REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	13 October 1995	Final Report

4. TITLE AND SUBTITLE

Object-Oriented Two-Dimensional Electromagetic Field Solver Code

5. FUNDING NUMBERS

F6170893W00612

6. AUTHOR(S)

Dr. Valentin Ivanov, Dr. Andrey Petkun, Dr. Victor Ryzhov, Dr. Igor Turchanovsky,

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Current Electronics Institute
Tomsk, Russia

8. PERFORMING ORGANIZATION REPORT NUMBER

SPC-93-4035

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

EOARD
PSC 802 BOX 14
FPO 09499-0200

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

SPC-93-4035

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

A

13. ABSTRACT (Maximum 200 words)

Object-oriented particles simulation project assumes an interactive operation under PC Windows. This brings about the necessity to construct a user-friendly interface and develop the object-oriented control and interaction of the parts of the model.

14. SUBJECT TERMS	15. NUMBER OF PAGES	16. PRICE CODE
	137	

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18
298-102

SPC-93-4035
F6170893W0612
Dr Barker has file report
"ECOPULSE"
Dgn Yu. Turchanovsky

```
\documentstyle[12pt,titlepage,bezier,a4]{article}
\textwidth 16true cm
\topmargin -1.4true cm
\def\pndh{\par\noindent\hrulefill}
\def\rot{\mathop{\rm rot}\nolimits}
\def\dvg{\mathop{\rm div}\nolimits}
\begin{document}

\title{Object-Oriented Two-Dimensional Electromagnetic Field Solver Code}
\author{written by:\\ Dr.~Valentin~Ivanov,\\ Dr.~Andrey~Petkun, \\
Dr.~Victor~Ryzhov, \\ Dr.~Igor~Turchanovsky, \\[5true mm]
in conjunction with Principle Investigator\\
Dr.~Victor~Ryzhov, \\
High Current Electronics Institute,
    Tomsk, Russia\\[5 true mm]
    for the OOPIC project funded by\\
    AIR Force Office of Scientific Research\\[5true mm]
    Contract SPC93--4035(F6170893W0612)}
\date{}
\maketitle

\tableofcontents
\pndh

\newpage

\section{Introduction}

Object-oriented particles simulation  project assumes an interactive
operation under PC Windows. This brings about the necessity to construct a
user-friendly interface and develop the object-oriented control and
interaction of the parts of the model. Therefore, while addressing the
general architecture of the OO model one should forsee the interaction
types of object classes not only within the model functionality itself
but also interactions between GUI classes, manager classes and
computational classes (Fig. 1). The first goal of decompositions is to separate
control, visualization and computation parts, thus providing
flexibility and extendibility of each of the sections irrespective of
each other. The problem is simplified by already existing class hierarchy
for the interface and the CUA standard supported, in particular, by OWL
Borland C++.

\begin{figure}[bht]
\unitlength=1.00mm
\linethickness{0.4pt}
\begin{center}
\begin{picture}(115.00,93.00)
\put(81.50,89.00){\oval(15.00,8.00)[]}
\put(81.50,89.00){\makebox(0,0)[cc]{USER}}
\put(0.00,55.00){\framebox(20.00,15.00)[cc]{}}
\put(0.00,30.00){\framebox(20.00,15.00)[cc]{}}
\put(30.00,25.00){\framebox(50.00,30.00)[cc]{}}
\put(55.00,60.00){\framebox(20.00,7.00)[cc]{Manager}}
\put(50.00,72.00){\framebox(65.00,7.00)[lc]{~~GUI-Interface}}
```

```
\put(60.00,28.00){\framebox(15.00,7.00)[cc]{Data}}
\put(35.00,28.00){\framebox(20.00,7.00)[cc]{Geometry}}
\put(85.00,48.00){\framebox(25.00,7.00)[cc]{Visualisation}}
\put(60.00,5.00){\framebox(20.00,10.00)[cc]{}}
\put(10.00,66.00){\makebox(0,0)[cc]{Windows}}
\put(10.00,59.00){\makebox(0,0)[cc]{messages}}
\put(10.00,41.00){\makebox(0,0)[cc]{Streams,}}
\put(10.00,34.00){\makebox(0,0)[cc]{Collections}}
\put(91.00,75.50){\makebox(0,0)[cc]{\scriptsize OWL}}
\put(33.00,50.00){\makebox(0,0)[lc]{Model}}
\put(50.00,41.00){\framebox(20.00,10.00)[cc]{}}
\put(60.00,48.00){\makebox(0,0)[cc]{Numerical}}
\put(60.00,44.00){\makebox(0,0)[cc]{algorithm}}
\put(70.00,12.00){\makebox(0,0)[cc]{Service}}
\put(70.00,8.00){\makebox(0,0)[cc]{classes}}
\put(45.00,18.00){\makebox(0,0)[cb]{\scriptsize Win API}}
\put(8.00,25.00){\makebox(0,0)[cc]{\scriptsize OWL}}
\put(25.00,41.00){\makebox(0,0)[cc]{\scriptsize Data}}
\put(30.00,58.00){\makebox(0,0)[cc]{\scriptsize Control}}
\put(25.00,67.00){\makebox(0,0)[lc]{\scriptsize Win API}}
\put(85.00,63.00){\makebox(0,0)[cc]{\scriptsize Control}}
\put(110.00,58.00){\makebox(0,0)[cc]{\scriptsize Win GDI}}
\put(93.00,40.00){\makebox(0,0)[cc]{\scriptsize Data}}
\put(15.00,9.00){\line(0,1){9.00}}
\put(27.00,18.00){\line(0,-1){9.00}}
\bezier{100}(15.00,18.00)(21.00,22.00)(27.00,18.00)
\bezier{100}(15.00,18.00)(21.00,14.00)(27.00,18.00)
\bezier{100}(15.00,9.00)(21.00,5.00)(27.00,9.00)
\put(21.00,4.00){\makebox(0,0)[cc]{HD memory}}
\put(0.00,71.00){\line(1,0){21.00}}
\put(21.00,71.00){\line(0,-1){17.00}}
\put(21.00,54.00){\line(-1,0){21.00}}
\put(0.00,54.00){\line(0,1){17.00}}
\put(52.00,41.00){\line(0,-1){6.00}}
\put(53.00,35.00){\line(0,1){6.00}}
\put(63.00,41.00){\line(0,-1){6.00}}
\put(64.00,35.00){\line(0,1){6.00}}
\put(60.00,32.00){\line(-1,0){5.00}}
\put(55.00,31.00){\line(1,0){5.00}}
\put(71.00,25.00){\line(0,-1){10.00}}
\put(70.00,15.00){\line(0,1){10.00}}
\put(31.00,24.00){\line(1,0){39.00}}
\put(71.00,24.00){\line(1,0){10.00}}
\put(81.00,24.00){\line(0,1){30.00}}
\put(82.00,53.00){\line(0,-1){30.00}}
\put(82.00,23.00){\line(-1,0){11.00}}
\put(70.00,23.00){\line(-1,0){38.00}}
\put(61.00,4.00){\line(1,0){20.00}}
\put(81.00,4.00){\line(0,1){10.00}}
\put(82.00,13.00){\line(0,-1){10.00}}
\put(82.00,3.00){\line(-1,0){20.00}}
\put(86.00,47.00){\line(1,0){25.00}}
\put(111.00,47.00){\line(0,1){7.00}}
\put(112.00,53.00){\line(0,-1){7.00}}
```

```
\put(112.00,46.00){\line(-1,0){25.00}}
\put(21.00,65.00){\vector(1,0){34.00}}
\put(55.00,62.00){\vector(-1,0){34.00}}
\put(65.00,70.00){\vector(0,-1){3.00}}
\put(65.00,69.00){\vector(0,1){3.00}}
\put(81.00,85.00){\vector(0,-1){6.00}}
\put(58.00,60.00){\vector(0,-1){5.00}}
\put(63.00,60.00){\vector(0,-1){9.00}}
\put(73.00,60.00){\vector(0,-1){25.00}}
\put(64.00,17.00){\vector(0,1){11.00}}
\put(64.00,17.00){\vector(-1,0){37.00}}
\put(20.00,20.00){\vector(-1,2){5.00}}
\put(12.00,30.00){\vector(1,-2){5.67}}
\put(55.00,60.00){\line(-1,0){32.00}}
\put(23.00,60.00){\vector(-1,-4){3.67}}
\put(50.00,46.00){\line(-1,0){5.00}}
\put(45.00,46.00){\line(0,-1){11.00}}
\put(45.00,39.00){\vector(-1,0){25.00}}
\put(75.00,32.00){\vector(4,3){21.00}}
\put(75.00,64.00){\vector(2,-1){18.00}}
\put(98.00,55.00){\vector(0,1){17.00}}
\end{picture}
\end{center}
\caption{Diagram of high-level relationships.}
\end{figure}
```

