NOTIFYCHANGE
    Procedure PAINT
    Function READCOMM
    Function READFROMEDIT
    Procedure SELECT
    Procedure SETCURS
    Procedure STORE
    Function TYPETOOL
END;

O_CollecBlks = OBJECT(TCollection)
    Function CALLUSERFUNC
    Proccedure DEL
    Procedure DELBLKSIN
    Function FIRSTTHATIN
    Function FIRSTNOTUNION
    Function FIRSTUNIONTHAT
    Procedure FOREACHIN
    Procedure FOREACHNOTUNION
    Procedure FOREACHUNION
    Function GETID
    Function GETMAXIDAVAIL
    Function GETMINIDAVAIL
    Function GETUNION
    Function ISALREADY
END;

O_Rule = OBJECT(TCollection)
Ruleltem : P_Ruleltem;
        {Elemento de la colección de
        identificadores}
State : P_State;
        {Copia del estado de la ventana
        gráfica}
Rule : TD_Rule;
        {Regla de asociación, si es que
        existe}
ValUsr : ARRAY[0..50] OF CHAR;
        {Valor de la regla, en caso de ser
        definida por el usuario}

    Constructor INIT
    Destructor DONE
    Procedure DEL
    Procedure DESELECTALL
    Procedure FREEITEM
    Function GETITEM
    Function GETFIRSTLINKABLE
    Function GETID
    Function GETMAXX
    Function GETMAXY
    Function GETMINX
    Function GETMINY
    Function GETSECONDLINKABLE
    Function GETWEIGHT
    Function HOWMANYLINKABLE
    Procedure INS
    Function ISALREADY
    Procedure LOAD
    Procedure PUTITEM
    Procedure SETWEIGHT
    Procedure STORE
END;

O_SimObj = OBJECT(TObject)
Types,
        {Tipos definidos por el usuario para
        los atributos de la parte}
Attributes: PStrCollection;
        {Atributos definidos por el usuario
        par a los atributos de la parte}
State : P_State;
        {Copia del estado de la ventana de
        definición gráfica}
```

```
      Constructor INIT
      Destructor DONE
      Procedure GENCODE
      Procedure GETINFOFROMUSER
      Procedure LOAD
      Function READFROMEDIT
      Procedure STORE
   END;

   O_StBar = OBJECT(TWindow)
      PRIVATE
         Height : INTEGER;
            {Altura de la barra de estado}
         HintText : ARRAY[0..150] OF CHAR;
            {Texto que se muestra en la
            barra}
         Font : HFont;
            {Tipo de letra con que se
            muestra}
         DefaultText : PChar;
            {Texto por omisión}

      PUBLIC
         Constructor INIT
         Destructor DONE
         Function GETCLASSNAME
         Function GETHEIGHT
         Procedure GETWINDOWCLASS
         Procedure PAINT
         Procedure SETTEXT
         Procedure WMSETTEXT
END;

O_DlgGetAttr = OBJECT( TDialog )
   Blk :PO_SimObj;
         {Referencia al bloque que representa
         la parte en proceso}

      Constructor INIT
      Procedure OK
      Procedure WMINITDIALOG
END;

O_DlgGetInfo = OBJECT( TDialog )
   Blk:PO_Block;
         {Referencia de bloque al que
         deseamos ingresar sus atributos}

      Constructor INIT
      Procedure CHGBMP
      Procedure SETRESOURCES
      Procedure SETSTR
END;

O_DlgMoveGraph = OBJECT(TDialog)
   Mode,
         {Relativo o absoluto}
   X,
         {Valor de X o Dx}
   Y : PInteger;
         {Valor de Y o Dy}

      Constructor INIT
      Procedure OK
      Procedure WMINITDIALOG
END;

O_DlgPrn = OBJECT( TDialog )
   PrtType : ^TD_PrtType;
         {Tipo de impresión}

      Constructor INIT
      Procedure OK
      Procedure WMINITDIALOG
END;

O_DlgSize = OBJECT( TDialog )
   State : P_State;
         {Referencia al estado de la ventana
         gráfica}