The OO programming allows description of the numerical model made of objects containing components of the physical task. These same objects, however, include abstract data describing the solution methods of the numerical model. Conventionally when writing a program a significant part of the code is taken by the input information detailing physical conditions of the task as well as by the selection of calculation data. Therefore, in developing OO-PIC we used an approach of structural differentiation of the code into functional sections connected with operation of the program. Thus, everything relating to input of initial and boundary conditions of the task and to visualization of information in our method is presented as a group of classes CUI-interface and visualization. In addition to these the program makes use of stream classes to save information on the model and the obtained data with the help of service stream classes. Between GUI and the numerical model we use the objects group ``Manager" responsible for handling messages between separate sections of the program and systems messages between OC Windows and the program. With such an approach at hand one can readily organize the temporal cycle of the task outstanding the objects describing the numerical model. The numerical model includes description of the task geometry, electrophysical data and the numerical algorithm. Therefore, while developing the class ``Model" we included an object desribing the relevant section. Thiss allows us (within one and the same approach for the same physical models) to easily modify the section in the oject ``Model" which is connected with modification of numerical algorithms. Utilization in the code of several objects ``Model" creates an opportunity for carrying investigation into optimization of plasma physics models and numerical algorithms.

```
\section{Differential Field Equations}
```

In the general case the macroscopic behavior of the electromagnetic fields of a continuous medium is described by the set of Maxwell's equations:

```
\begin{eqnarray}
 \rot\vec{E} &=& -\partial\vec{B}/ \partial t - \vec{K} , \label{2.1} \\
\dvg \vec{B} &=& \rho_{m} ,                    \label{2.2}
 \\ \rot \vec{H} &=& \partial \vec{D}/ \partial t + \vec{J} ,
 \label{2.3} \\ \dvg \vec{D} &=& \rho_{e} ,
 \label{2.4}
\end{eqnarray}
```

where $\vec E$ and $\vec B$ are the electric and magnetic fields, respectively. The electrostatic and magnetic induction are defined by the vectors $\vec D$ and $\vec B$, the volume densities of electric and magnetic currents are given by the vectors $\vec J$ and $\vec K$, and the corresponding space charge densities are denoted by ρ_m and ρ_e.

Leaving aside the philosophical problems concerning the interpretation of the above equations that can be found in books \cite{1}-\cite{9}, we should note that in some cases the introduction of magnetic currents and charges offers a convenient formalism that makes electric and magnetic fields tantamount, so that the equations become symmetrical.

Maxwell's equations for a continuous medium are written in the form

```
\begin{eqnarray}
  \vec{D} &=& \hat{\varepsilon} (E) \vec{E} ,     \label{2.5} \\
  \vec{B} &=& \hat{\mu} (H) \vec{H} ,          \label{2.6}
\end{eqnarray}
```

where the $\hat\varepsilon$ and $\hat\mu$ tensors are, in the general case, nonlinear functions of field strength and, indirectly, functions of coordinates and time.

Given a temporal dispersion, the polarization and magnetization of the medium at a given time are determined by the values of these quantities at preceding times. The time dependencies of the tensors are given by the integral equations

```
\begin{eqnarray}
 \varepsilon(t) = \int\varepsilon(\tau)G_{1}(t-\tau)d\tau ,  \label{2.7} \\
 \mu (t) = \int \mu (\tau ) G_{2}(t- \tau ) d \tau .   \label{2.8}
\end{eqnarray}
```

The nuclei $G_1(t-\tau)$ and $G_2(t-\tau)$ describe the relaxations mechanisms of polarization and magnetization of the medium, which possesses in this case a sort of ``memory''.

From the mathematical point of view, this type of the problem is complicated in that the tensors $\hat\varepsilon$ and $\hat\mu$ are not single-valued functions of the field even in a steady-state treatment. This shows up in hysteretic phenomena and substantially dramatizes the problem concerning the solution uniqueness.

Written in the above form, Eqs. (\ref{2.5}) to (\ref{2.8}) present in fact a phenomenological model of a medium, where the microfields of elementary charge carriers are averaged in volumes exceeding the atomic sizes and, the more so, the charge carriers {\em per se}.

The external field sources, space charges and currents are not absolutely independent quantities. They obey the conservation laws formulated as continuity equations:

$$\partial \rho_{e}/\partial t + \dvg \vec{J} &=& 0 , \quad \label{2.9} \\$$
$$\partial \rho_{m}/\partial t + \dvg \vec{K} &=& 0 , \quad \label{2.10}$$

and the relations between the current densities in the medium and the field characteristics are given by generalized Ohm's laws

$$\vec{J} &=& \gamma_{e}(E) \vec{E} , \label{2.11} \\$$
$$\vec{K} &=& \gamma_{m}(H) \vec{H} . \label{2.12}$$

The electric conductivity, $\gamma_e(E)$, and magnetic conductivity, $\gamma_m(H)$, of a medium are generally nonlinear tensors like ε and μ.

To isolate a unique solution to the set of Eqs. (\ref{2.1})--(\ref{2.4}) it is necessary to formulate the boundary-value problem by setting boundary conditions for the field components at the intermediate boundaries:

$$E_{\tau_+} = E_{\tau_-} , \:\: H_{\tau_+} = H_{\tau_-}, \label{2.13} \\$$
$$D_{n_+} = D_{n_-} , \:\: B_{n_+} = B_{n_-} \nonumber$$

and at perfectly conducting surfaces:

$$D_{n} = \sigma_{e} , \:\: B = \sigma_{m}, \label{2.14} \\$$
$$H_{\tau} = j_{e} , \:\: E_{\tau} = j_{m}, \nonumber$$

Here the indices τ and n are associated with tangential and normal field components, respectively, and j_e, j_m, σ_e, and σ_m are the respective densities of electric and magnetic surface currents and charges. The signs ``+'' and ``-'' refer to different sides of boundaries.

For the impedance-type boundaries, whose radii of curvature are large compared to the skin layer thickness, the Leontovich boundary conditions are valid, which relate the surface current densities as follows:

$$\vec{K} = Z [\vec{n} \times \vec{J}] , \label{2.17}$$

Here Z is the wave impedance of the conducting surface.

For the space charges and currents, it is necessary to set their initial $(t=0)$ distributions in volume V:

$$\rho_{e} (r,t_0) = \rho_{e0}(r), \:\:$$
$$\rho_{m} (r,t_0) = \rho_{m0}(r), \:\: r \in V, \label{2.15} \\$$
$$\vec{J} (r,t_0) = \vec J_0(r), \:\:$$
$$\vec{K} (r,t_0) = \vec K_0(r), \:\: r \in V. \label{2.16}$$

Given dispersion media, it is necessary to prescribe the `prehistory' for

the polarization and magnetization of the medium by Eqs. \ref{2.7} and \ref{2.8}.