      Constructor INIT
      Procedure OK
      Procedure WMINITDIALOG
END;
```

Function CHECKOVERLAP
Procedure GETRESOURCES
Procedure SHOWBLKST
Procedure SHOWSTGRAPH
Function VALTYPE

CONST_Varias
PStBar : PO_StBar=NIL;
{Puntero a la barra de estado del programa}
ShowIds : BOOLEAN=True;
{Indicador de estado del despliege de los identificadores}
Help : BOOLEAN=False;
{Inidicador para Shift+F1 en la ayuda}
VAR
hMenuInit, hMenuGraf, hMenuEdit : HMenu;
{Variables de cada menú}
Printer : PPrinter;
{Impresora}
WhitePen,DarkGrayPen,BlackPen : HPen;
{Plumillas disponibles para el programa}
GrayBrush, GrayingBrush : HBrush;
{Mapas para rellenados disponibles para el programa}
HelpCursor : HCursor;
{Cursor de ayuda}

PROCEDURE_AllocateResources
En este procedimiento se cargan las plumillas y los mapas para rellenado del programa.

CONSTRUCTOR_Init
Un constructor, en cualquiera de sus aplicaciones, se encarga de inicializar los atributos que contiene el objeto.
En el caso de un bloque se construye en base a una copia del estado de la ventana gráfica, y a un mapa de bits.
En el caso de una colección de identificadores (*O_Rule*) se construye a partir de un límite, un delta y una copia del estado del sistema.

En el caso de la clase *O_SimObj*, lo único que se necesita es la copia del estado del sistema.
En el caso de la barra de estado, se construye con la ventana padre.
Y finalmente, en el caso de un diálogo, se construye a partir del puntero al bloque correspondiente; además de la ventana padre.

Ver también:
DONE

DESTRUCTOR_Done
Un destructor, en cualquiera de sus aplicaciones, se encarga de destruir los elementos creados en el constructor.
El destructor siempre es VIRTUAL.

Ver también:
INIT

PROCEDURE_BuildFromCopy
Parámetros:

VAR aStream :TStream	{Lugar de donde se leen los atributos}
aDeltaIds : INTEGER	{Incremento a los identificadores leídos}
aDx,	{Coordenada en X}
aDy,	{Coordenada en Y}

ModeInc : INTEGER {Modo de incrementar: absoluto o relativo}

Rutina que se encarga de cargar la información de un bloque que se encuentra en algún archivo.
Esto se utiliza para la opción de Cortar, Copiar y pegar.

Ver también:
LOAD
STORE

PROCEDURE_BuildUnions
En *O_Block* está declarado como un método abstracto.
Se encarga de construir las uniones declaradas en los bloques siguientes y anteriores (*NxtBlocks*, *PrevBlocks*).
Este servicio será declarado por clases hijas.
Por supuesto es un servicio VIRTUAL.

PROCEDURE_ChangeIdTo
Parámetros
NewId:INTEGER {Nuevo identificador}

Servicio que cambia el identificador del bloque.
Notifica a los demás bloques de su cambio.
VIRTUAL.

PROCEDURE_CheckUnions
Servicio que se encarga de revisar las uniones que tiene el bloque.
En caso de no ser válidas se destruirán y se construirán nuevos flujos de acuerdo con los estándares establecidos.
Este método se utiliza al completar los flujos. Para mayor información revisar el manual de usuario, en el apartado de Completar flujos dentro del menú de opciones.
Este servicio es abstracto al nivel de *O_Block*, y será redefinido por clases hijas.VIRTUAL.

PROCEDURE_Deselect
Deselección del bloque. Es decir,

Selected:=False

Además, se borra de la colección de bloques seleccionados dentro de la variable de estado de la ventana gráfica (*E_State.SelBlocks*).

Este servicio es VIRTUAL.

PROCEDURE_GenCode
Parámetros
EditWnd : PFileWindow {Ventana en la que se generará el código}

Servicio que se encarga de generar el código correspondiente al bloque (normalmente sólo los atributos del bloque).
Este servicio es VIRTUAL, por lo que cada clase hija definirá el código que genera.

Ver también:
GENCODELINKS

PROCEDURE_GenCodeLinks
Parámetros
EditWnd : PFileWindow; {Ventana en la que se genera el código}

Genera el código de los enlaces para un bloque dado.

Este servicio es VIRTUAL, por lo que las clases hijas definirán la forma de escribir sus enlaces en el código.

Ver también:
GENCODE

FUNCTION_GetClassName
Salida
Nombre de la clase: PChar

Servicio VIRTUAL, por lo que cada clase hija debe definir el nombre de su generación.

PROCEDURE_GetInfoFromUser
Servicio que se encarga de pedir los atributos de la clase.
Este servicio es VIRTUAL, por lo que cada clase hija define que tipo de diálogo se manda llamar para capturar la información correspondiente.

FUNCTION_HasMultInps
Salida
Si tiene mútliples entradas :BOOLEAN

HasMultInps:= NxtBlocks^.Count > 1

Ver también:
HASMULTOUTS

FUNCTION_HasMultOuts
Salida
Si tiene múltiples salidas :BOOLEAN

HasMultOuts:= PrvBlocks^.Count > 1

Ver también:
HASMULTINPS

FUNCTION_IsIn
Parámetros
X,Y : INTEGER {Coordenadas de un punto en coordenadas relativas}
Salidas
Si está dentro :BOOLEAN

Función que determina si un punto se encuentra dentro del área de definición del bloque (es decir, dentro de la imagen).
Es VIRTUAL.

Ver también:
ISINSIZERGN

FUNCTION_IsInSizeRgn
Parámetros
X,Y : INTEGER {Coordenadas relativas del punto}
VAR ModeBy:TD_MvSzMode {Modo que corresponde a la posición del punto}
Salida
Si está dentro de la región para cambiar el tamaño: BOOLEAN

Entrega si un punto se encuentra dentro de la región para cambiar el tamaño a la representación gráfica.

Ver también:
ISINSIZERGN

FUNCTION_IsLinkable
Salida
Si es enlazable:BOOLEAN

Declara si la clase puede ser enlazada por otros bloques.

FUNCTION_IsLinkableIn
Salida
Si es enlazable:BOOLEAN

Ver también:
ISLINKABLEOUT

FUNCTION_IsLinkableOut
Salida
Si el bloque puede enlazar :BOOLEAN;