In some cases, it appears convenient to introduce auxiliary functions, such us scalar potentials ϕ_{e} and ϕ_{m} and vector potentials A and F related to the field components by the equations

```
\begin{eqnarray}
 \vec{E} = - \nabla \phi_{e} - \partial \vec{A} / \partial t, &&
 \vec{D} = \rot \vec{F} ,                    \label{2.18} \\
 \vec{H} = - \nabla \phi_{m} - \partial \vec{F} / \partial t, &&
 \vec{B} = \rot \vec{A}.
\end{eqnarray}
```

In order that the introduced quantities be singlevalued, a certain gauge condition should by applied to them. For example, the Lorentz gauging is described by the equations

```
\begin{eqnarray}
 \dvg \vec{A} + \varepsilon\,\mu\,\partial\phi_e/\partial t +
  \mu\,\gamma_e\phi_e &=& 0, \label{2.19} \\
 \dvg \vec{F} + \varepsilon\,\mu \,\partial \phi_m/\partial t +
  \varepsilon\,\gamma_m\phi_m &=& 0.  \nonumber
\end{eqnarray}
```

Having performed a rotor operation on Eqs. \ref{2.1} and \ref{2.3}, we obtain so-called wave equations with separated characteristics of electric and magnetic fields:

```
\begin{eqnarray}
 \rot (\mu^{-1} \rot \vec{E}) + \varepsilon \,\partial^{2} \vec{E}/\partial
 t^{2} &=& \partial \vec{J}/\partial t + \gamma_{e} \partial \vec{E}/
 \partial t, \label{2.20} \\
 \rot (\varepsilon^{-1} \rot \vec{H}) + \mu
 \,\partial^{2} \vec{H}/\partial t^{2} &=& \partial \vec{K}/\partial t +
 \gamma_{m} \partial \vec{H}/ \partial t .  \nonumber
\end{eqnarray}
```

If we introduce a symbolic d'Alamberian $\Box=\Delta-\partial^2/\partial t^2$, where $\Delta=d^2/dx^2+d^2/dy^2+d^2/dz^2$ is Laplace's operator, then we obtain similar wave equation for the fields and potentials:

```
\begin{eqnarray}
 \Box\vec{E} &=& \mu \,\frac{\partial}{\partial t} (\vec{J} + \gamma_{e}
    \vec{E}) , \cr
 \Box\vec{H} &=& \varepsilon \,\frac{\partial}{\partial t} (\vec{K} +
    \gamma_{m} \vec{H}) , \cr
 \Box\vec{A} &=& \mu\, (\gamma_{e} \frac{\partial \vec{A}}{\partial t} -
    \vec{J}) , \cr
 \Box\vec{F} &=& \varepsilon\, (\gamma_{m} \frac{\partial \vec{F}}
    {\partial t} - \vec{K}) , \label{2.21} \\
 \Box \phi_{e} &=& \mu \,\gamma_{e} \frac{\partial \phi}{\partial t} -
    \frac{\rho_{e}}{\varepsilon}, \cr
 \Box \phi_{m} &=& \varepsilon \,\gamma_{m} \frac{\partial \phi_{m}}
    {\partial t} - \frac{\rho_{m}}{\mu}. \nonumber
\end{eqnarray}
```

In the right-hand sides of these equations, the first term describes

the external or foreign field sources and the second one the conduction currents of the medium. The potentials $\vec A$ and $\vec F$, ϕ_e and ϕ_m are not independent, as the field strength vectors are related to the induction vectors by field equations. The introduction of one or another potential couple depends on the statement of the problem, i.e. on which type of field source, electric or magnetic, can be introduced in a more convenient manner.

At this point we suppose that the statement of the problem in the form of a set of Maxwell's equations is completed and we may turn to the discussion of the most important particular cases. Throughout the subsequent consideration we shall deal with dispersion-free media.

First of all, the case of harmonic oscillations deserves consideration, where any characteristic of the field or the source, $T(r,t)$, can be expressed as

```
\begin{equation}
 T (r, t) = T_0(r) \,e^{i\omega t} ,      \label{2.22}
\end{equation}
```

where T_0 and ω are, respectively, the oscillation amplitude and frequency.

Such statements are typical of electrodynamic systems possessing pronounced selective or resonance capabilities, such as waveguides, resonators, and slowdown system in microwave devices. For this case Maxwell's equations take the form

```
\begin{eqnarray}
 \rot \vec{E}_{0} &=& -i\omega \vec{B}_{0} + \vec{K}_{0} ,
                                    \label{2.23} \\
 \rot \vec{H}_{0} &=&  i\omega \vec{D}_{0} + \vec{J}_{0} .\nonumber
\end{eqnarray}
```

Using two remaining equation, Eqs. (\ref{2.2}) and (\ref{2.4}), we may represent the complete set of equations as a couple of second-order equations, each including quantities only relevant to the electric field or to the magnetic field:

```
\begin{eqnarray}
 \rot(\mu^{-1}\rot\vec E_0) &=& \omega\vec D_0-
  i\omega\vec J_0-\rot(\mu^{-1}\vec K_0), \label{2.24} \\
 \rot(\varepsilon^{-1}\rot\vec H_0) &=& \omega\vec B_0 -
  i\omega\vec K_0+\rot(\varepsilon^{-1} \vec{J}_{0}).\nonumber
\end{eqnarray}
```

For piecewise homogeneous media these equations can be simplified to

```
\begin{eqnarray}
 \rot\rot\vec E_0 - k^2\vec E_0 &=& {} - i\omega\mu\vec J_0 -
   \rot\vec K_0, \label{2.25} \\
 \rot\rot\vec H_0 - k^2\vec H_0 &=& {} - i\omega\varepsilon\vec K_0 +
   \rot\vec J_0, \nonumber
\end{eqnarray}
```

where $k^2=\varepsilon\mu\omega^2=(\omega/c)^2$, $\vec k$ is the wave vector and c is the velocity of propagation of electromagnetic oscillations in the medium.

With the above assumptions, the finite conductivity of the medium can be taken into account by introducing complex quantities marked by an upper dot, $\dot\varepsilon=\varepsilon-i\gamma_e/\omega$, $\dot\mu=\mu-i\gamma_m/\omega$, the relations between the field strengths and the sources intensities are determined by formulas (\ref{2.11}) and (\ref{2.12}).

If we put the source intensities equal to zero, nontrivial solutions to the set of Eqs. (\ref{2.25}) will exist only for fixed k or ω values. So the problem is reduced to finding the eigenvalue spectrum for k (or resonance frequency ω) and the free oscillation modes described by the Helmholtz equations

```
\begin{equation}
 \begin{array}{rcl}
  \Delta \vec{E}_{0} + k^{2} \vec{E}_{0} &=& 0, \\
  \Delta \vec{H}_{0} + k^{2} \vec{H}_{0} &=& 0 .
 \end{array}   \label{2.26}
\end{equation}
```

Note that, despite the fact that the equations for the electric and magnetic fields are separated, this nevertheless does not mean that they are independent of each other, since in an oscillatory mode the electric field energy is continuously converting into the magnetic field energy and vice versa. This is not the case for steady-state or time-independent fields.

The equations for static fields are substantially simplified, as they involve no time derivatives, and can be separated into couples. Let us first consider the magnetic field equations

```
\begin{eqnarray}
  \dvg \vec{B} &=& \rho_{m} ,      \label{2.27} \\
  \rot \vec{H} &=& \vec{J} .       \label{2.28}
\end{eqnarray}
```

The efficiency of one or another algorithm of solving the problems of magnetostatics depends essentially on the details of the problem statement and on the choice for the unknown quantities fields, vector or scalar potentials. These issues are treated rather comprehensively in review articles \cite{10}--\cite{15}. We shall only give a brief outline of these approaches.

The simplest algorithm involves a straightforward solution of the set of Eqs. (\ref{2.27}) and (\ref{2.28}) jointly with field equations

```
\begin{equation}
        B_i=\mu_{ij}(H)H_j.        \label{2.29}
\end{equation}
```

This technique is not frequently applied because of its high cost. In deriving Eqs. (\ref{2.29}) we used the Einstein rule for summation over repeating indices that will be frequently used in the subsequent discussion.

A more widespread approach is to introduce a vector potential by the

relation $\vec B=\rot\vec A$, that results in a unique second-order equation
\begin{equation}
\Delta \vec{A} = -\mu (\vec{J} + \vec{J}_{m}) . \label{2.30}
\end{equation}
Here $\vec J_m=\rot\vec M$ is the volume density of the magnetization current for the medium, $\vec J$ is the density of the conduction current for the foreign field sources, and the magnetization vector $\vec M$ is related to $\vec B$ and $\vec H$ as
\begin{equation}
\vec{B} = \vec{H} + \vec{M} . \label{2.31}
\end{equation}

Thus, for systems including constant magnets we way put $\vec J=0$, while for linear media we have $\vec M=0$ and $\vec J_m=0$. Arrange the boundary conditions for Eqs. (\ref{2.30}) to the form
\begin{equation}
\vec{n} \times (\rot\vec{A}_+ - \rot\vec{A}_-) =
\mu(\vec{j} + \vec{j}_m) , \label{2.32}
\end{equation}
where $\vec j$ and $\vec j_m$ are the corresponding surface current.

Using a scalar magnetic potential, ϕ_m, such that $\vec H=-\nabla\phi_m$, leads to quasilinear equation of the form
\begin{equation}
\frac{\partial}{\partial x_i} \left(\mu_{ij} \frac{\partial\phi_m}{\partial x_j}\right)=-\rho \label{2.33}
\end{equation}
with the boundary conditions given by
\begin{equation}
\mu_+\frac{\partial\phi_m}{\partial n_+}+\left(\vec n [\vec j\times\vec n]\right)=\mu_-\frac{\partial\phi}{\partial n_-}. \label{2.34}
\end{equation}

The use of a scalar function substantially moderates the requirements to the computer operative memory needed to remember the values of the unknown variables, being, however, not free from some disadvantages. The principal disadvantage is that a scalar potential is not a single-valued function in a space incorporating vortex sources, i.e. conduction currents. In order to eliminate this defect, an intermediate boundary is introduced into the space, at which a jump in ϕ_m proportional to the total excitation current of the magnetic system is specified.

Most widely used in practice is the method of a modified scalar potential ψ_m introduced by the relationship
\begin{equation}
\vec{H} = \vec{H}_{0} - \nabla \psi_{m} . \label{2.35}
\end{equation}
Here the total field $\vec H$ consists of the vortex component $\vec H_0$ excited by the given conduction current $\vec J=\rot\vec H_0$ and the potential components described by a function ψ_m obeying the equation
\begin{equation}
\dvg(\mu\nabla\psi_m) = \dvg (\mu\vec{H}_0) . \label{2.36}

\end{equation}

In such a model a nonlinear boundary problem is solved only for the region filled with a magnetized medium, while the quantity $\vec H_0$ involved in the right-hand side of the equation is calculated by integration over the volume occupied by conduction currents:

\begin{equation}
\vec{H}_{0}(r) = \frac{1}{4 \pi} \int_{V} \frac{[\vec{J} \times \vec{r}\,]}{|\vec{r} - \vec{r}\,'|}\, dV, \:\:\vec{r}\,' \in V. \label{2.37}
\end{equation}

The simplest form is taken by the equations of electrostatics for linear piecewise homogenous media, which transform into the Laplace's equation

\begin{equation}
\Delta \phi = 0 . \label{2.38}
\end{equation}

The Dirichlet conditions

\begin{equation}
\phi |_{S_0} = \phi_0(r) \label{2.39}
\end{equation}

are set at the surfaces of the conducting electrodes, while at the boundary between two media having dielectric constants ε_+ and ε_- the continuity conditions for the normal component of the electrostatic induction vector

\begin{equation}
\varepsilon_+\frac{\partial\phi}{\partial n}|_{S_+} = \varepsilon_- \frac{\partial\phi}{\partial n}|_{S_-} \label{2.40}
\end{equation}

is satisfied.