Ver también:
ISLINKABLEIN

PROCEDURE_KeyChar
Parámetros

Key,	{Tecla oprimida}
Count,	{Contador}
lParamHi: WORD	{Parámetro longint de Msg};

Esta rutina es llamada por *Canvas*, cuando se oprime una tecla.
Si la tecla es *ENTER*, manda llamar *GetInfoFromUser*.
Es VIRTUAL (por lo que cada bloque puede definir su comportamiento ante ciertas teclas).

PROCEDURE_LinkWith
Parámetros

ABlock:PO_Block	{Bloque con el que se desea enlazar}

Se toma como bloque final el *ABlock* y como bloque inicial el actual.
El método se encarga de agregar en las listas correspondientes de los objetos, los identificadores de los bloques.

PROCEDURE_Load
Parámetros

VAR S:TStream	{De donde se lee la información}

Se manda guardar la información correspondiente a la clase. En el caso de un bloque, se guardarán los atributos que los definen; en el caso de una colección, se guardarán el número de elementos y los elementos que la conforman.

Es VIRTUAL (Cada bloque o colección define que cantidad de información se manda llamar).

Ver también:
STORE

PROCEDURE_MouseDown
Parámetros

AWindow: HWnd	{La ventana donde ocurrió el evento}
X, Y: INTEGER	{Coordenadas relativas del ratón}

Este método se encarga de definir el comportamiento de un bloque al oprimir el botón izquierdo del ratón. Normalmente si es la primera vez (atributo *FirstTime*) se definirá la posición del bloque en donde se encuentra el ratón. En caso contrario se procederá a analizar en que lugar se encuentra, para determinar si se moverá el

bloque o se cambiará de tamaño (en X,Y ó XY).
Este servicio es virtual, por lo que cada bloque decide como comportarse ante este mensaje (por ejemplo, el comportamiento en una unión es totalmente distinto).

Ver también:
MOUSEUP
MOUSEMOVE

PROCEDURE_MouseMove
Parámetros
X, Y: INTEGER; {Coordenadas de la nueva posición del ratón}

Este servicio es mandado llamar por Canvas cuando el usuario se mueve en el área de edición gráfica.
En este método se definen los comportamientos, ya sean de cambio de posición o de cambio de tamaño.
Es VIRTUAL.

Ver también:
MOUSEUP
MOUSEDOWN

PROCEDURE_MouseUp
Servicio mandada llamar por Canvas cuando el usuario no está presionando el ratón. Este evento permite determinar cuando una acción está finalizada (mover, cambiar de tamaño, etcétera).
Es VIRTUAL.

Ver también:
MOUSEMOVE
MOUSEDOWN

PROCEDURE_NotifyChange
Parámetros
IdOld, {Identificador anterior}
IdNew:INTEGER {Identificador nuevo}

Servicio de notificación cuando existe un cambio en algún identificador. Se revisan todas las colecciones de identificadores para cambiar el identificador anterior por el identificador nuevo.
Este servicio es utilizado principalmente por O_Renum dentro de la unidad de verificación.

Ver también:
CHANGEIDTO

FUNCTION_NeedToBeDeleted
Parámetros
IdDel:INTEGER {Revisa si es necesario borrar este bloque debido a la desaparición de IdDel}
Salida
Si es necesario que se borre:BOOLEAN

Cuando se borra algún bloque se notifica a los demás si es necesario que otros bloques también desaparezcan. Esto es principalmente utilizado cuando se borra alguna conexión.
Es VIRTUAL.

PROCEDURE_Paint
Parámetros
PaintDC:HDC {Contexto de pintado}
PaintInfo:TPaintStruct {Información del pintado}

Servicio que se manda llamar desde Canvas para que se pinte el bloque en un contexto determinado. Para mayor información sobre la estructura *TPaintStruct* consulte el manual de referencia de Windows.
Es VIRTUAL

FUNCTION_ReadComm
Parámetros

aLine:PChar	{Línea actualmente leída}
VAR NumLine:INTEGER	{Número de línea}
Edit:PEdit	{Caja de edición de donde se está leyendo el código}

Esta rutina se encarga de leer los parámetros correspondientes al tipo de bloque. Se leen las palabras reservadas; y en caso de existir algún comando no conocido, se ignora.

En el caso de un servidor se buscará leer los siguientes parámetros (además de los establecidos por bloque):

DISTRIBUTION=x
ADD_DELAY=x
COST_PER_TIME=x
COST_PER_OPER=x
USE RESOURCES=x,y,...
LINKED WITH x
BEING LINKED BY x WITH:

La lectura de comandos se realiza hasta encontrar un *END;* que finaliza la definición de un bloque.

Similarmente para un generador los atributos son (además de los heredados):

DISTRIBUTION=x
INIT_TIME=x
GEN_GROUPS=x
MAX_NUMBER=x
LINKED WITH x

Para una referencia de comandos consulte el Manual de usuario en el apartado de metalenguaje.
Es VIRTUAL.

Ver también:
READFROMEDIT

FUNCTION_ReadFromEdit
Parámetros

aWnd:PWindowsObject	{ Ventana de donde se carga la información }
Edit:PEdit	{ Caja de edición de código}
VAR NumLine:INTEGER	{ Número de línea leído }

Salidas
Si no existió error:BOOLEAN;

Este servicio es llamado para cargar los atributos de cada bloque. Aquí se tiene un bucle de lectura, mandando llamar *ReadComm* hasta que se encuentra un *END;*.

Ver también:
READCOMM

PROCEDURE_Select
Selección del bloque correspondiente. Normalmente se activa la bandera de *Selected* y se ingresa el identificador del bloque a la lista de bloques seleccionados en el sistema (*E_State.SelBlocks*).
Es VIRTUAL.

Ver también:
DESELECT

PROCEDURE_SetCurs
Parámetros
X,Y:INTEGER {Posición del ratón}

Selección del cursor de acuerdo a la posición del ratón.
Es VIRTUAL.

PROCEDURE_Store
Parámetros
VAR S:TStream {Lugar donde se guarda la información del bloque}

Guarda la información del bloque en un STREAM definido.

En el caso de un servidor, se graba la siguiente información (además de la heredada):