The current passage through a medium having a finite conductivity is described by the equation

\begin{equation}
\frac{\partial}{\partial x_{i}} \left(\gamma_{ij} \frac{\partial \phi} {\partial x_{j}} \right) = 0 , \label{2.41}
\end{equation}

where the conductivity tensor for the medium, $\hat\gamma$, is related to the current density by the expression $J_i=-\gamma_{ij}\partial\phi/\partial x_j$.

Statements of electric elasticity and magnetic thermoelasticity problems can be found in articles by Bitsadze \cite{16} and Rukhadze \cite{17}.

\section{Algorithms for Numerical Solution the Problems of Electrodynamics}

For a finite difference approximation of Maxwell's equations (\ref{2.1}) and (\ref{2.3}), the great diversity of double-layer schemes applied for solving the evolution problems can be classified as explicit schemes described by the equations

\begin{eqnarray}
B^{n+1/2} &=& B^{n-1/2} + \tau (\rot E^{n} - K^{n}) , \nonumber \\ &&

```
                                        \label{3.1} \\
  E^{n+1} &=& E^{n} + \tau ( -rot B^{n+1/2} - J^{n+1/2} ) \nonumber
\end{eqnarray}
```
and implicit schemes described by the equation
```
\begin{eqnarray}
  B^{n+1}_{k+1} &=&  B^{n} + \tau (\rot(E^{n+1}_k + E^n) - K^{n+1/2} ) ,
  \nonumber \\ &&
                        \label{3.2} \\
  E^{n+1}_{k+1} &=& E^{n} + \tau(-\rot( B^{n+1}_{k} + B^{n} ) - J^{n+1/2} ).
   \nonumber
\end{eqnarray}
```
Here the superscript denotes the number of time steps. The Lack's time-centered explicit scheme, when applied to hyperbolic equation sets like Eq. (\ref{3.1}), is of the second-order accuracy. It is stable when the Courant condition $\tau/h<1$, where τ is the step of discretization in time and $h = (h^{-2}_{x\,\min} + h^{-2}_{y\,\min} + h^{-2}_{z\,\min})^{-1/2}$ is the step of discretization in spatial variables is satisfied.
Here the superscript denotes
the number of time steps. The idea of applying these schemes to Maxwell's equations was proposed by Yee \cite{42}. So stringent conditions for stability result in that most of versions of this scheme use a uniform spatial grid.

The implicit scheme, Eqs. (\ref{3.2}) is absolutely stable. However, it requires inner iterations at each time step marked with a subscript. The initial values of variables in iterations may be set equal to the corresponding values at the preceding time step, e.g. $E_1^{n+1}=E^n$. As pointed out in Ref. \cite{43}, in order to attain the convergence to a relative accuracy of 10^{-3}, four to nine iterations are needed. Application of explicit schemes may be recommended for the simulation of the motion of ultrarelativistic particles as well as for systems whose state is close to physical instability.