```
BEGIN
  INHERITED Store(S);
  S.StrWrite(Distribution);
  S.StrWrite(CostPerTime);
  S.StrWrite(CostPerOper);
  S.StrWrite(AddDelay)
END; { Store }
```

En estas líneas se graba la información de un servidor en un STREAM dado.
Para una referencia completa de formatos de grabación, consulte el apartado de grabación.
Es VIRTUAL.

Ver también:
LOAD

FUNCTION_TypeTool
Salida
Tipo de bloque de la clase:TD_ToolName;

Es VIRTUAL.

FUNCTION_GetHeight
Servicio que obtiene la altura de la barra de estado.

PROCEDURE_GetWindowClass
Parámetros:
VAR WndClass: TWndClass {Atributos especiales de la clase}

Se cambia el fondo a gris.
Para mayor información consulte el manual de referencia de Windows.

PROCEDURE_SetText
Parámetros
Txt:PChar {Texto que se pondrá en la barra de estado}

Copia el texto a la barra de estado.

Ver también:
WMSETTEXT

PROCEDURE_WMSetText
Responde al mensaje wm_SetText de Windows.

Ver también:
SETTEXT

PROCEDURE_Ok
Parámetros

VAR Msg:TMessage {Mensaje mandado por *Windows*}

Servicio que se manda ejecutar cuando el usuario oprime el botón de "Aceptar". Normalmente es en este momento cuando se copia la información capturada al bloque o estructura correspondiente.
Es VIRTUAL.

Ver también:
WMINITDIALOG

PROCEDURE_WMInitDialog
Parámetros:
VAR Msg:TMessage {Mensaje mandado por *Windows*}

Servicio que se manda ejecutar antes de desplegar el diálogo. Normalmente es en este servicio donde se inicializa la información de las cajas de edición, con la información que contiene el bloque o la estructura.
Es VIRTUAL.

Ver también:
OK

PROCEDURE_ChgBMP
Parámetros
VAR Msg:TMessage {Mensaje mandado por *Windows*}

Servicio que se manda ejecutar cuando el usuario presiona el botón de *mapa de bits*, en este momento se manda desplegar un diálogo de *O_DlgPreview* (se encuentra en la unidad de *PREVIEW*), que muestra los archivos *BMP* en una ventana localizada en la parte derecha de la caja del diálogo.

Ver también:
OK
WMINITDIALOG

PROCEDURE_SetResources
Parámetros
VAR Msg:TMessage {Mensajse de *Windows*}

Servicio que se manda ejecutar cuando el usuario presiona el botón de *Recursos*.

Ver también:
OK
WMINITDIALOG

PROCEDURE_SetStr
Parámetros
id:INTEGER {Identificador de la caja de edición}
VAR Str_:PChar {Variable en donde se copiará la información}

Servicio que copia los contenidos de una caja de edición a una variable *Str_*. La rutina revisa si *Str_* es distinto de *NIL*. En caso de serlo, lo manda a *StrDispose_* ; posteriormente le crea el espacio suficiente mediante *GetMem* y copia el contenido de la caja de edición.

FUNCTION_CheckOverlap
Parámetros
P1, {Bloque 1}
P2: PO_Block {Bloque 2}
Salidas
Si hubo traslape: BOOLEAN;

Revisa si dos bloques se encuentran traslapados.

PROCEDURE_GetResources
Parámetros

Line:PChar	{Línea de donde se leerán los identificadores de recursos}
Collec:PO_Rule	{Colección de recursos en donde se ingresarán los id's}

Rutina que ingresa en la colección todos los identificadores separados por coma.

PROCEDURE_ShowBlkSt
Parámetros

Blk:PO_Block	{Bloque del que se despliega la información}

Rutina que muestra en la barra de estado información sobre el bloque.

PROCEDURE_ShowStGraph
Parámetros

State:P_State	{Copia del estado de la ventana gráfica}

Muestra : *ShowBlkSt(State->Block)*

FUNCTION_ValType
Parámetros

TypeTool_: TD_ToolName	{Tipo de bloque}

Salida
Si está dentro de los bloques válidos: BOOLEAN

Determina si un bloque no es de tipo Unión, Texto, O_BMP.

FUNCTION_CallUserFunc
Parámetros

Action:Pointer	{Puntero al procedimiento interno}
NBytes:Integer	{Número de bytes a partir de la tabla de punteros}

Salida
Valor regresado por *Action* :BOOLEAN;

Esta rutina ejecuta un procedimiento (*Action*) determinado, mandando como parámetro el *puntero* que se encuentra en la posición: *NumBytes DIV SizeOf(Pointer)*. El procedimiento cumple con los mismos requisitos de uno utilizado en *ForEach* (Ver manual de *ObjectWindows*).
Recuerde la importancia de que este procedimiento sea interno al servicio de una clase que tenga como raíz *TObject*; además no se debe mandar llamar más de 3 veces en forma anidada (debe declararse como *FAR*).

PROCEDURE_Del
Parámetros

IdBlk : INTEGER	{Identificador a borrar}

Borra el bloque *(O_CollecBlks)* o el elemento *(O_Rule)* con el identificador especificado de la colección.

Ver también:
DELBLKSIN

PROCEDURE_DelBikeIn
Parámetros
CollecIds:PO_Rule {Colección de identificadores}

Borra todos los bloques con identificadores iguales a los que tienen los elementos de la colección de identificadores (*CollecIds*).

Ver también:
DEL

FUNCTION_FirstThatIn
Parámetros
CollecIds:PO_Rule {Colección de identificadores}
Action:Pointer {*Puntero* de la función a ejecutar con cada uno de ellos}
Salida
Bloque que cumple: PO_Block

Regresa el primer bloque, dentro de *CollecIds*, que cumple con la condición (*Action*) establecida.
Se maneja igual que *FirstThat* (consulte el manual de *ObjectWindows*).

Ver también:
FIRSTNOTUNION
FIRSTUNIONTHAT

FUNCTION_FirstNotUnion
Parámetros
Action:POINTER; {*Puntero* de la función a ejecutar con cada uno de los elementos}
Salida
Bloque que cumple: PO_Block

Regresa el primer bloque, con tipo distinto de *O_Union*, que cumple con la condición (*Action*) establecida.
Se maneja igual que *FirstThat* (consulte el manual de *ObjectWindows*).

Ver también:
FIRSTTHATIN
FIRSTUNIONTHAT

FUNCTION_FirstUnionThat
Parámetros
Action:POINTER; {*Puntero* de la función a ejecutar con cada uno de los elementos}
Salida
Unión que cumple con la condición: PO_Union

Regresa el primer bloque de unión, que cumple con la condición (*Action*) establecida.
Se maneja igual que *FirstThat* (consulte el manual de *ObjectWindows*).

Ver también:
FIRSTTHATIN
FIRSTNOTUNION

PROCEDURE_ForEachIn
Parámetros:
CollecIds:PO_rule {Colección de identificadores}
Action:Pointer {*Puntero* del procedimiento a ejecutar con cada uno de los elementos}

Manda llamar el procedimiento establecido con cada uno de los elementos dentro de la colección (*CollecIds*).
Se maneja igual que *ForEach* (consulte el manual de *ObjectWindows*).

Ver también:
FOREACHNOTUNION
FOREACHUNION

PROCEDURE_ForEachNotUnion
Parámetros
Action:Pointer {*Puntero del procedimiento a ejecutar*}

Manda llamar el procedimiento con todos los bloques distintos a *UnionTool*.
Se maneja igual que *ForEach* (consulte el manual de *ObjectWindows*).

Ver también:
FOREACHIN
FOREACHUNION

PROCEDURE_ForEachUnion
Parámetros
Action:Pointer {*Puntero del procedimiento a ejecutar*}

Manda llamar el procedimiento con todos los bloques iguales a *UnionTool*.
Se maneja igual que *ForEach* (consulte el manual de *ObjectWindows*).

Ver también:
FOREACHIN
FOREACHNOTUNION

FUNCTION_GetId
Parámetros
Id:INTEGER {Identificador buscado}
Salida
Bloque que coresponde al identificador: PO_Block

Obtiene el bloque con el identificador específico. Si no existe regresa NIL.

Ver también:
GETMAXIDAVAIL
GETMINIDAVAIL

FUNCTION_GetMaxIdAvail
Salida
Identificador máximo:INTEGER

Regresa el identificador más grande dentro de la colección.