\subsection{Three--Dimensional Fields}

Let us write an explicit scheme in a Cartesian coordinate system with shifted grids, which is of the second-order accuracy both in time and in spatial variables and was used in most of the relevant computations \cite{44}-\cite{61}. This scheme uses a uniform grid
$\{ x_{i} = (i-1) h_{x} \} \times \{ y_{j} = (j-1) h_{y} \} \times
\{ z_{k} = (k-1) h_{z} \} \times \{ \tau_{n} = (n-1) h_{\tau} \}$,
such that
```
\begin{eqnarray}
\frac{( E_{x} )^{n+1}_{i-1/2,j,k} - ( E_{x} )^{n}_{i-1/2,j,k}}{\tau} =
\frac{( B_{z} )^{n+1/2}_{i-1/2,j+1/2,k} - ( B_{z} )^{n+1/2}_{i-1/2,j-1/2,k}}{h_{y}} \nonumber \\
- \frac{( B_{y} )^{n+1/2}_{i-1/2,j,k+1/2} - ( B_{y} )^{n+1/2}_{i-1/2,j,k-1/2}}{h_{z}}
- ( J_{x} )^{n+1/2}_{i-1/2,j,k} , \label{3.3} \\
\frac{( E_{y} )^{n+1}_{i,j-1/2,k} - ( E_{y} )^{n}_{i,j-1/2,k}}{\tau} =
\frac{( B_{x} )^{n+1/2}_{i,j-1/2,k+1/2} - ( B_{x} )^{n+1/2}_{i,j-1/2,k-1/2}}{h_{z}} \nonumber \\
- \frac{( B_{z} )^{n+1/2}_{i+1/2,j-1/2,k} - ( B_{z} )^{n+1/2}_{i-1/2,j-1/2,k}}{h_{x}}
- ( J_{y} )^{n+1/2}_{i,j-1/2,k} , \label{3.4} \\
\frac{( E_{z} )^{n+1}_{i,j,k-1/2} - ( E_{z} )^{n}_{i,j,k-1/2}}{\tau} =
\frac{( B_{y} )^{n+1/2}_{i+1/2,j,k-1/2} - ( B_{y} )^{n+1/2}_{i-1/2,j,k-1/2}}{h_{x}} \nonumber \\
- \frac{( B_{x} )^{n+1/2}_{i,j+1/2,k-1/2} - ( B_{x} )^{n+1/2}_{i,j-1/2,k-1/2}}{h_{y}}
```

```
- ( J_{z} )^{n+1/2}_{i,j,k-1/2} , \label{3.5} \\
\frac{( B_{x} )^{n+1/2}_{i,j-1/2,k-1/2} - ( B_{x} )^{n-1/2}_{i,j-1/2,k-1/2}}{\tau} =
\frac{( E_{y} )^{n}_{i,j-1/2,k} - ( E_{y} )^{n}_{i,j-1/2,k-1}}{h_{z}} \nonumber \\
- \frac{( E_{z} )^{n}_{i,j,k-1/2} - ( E_{z} )^{n}_{i,j-1,k-1/2}}{h_{y}}
- ( K_{x} )^{n}_{i,j-1/2,k-1/2} , \label{3.6} \\
\frac{( B_{y} )^{n+1/2}_{i-1/2,j,k-1/2} - ( B_{y} )^{n-1/2}_{i-1/2,j,k-1/2}}{\tau} =
\frac{( E_{z} )^{n}_{i,j,k-1/2} - ( E_{z} )^{n}_{i-1,j,k-1/2}}{h_{x}} \nonumber \\
- \frac{( E_{x} )^{n}_{i-1/2,j,k} - ( E_{x} )^{n}_{i-1/2,j,k-1}}{h_{z}}
- ( K_{y} )^{n}_{i-1/2,j,k-1/2} , \label{3.7}\\
\frac{( B_{z} )^{n+1/2}_{i-1/2,j-1/2,k} - ( B_{z} )^{n-1/2}_{i-1/2,j-1/2,k}}{\tau} =
\frac{( E_{x} )^{n}_{i-1/2,j,k} - ( E_{x} )^{n}_{i-1/2,j-1,k}}{h_{y}} \nonumber \\
- \frac{( E_{y} )^{n}_{i,j-1/2,k} - ( E_{y} )^{n}_{i-1,j-1/2,k}}{h_{x}}
- ( K_{z} )^{n}_{i-1/2,j-1/2,k} . \label{3.8}
\end{eqnarray}

\subsection{Plane Symmetric Fields}

The case of plane symmetrical fields characteristic of uniform waveguides of
any cross section is analyzed using the limit $h_z\Rightarrow\infty$ and
removing the indices corresponding to the coordinate $z$:
\begin{eqnarray}
\frac{( E_{x} )^{n+1}_{i-1/2,j} - ( E_{x} )^{n}_{i-1/2,j}}{\tau} &=&
- ( J_{x} )^{n+1/2}_{i-1/2,j} \nonumber \\ &+&
\frac{( B_{z} )^{n+1/2}_{i-1/2,j+1/2} - ( B_{z} )^{n+1/2}_{i-1/2,j-1/2}}{h_{y}}
, \label{3.9} \\
\frac{( E_{y} )^{n+1}_{i,j-1/2} - ( E_{y} )^{n}_{i,j-1/2}}{\tau} &=&
- ( J_{y} )^{n+1/2}_{i,j-1/2} \nonumber \\
&-& \frac{( B_{z} )^{n+1/2}_{i+1/2,j-1/2} - ( B_{z} )^{n+1/2}_{i-1/2,j-1/2}}{h_{x}}
, \label{3.10} \\
\frac{( E_{z} )^{n+1}_{i,j} - ( E_{z} )^{n}_{i,j}}{\tau} &=&
\frac{( B_{y} )^{n+1/2}_{i+1/2,j} - ( B_{y} )^{n+1/2}_{i-1/2,j}}{h_{x}}
- ( J_{z} )^{n+1/2}_{i,j} \nonumber \\
&-& \frac{( B_{x} )^{n+1/2}_{i,j+1/2} - ( B_{x} )^{n+1/2}_{i,j-1/2}}{h_{y}}
, \label{3.11} \\
\frac{( B_{x} )^{n+1/2}_{i,j-1/2} - ( B_{x} )^{n-1/2}_{i,j-1/2}}{\tau} &=&
- \frac{( E_{z} )^{n}_{i,j} - ( E_{z} )^{n}_{i,j-1}}{h_{y}} \nonumber \\
&&- ( K_{x} )^{n}_{i,j-1/2} , \label{3.12}
\end{eqnarray}
\begin{eqnarray}
\frac{( B_{y} )^{n+1/2}_{i-1/2,j} - ( B_{y} )^{n-1/2}_{i-1/2,j}}{\tau} &=&
- ( K_{y} )^{n}_{i-1/2,j} \nonumber \\
&+& \frac{( E_{z} )^{n}_{i,j} - ( E_{z} )^{n}_{i-1,j}}{h_{x}}
, \label{3.13} \\
\frac{( B_{z} )^{n+1/2}_{i-1/2,j-1/2} - ( B_{z} )^{n-1/2}_{i-1/2,j-1/2}}{\tau} &=&
\frac{( E_{x} )^{n}_{i-1/2,j} - ( E_{x} )^{n}_{i-1/2,j-1}}{h_{y}} \nonumber \\
- \frac{( E_{y} )^{n}_{i,j-1/2} - ( E_{y} )^{n}_{i-1,j-1/2}}{h_{x}}
&-& ( K_{z} )^{n}_{i-1/2,j-1/2} . \label{3.14}
\end{eqnarray}

\subsection{Axially Symmetric Fields}

For a cylindrical coordinate system $(r,\phi,z)$ with the assumption that
$\partial/\partial\phi\equiv0$ we have the scheme
\begin{eqnarray}
```

```
\frac{( E_{r} )^{n+1}_{i-1/2,j} - ( E_{r} )^{n}_{i-1/2,j}}{\tau} &=&
- ( J_{r} )^{n+1/2}_{i-1/2,j} \nonumber \\ &+&
\frac{( B_{\phi} )^{n+1/2}_{i-1/2,j+1/2} - ( B_{\phi} )^{n+1/2}_{i-1/2,j-1/2}}{h_{z}}
, \label{3.15} \\
\frac{( E_{z} )^{n+1}_{i,j-1/2} - ( E_{z} )^{n}_{i,j-1/2}}{\tau} &=&
- ( J_{z} )^{n+1/2}_{i,j-1/2} - \nonumber \\
- [ r_{i+1/2}( B_{\phi} )^{n+1/2}_{i+1/2,j-1/2} &-& r_{i-1/2}
( B_{\phi} )^{n+1/2}_{i-1/2,j-1/2} ] / ( r_{i}h_{r} ) , \label{3.16} \\
\frac{( E_{\phi} )^{n+1}_{i,j} - ( E_{\phi} )^{n}_{i,j}}{\tau} &=&
\frac{( B_{r} )^{n+1/2}_{i+1/2,j} - ( B_{r} )^{n+1/2}_{i-1/2,j}}{h_{z}}
- ( J_{\phi} )^{n+1/2}_{i,j} \nonumber \\
&-& \frac{( B_{z} )^{n+1/2}_{i,j+1/2} - ( B_{z} )^{n+1/2}_{i,j-1/2}}{h_{r}}
, \label{3.17} \\
\frac{( B_{r} )^{n+1/2}_{i,j-1/2} - ( B_{r} )^{n-1/2}_{i,j-1/2}}{\tau} &=&
- \frac{( E_{\phi} )^{n}_{i,j} - ( E_{\phi} )^{n}_{i,j-1}}{h_{z}} \nonumber \\
&-& ( K_{r} )^{n}_{i,j-1/2} , \label{3.18} \\
\frac{( B_{z} )^{n+1/2}_{i-1/2,j} - ( B_{z} )^{n-1/2}_{i-1/2,j}}{\tau} &=&
- ( K_{z} )^{n}_{i-1/2,j} \nonumber \\ &+&
\frac{r_{i}( E_{\phi} )^{n}_{i,j} - r_{i-1}( E_{\phi} )^{n}_{i-1,j}}{r_{i-1/2}h_{z}}
, \label{3.19} \\
\frac{( B_{\phi} )^{n+1/2}_{i-1/2,j-1/2} - ( B_{\phi} )^{n-1/2}_{i-1/2,j-1/2}}{\tau} &=&
\frac{( E_{r} )^{n}_{i-1/2,j} - ( E_{r} )^{n}_{i-1/2,j-1}}{h_{z}} \nonumber \\
- \frac{( E_{z} )^{n}_{i,j-1/2} - ( E_{z} )^{n}_{i-1,j-1/2}}{h_{r}}
&-& ( K_{\phi} )^{n}_{i-1/2,j-1/2} . \label{3.20}
\end{eqnarray}
```