Ver también:
GETID
GETMINIDAVAIL

FUNCTION_GetMinIdAvail
Salida
Identificador mínimo posible:INTEGER

Regresa el identificador mínimo utilizable.

Ver también:
GETID
GETMAXIDAVAIL

FUNCTION_GetUnion
Parámetros
InId {Identificador del bloque inicial}
EndId:INTEGER {Identificador del bloque final}

Salidas
El bloque de unión :PO_Union;

Regresa el bloque de unión correspondiente a los identificadores: *InIId,EndId*.
En caso de no encontrar alguno que cumpla, regresa *NIL*.

FUNCTION_IsAlready
Parámetros
Id:INTEGER {Identificador del bloque}
Salida
Si existe previamente:BOOLEAN;

Regresa si existe previamente el identificador del bloque especificado.

Ver también:
GETID
GETMAXIDAVAIL
GETMINIDAVAIL

PROCEDURE_DeselectAll
Deselecciona todos los bloque que se encuentran dentro de la colección. Ejecuta el servicio *Deselect* de los bloques.

PROCEDURE_FreeItem
Parámetros
Item:POINTER {Elemento a borra}

Borra el elemento especificado. (Consulte el manual de *ObjectWindows*)

FUNCTION_GetItem
Parámetros
VAR S:TStream:POINTER
{Especifica el lugar de donde se carga el elemento}

(Consulte el manual de *ObjectWindows*)

FUNCTION_GetFirstLinkable
Salida
El primer bloque enlazable:PO_Block;

Regresa el primer bloque que responde a *IsLinkable* verdadero.

Ver también:
GETSECONDLINKABLE
GETID
HOWMANYLINKABLE

FUNCTION_GetMaxX
Salida
La X mayor de los bloques en la colección:INTEGER
Ver también:
GETMAXY
GETMINX
GETMINY

FUNCTION_GetMaxY
Salida
La Y mayor de los bloques en la colección :INTEGER
Ver también:
GETMAXX
GETMINX
GETMINY

FUNCTION_GetMinX
Salida
La X menor de los bloques en la colección :INTEGER
Ver también:
GETMAXX
GETMINY
GETMAXY

FUNCTION_GetMinY
Salida
La Y menor de los bloques en la colección:INTEGER

Ver también:
GETMAXX
GETMINX
GETMAXY

FUNCTION_GetSecondLinkable
Salida
Segundo bloque enlazable :PO_Block

Regresa el segundo bloque que responde a *IsLinkable* verdadero.

Ver también:
GETID
GETFIRSTLINKABLE
HOWMANYLINKABLE

FUNCTION_GetWeight
Parámetros
I:INTEGER {Identificador del bloque}
Salida
Peso :Real

Regresa el peso de la regla para el bloque I.

Ver también:
SETWEIGHT
GETID
INS

FUNCTION_HowManyLinkable
Salida
Cuantos elementos enlazables existen :INTEGER

Regresa el número de bloques que responden a *IsLinkable* verdadero.

Ver también:
GETFIRSTLINKABLE
GETSECONDLINKABLE
GETID

PROCEDURE_Ins
Parámetros
Id:INTEGER {Identificador a insertar en la colección}
Ingresa un elemento con identificador igual a Id a la colección.
Ver también:
DEL
DESELECTALL
FREEITEM

PROCEDURE_PutItem
Parámetros
VAR S:TStream {Lugar donde se guarda la información}
Item:POINTER {Elemento}

(Consultar el manual de *ObjectWindows*)

Ver también:
STORE
LOAD
INS

PROCEDURE_SetWeight
Parámetros
Id:INTEGER {Elemento con identificador I}
W:Real {Peso deseado}
Pone un peso al elemento con el identificador dado.
Ver también:
STORE
LOAD
INS

15. USRBMP

UNIT_UsrBMP
Unidad que maneja los archivos de mapas de bits (con formato *BMP*).

CONST_Constantes
OneIO = 32768
{Tamaño del registro de lectura/Escritura}
BMType = $4D42
{Bandera de identificación del formato BMP}

TYPE_Tipos
PBool = ^BOOLEAN;

PtrRec = RECORD
Lo, Hi: WORD
END;

IOFunction = Function(FP: INTEGER; Buf: PChar; Size: INTEGER): WORD;

PO_UsrBMP = ^O_UsrBMP;
O_UsrBMP = OBJECT(TObject)
PRIVATE
BMP2 : HBitmap;
{Variable para cambiar el tamaño al BMP}
OrgWidth,
{Ancho original}
OrgHeight : INTEGER;
{Altura original}
Offset,
{Número de *bytes* para comenzar la lectura del BMP}
Mode : LONGINT;
{Modo de copiado al contexto de pintado}
CopyName : PChar;
{Copia del nombre}
IsPrinting : PBool;
{Copia de impresión}

PUBLIC
BitmapHandle : HBitmap;
{Manejador del BMP}
FileName : ARRAY[0..fsPathName] OF CHAR;
{Nombre del archivo}
IsExtFile : BOOLEAN;
{Indicador de clase de BMP. interno o externo}
PixelHeight,
{Altura en *pixels*}
PixelWidth : INTEGER;
{Ancho en *pixels*}

Constructor INIT
Constructor INITBMP
Destructor DONE
Procedure COPYFROM
Procedure DRAWUSRBMP
Function HUGEIO
Procedure LOAD
Function LOADBITMAPFILE
Function OPENDIB
Procedure RELOAD
Procedure RESIZE
Procedure SETBITMAP
Procedure SETFILE
Procedure STORE
END;

PROCEDURE_AHIncr
Rutina de lectura de segmentos.
Consulte manual de Windows.

CONSTRUCTOR_Init
Parámetros
Name:PChar {Nombre del archivo}

anIsPrinting: PBool	{Indicador de impresión}

Inicializa el BMP como un archivo externo (es decir, no pertenece al archivo de recursos del programa).

Ver también:
INITBMP

CONSTRUCTOR_InitBMP
Parámetros

ABmp:HBitmap	{BMP}
Name:PChar	{Nombre del BMP, para futura recarga}
anIsPrinting: PBool	{Variable de impresión}

Inicializa el BMP con el manejador establecido.

Ver también:
INIT

DESTRUCTOR_Done
Destruye el manejador del BMP.

PROCEDURE_CopyFrom
Parámetros

aBmpHandle:HBitmap	{BMP a copiar}

Se encarga de copiar el BMP establecido.

Ver también:
RELOAD

PROCEDURE_DrawUsrBMP
Parámetros

DC: HDC	{Contexto de pintado}
X,Y:INTEGER	{Coordenadas}
WhatWindow:HDC	{Ventana}

Dibuja el BMP en las coordenadas establecidas.

FUNCTION_HugeIO
Parámetros

IOFunc: IOFunction	{Función de Lectura/Escritura}
F: INTEGER;	{Archivo}
P: POINTER;	{Puntero donde se Lee/Escribe}
Size: Longint	{Tamaño en bytes de la operación}

Salidas
Si fue posible el número de bytes de la operación: WORD;

Manejador de lectura de segmentos de gran tamaño.

PROCEDURE_Load
Parámetros

VAR S:TStream	{Lugar de donde se carga la información}

Guarda la información correspondiente al BMP en el archivo indicado.

Ver también:
RELOAD
STORE

FUNCTION_LoadBitmapFile
Parámetros

WhatWindow:HDC	{Ventana padre}

Salida

Si se pudo leer el BMP: BOOLEAN

Lectura del archivo BMP.

Ver también:
OPENDIB

FUNCTION_OpenDIB
Parámetros
F: INTEGER {Archivo}
Salida
Si fue posible leer la información de BMP: BOOLEAN

Lee la información del archivo.

Ver también:
LOADBITMAPFILE

PROCEDURE_Reload
Parámetros
WhatWindow:HDC {Ventana padre}

Vuelve a cargar el BMP (en caso de ser posible).

Ver también:
LOAD

PROCEDURE_Resize
Parámetros
aNewWidth, {Nuevo ancho}
aNewHeight:WORD {Nueva altura}

Dimensiona el BMP a las nuevas especificaciones.

Ver también:
LOAD
RELOAD

PROCEDURE_SetBitmap
Parámetros
aBmpHandle:HBitmap {BMP}

Establece el BMP.

PROCEDURE_SetFile
Parámetros
Path:PChar {Nombre del archivo}

Se copia el nombre del archivo al atributo correspondiente.

PROCEDURE_Store
Parámetros
VAR S:TStream {Lugar donde se guarda la información del BMP}

Se guarda la información del BMP en el lugar especificado.

Ver también:
LOAD

16. VERIFY

UNIT_Verify
Unidad de verificación de bloques.

TYPE_O_Verify
{Esta clase se encarga de revisar la información definida por el usuario. Se revisan tres niveles: validez de identificadores, validez de pesos en las reglas de asociación, y validez en conexiones. Existen varios modos de revisión:

a) Revisión completa. En este tipo se revisa y se completan los flujos que así lo necesiten.
b) Revisión parcial. En este tipo no se completan los flujos.
c) Revisión de Generadores-Terminadores (Begin-Ends). En este tipo se buscan que existan terminadores y generadores en cada línea definida}

```
PO_Verify= ^O_Verify;
O_Verify = OBJECT(TObject)
   PRIVATE
      State :P_State;
         {Estado a revisar}
      Parent :PWindowsObject;
         {Ventana padre}
      VerifyBE,
         {Verificación de generadores y
         terminadores}
      FullVerif:BOOLEAN;
         {Verificación completa}

      Procedure CHECKIDS
      Procedure CHECKWEIGHTS
      Procedure DELINVALID
      Procedure VERIFYEXIST

   PUBLIC
      Constructor INIT
      Procedure RUN
   END;
```

TYPE_O_Renum
{Clase utilizada para la reenumeración de los bloques}

```
   PO_Renum=^O_Renum;
   O_Renum = OBJECT(TObject)
      PRIVATE
         anState:P_State;
            {Copia del estado de la ventana
            gráfica}

      PUBLIC
         Constructor INIT
         Procedure RUNRENUM
   END;
```

CONSTRUCTOR_Init
Parámetros
aParent:PWindowsObject {Ventana padre}
anState:P_State {Copia del estado de la ventana gráfica}
FullVerify:BOOLEAN {Verificación completa}
VerifyBEnd:BOOLEAN {Verificación de generadores-terminadores por línea}

Inicializa las variables correspondientes.

PROCEDURE_CheckIds
Parámetros
Blk:PO_Block {Bloque a revisar}

Se revisa la validez de los identificadores declarados en el bloque correspondiente.

Ver también:
CHECKWEIGHTS
DELINVALID

PROCEDURE_CheckWeights
Parámetros
Blk:PO_Block {Bloque a revisar}

Se revisa la validez de los pesos declarados en el bloque.

Ver también:
CHECKIDS
DELINVALID

PROCEDURE_DelInvalid
Parámetros
Collec:PO_Rule {Borrar los elementos no válidos}

Ver también:
CHECKIDS
CHECKWEIGHTS

PROCEDURE_Run
Se ejecuta la revisión de los bloques definidos en State.

PROCEDURE_VerifyExist
Parámetros
SetMustExist:TSetTooIN {Conjunto de tipo de bloques}

Se revisa la existencia de los bloques definidos en cada una de las líneas establecidas.

PROCEDURE_RunRenum
Se ejecuta la reenumeración de los bloques.

Ver también:
RENUM

PROCEDURE_Renum
Parámetros
anState:P_State {Estado de la ventana gráfica}

En este procedimiento se revisa el estado de la ventana gráfica establecido. Se crea un objeto de tipo O_Verify y se ejecuta la revisión.

RUNRENUM

17. VIEWBMP

UNIT_ViewBMP
Esta unidad tiene como finalidad crear una ventana que presente un bitmap en la opción de Acerca... (About) del menú de Ayuda (Help) de cualquier aplicación.
Para cerrar esta ventana basta con realizar in clic dentro del área cliente de la misma o por los medios comunes del menú de sistema.

CONST_Varias
AboutActive : BOOLEAN = False;

Esta constante tiene como finalidad dejar la firma de esta ventana como ya creada, pues podrían crearse demasiadas ventanas sin un orden y no cumpliendo con la finalidad para la que fue diseñada.

```
TYPE_O_ViewBMP
  PO_ViewBMP = ^O_ViewBMP;
  O_ViewBMP = OBJECT(TWindow)
    PRIVATE
      BMPShowed : PO_UsrBMP;
      {Objeto que maneja un bitmap}
    PUBLIC
      Constructor INIT
      Destructor DONE
      Function CANCLOSE
      Procedure PAINT
      Procedure WMLBUTTONDOWN
  END;
```

{Objeto de presentación de bitmap para Acerca... (About)}

CONSTRUCTOR_Init
Parámetros
AParent : PWindowsObject {Padre al que reportar}
Title, {Título de la ventana}
PathBMP : PChar {Ruta de acceso al archivo BMP}

Inicializa la ventana, al igual que el objeto O_UsrBMP por medio de PathBMP.

DESTRUCTOR_Done
Destruye la ventana, después de haber destruido el objeto O_UsrBMP.

FUNCTION_CanClose
Salida
Si se puede cerrar : BOOLEAN

Función que tiene por objetivo reasignar a la constante AboutActive el valor de False, para que posteriormente volver a crear la ventana.

PROCEDURE_Paint
Parámetros
PaintDC : HDC {Display Context de la ventana}
VAR PaintInfo : TPaintStruct
{Información de pintado}

Procedimiento que llama a la rutina de pintado del objeto padre, y después pinta el bitmap en el área cliente de la ventana, centrado, y con un margen de 10 Pixels en todas direcciones.

PROCEDURE_WMLButtonDown
Parámetros
VAR Msg : TMessage {Mensaje sobre el estado del ratón}

Si existe un clic en el área cliente de la ventana, se manda un mensaje de destrucción de la ventana.

18. FORMATO DE GRABACIÓN

PSI tiene un formato binario, de escritura secuencial variable que explicaremos ampliamente en este capítulo.

Cuando decimos que es de escritura/lectura variable, se establece que cada elemento definido dentro de la colección de bloques (*O_CollecBlks, E_State.Blocks* dentro de *O_LogDefWin*) escribe la cantidad de *bytes* necesarios para guardar su información. De esta forma, cada tipo de bloque tiene cierta información ordenada, que desea guardar dentro del archivo.

Precisamente este capítulo, dará el orden y el número de bytes que se tienen que leer de este archivo, de acuerdo al tipo de bloque que se tenga. Debido al diseño orientado a objetos, existe la propiedad de heredar comportamientos o servicios; entre éstos se encuentra el de guardar su información, por lo que será común observar que se mande llamar la grabación de la superclase y posteriormente, se grabe la información añadida de la subclase.

Este formato de grabación es el mismo utilizado en PSICF (*Process Simulation Interface Clipboard Format*) que desarrollaremos más adelante.

a. Formato de grabación para una representación gráfica (BMP)

El mapa de bits con formato BMP se graba con el estándar establecido (si tiene alguna duda, es conveniente que consulte el manual de referencia de *Windows*, en el apartado de archivos BMP).

El formato tiene la siguiente configuración:

a) Se graba un encabezado definido por la estructura *BITMAPFILEHEADER*, con los siguientes campos:

Campo	Tamaño	Descripción
bfType	WORD	Los *bytes* ORD("B"), ORD("M").
bfSize	DWORD	Tamaño total del archivo BMP. En el caso de PSI esto se refiere únicamente al tamaño de la imagen.
bfReserved1	WORD	Se pone igual a 0
bfReserved2	WORD	Se pone igual a 0
bfOffBits	DWORD	Número de *bytes* a partir de los cuales comienza la definición de la imagen.

b) Posteriormente se graba un registro que contiene la siguiente información:

Campo	Tamaño	Descripción
biSize	DWORD	Tamaño de esta estructura.
biWidth	DWORD	Ancho del BMP en *pixels*
biHeight	DWORD	Alto del BMP en *pixels*
biPlanes	WORD	Igual a 1 (VGA)
biBitCount	WORD	Número de bits para representar un color por *pixel* (1,4,8,24).
biCompression	DWORD	Esquema de compresión (0 para establecer ninguno).
biSizeImage	DWORD	Tamaño de bits de la imagen en *bytes*. (Sólo se requiere si hay algún esquema de compresión).
biXPelsPerMeter	DWORD	Resolución horizontal en *pixels* por metro.
biYPelsPerMeter	DWORD	Resolución vertical en *pixels* por metro
biClrUsed	DWORD	Número de colores usados en la imagen.
biClrImportant	DWORD	Número de colores importantes para la imagen.

c) Después de lo anterior, se graba una tabla de colores que consiste en 2 ó más estructuras RGBQUAD (dependiendo del valor de *biClrUsed*) que contienen los siguientes campos: *rgbBlue* (BYTE), *rgbGreen* (BYTE), *rgbRed* (BYTE) y *rgbReserved* (BYTE).

d) Finalmente, se escribe un vector de *bits* que define la imagen. Esta imagen comienza en el renglón inferior. Los renglones siempre empiezan con los *pixels* de la izquierda.
Para un BMP con 16 colores cada 4 *bits* representan 1 *pixel*; para uno de 256 colores cada 8 *bits* representan un *pixel*.

El único cambio que hace PSI al formato, es que el tamaño del archivo se convierte en el tamaño del archivo de la imagen.

Ahora veamos la información de cada tipo de bloque:

b. Clase O_Rule

Campo	Tamaño	Descripción
Rule	WORD	Regla de asociación ORD(TD_Rule).
ValUsr	Variable (al final un #0)	Valor de la regla del usuario.
Count	WORD	Número de elemento a grabar.
RuleItem	E_RuleItem (Id:WORD, W:REA).	Elemento
...	...	...

c. Clase O_Block

Se graba la siguiente información (en el orden establecido):

Campo	Tamaño	Descripción
Pos	DWORD	Posición del bloque X,Y
WH	DWORD	Ancho y alto del bloque
Mode	BYTE	Reservado
FirstTime	BYTE	Reservado
IdNum	WORD	Identificador del bloque
ResBlocks	Variable (En el formato de O_Rule)	Bloques de recursos
NxtBlocks	Variable (En el formato de O_Rule)	Bloques siguientes
PrvBlocks	Variable (En el formato de O_Rule)	Bloques anteriores
Bitmap	Variable (en el formato de BMP).	Representación BMP.

d. Clase O_BMPBlk

Graba el mismo formato de O_Block.

e. Clase O_Divider

Graba el mismo formato de O_Block.

f. Clase O_Generator

Graba la misma información que O_Block, añadiendo:

Campo	Tamaño	Descripción
Distribution	Variable (al final un #0)	Distribución
InitTime	Variable (al final un #0)	Tiempo para generar la primera parte.
GenGroups	Variable (al final un #0)	Número de partes por grupo.
MaxNumber	Variable (al final un #0)	Número máximo de partes.

g. Clase O_Queue

Graba la misma información que O_Block, añadiendo:

Campo	Tamaño	Descripción
InitLen	Variable (al final un #0)	Longitud inicial.
MaxNumer	Variable (al final un #0)	Número máximo de partes.
CostPerUnit	Variable (al final un #0)	Costo por unidad.
Tipo	WORD	Tipo de cola

h. Clase O_Server

Graba la misma información que O_Block, añadiendo:

Campo	Tamaño	Descripción
Distribution	Variable (al final un #0)	Distribución
CostPerTime	Variable (al final un #0)	Costo por unidad de tiempo
CostPerOper	Variable (al final un #0)	Costo por operación
AddDelay	Variable (al final un #0)	Tiempo añadido al final

i. Clase O_Assem, O_Batch, O_SingleOp, O_SumFlow, O_Transport

Graba el mismo formato de O_Server.

j. Clase de O_AGV, O_Convey, O_Crane, O_MotorTrsp

Graba el mimso formato de O_Server.

k. Clase O_Terminator

Graba la misma información que O_Block, añadiendo:

Campo	Tamaño	Descripción
NoPartsDec	Variable (al final un #0)	Núm. de partes a restar.
NoMaxParts	Variable (al final un #0)	Número máximo de partes.

l. Clase O_Union

Campo	Tamaño	Descripción
Pos	DWORD	Posición del bloque X,Y
WH	DWORD	Ancho y alto del bloque
Mode	BYTE	Reservado
FirstTime	BYTE	Reservado
Selected	BYTE	Reservado
XY	TA_Pts	Puntos que definen a la curva Bezier.
EditLine	BYTE	Reservado
ColLine	TColorRef	Color de la línea
WidthLine	WORD	Ancho de la línea
LineData	E_PenData	Información de la línea.

m. Clase O_TextBlk

Campo	Tamaño	Descripción
Pos	DWORD	Posición del texto.
WH	DWORD	Ancho y alto del texto.
PText	Variable (al final un #0)	Texto
InfoF	TLogFont	Información del tipo de letra.

n. Clase O_SimObj

Se utiliza la grabación de colecciones de *ObjectWindows* (Para mayor información consulte el manual de *ObjectWindows*) en los siguientes campos:

```
S.Put(Types);
S.Put(Attributes);
```

Donde *Types* y *attributes* son colecciones de tipo *TStrCollection*.
La rutina de *Put* guarda la constante definida para la clase de tipo *TStreamRec* y manda llamar al método *Store*.

La rutina de *Store* en colecciones hace los siguiente:

```
PROCEDURE TCollection.Store(VAR S: TStream);

        PROCEDURE DoPut(P: Pointer); FAR;
        BEGIN
            PutItem(S, P)
        END;

BEGIN
    S.Write(Count, 6); { Escribe 6 bytes }
    ForEach(@DoPutItem)
END;
```

o. Grabación del archivo de definición gráfica (extensión PSI)

Campo	Tamaño	Descripción
PenSize	WORD	Tamaño de la plumilla
Offset	DWORD	Reservado
ToolSel	WORD	Herramienta seleccionada
XYScale	DWORD	Tamaño del área de edición (X Y)
SimObj	Variable (Formato de O_SimObj)	Objeto que representa a la parte.
Count	WORD	Número de bloques definidos
...	...	...
TypeBlock	WORD	Tipo de bloque. ORD(*TD_ToolName*)
Block^.Store	Variable	Bloque en cuestión
...	...	...

p. Formato PSICF

Para que otras aplicaciones puedan utilizar el formato de PSICF se tiene la siguiente información:

El nombre del formato es PSICF, que se puede obtener con la función *GetClipboardFormatName* de *Windows API*.
Se manda el nombre del archivo de comunicación en formato *cf_Text*.
El archivo en PSICF tiene el formato siguiente:

Campo	Tamaño	Descripción
Count	WORD	Número de bloques definidos
...	...	...
TypeBlock	WORD	Tipo del Bloque : ORD(*TD_ToolName*)
Block^.Store	Variable	Formato del bloque en cuestión
...	...	...

19. CONVERTIDORES

Los convertidores dentro de PSI son archivos de enlace dinámico (DLL's) que manejan cuatro procedimientos exportables:

```
PROCEDURE OpenF(Name:PChar); EXPORT;
PROCEDURE Load; EXPORT;
PROCEDURE Convert(CommandLine:PChar); EXPORT;
PROCEDURE CloseF; EXPORT;
```

En *OpenF* se manda como parámetro el nombre del archivo con formato de definición gráfica (extensión .PSI).
Se ejecutan en el orden establecido.