By passing the limit, we obtain for the axis of symmetry:

```
\begin{equation}
( E_{z} )^{n+1}_{1,k-1/2} - ( E_{z} )^{n}_{1,k-1/2} = \frac{4 \, \tau }{h_{r}}
( B_{\phi} )^{n+1/2}_{3/2,k-1/2} - 4 \, \pi \tau ( J_{z} )^{n+1/2}_{1,k-1/2} .
\label{3.21}
\end{equation}
```

Approximation of Boundary Conditions

The principal disadvantage of rectangular grids is that the boundary of the simulation region for any actual problem does not conform to the grid, i.e., does not lie along the grid lines or parallel to them. The roughest model for taking into account boundary conditions approximates the actual boundary by a polygonal line consisting of segments that run along the grid lines as close to the actual boundary as possible. The perfectly conducting boundaries, the tangential components of the electric field, and the normal components of the magnetic field at corresponding points of the approximated boundary are set to be zero for each time step. An analysis of the properties of the difference scheme shows that with such an approximation of boundary conditions the scheme as a whole has the first rather than second order of convergence, if only there exits just one segment of the actual boundary that does not coincide with the grid lines.
By contrast, at the axis of symmetry and in the planes of symmetry of the solution the normal components of the electric field and the tangential components of the magnetic field are set to be zero.

It appears more

complicated to analyze the conditions for emission of an electromagnetic wave from boundary. They result in simple expressions only in the case when the TEM wave is normally incident on a boundary segment that coincides with a grid line. For instance, for a wave propagating in a coaxial along the z axis in the positive direction these conditions in a cylindrical coordinate systems have the form

\begin{equation}
B_r = -E_\phi,\quad B_\phi = E_r,\quad E_z = B_z = 0. \label{3.22}
\end{equation}

For a wave striking the boundary at an angle these conditions will not be exact. In order to diminish the effect of the inexactness of the boundary conditions for an inclined incident wave, open boundaries should be placed at a distance from the region of charged particle-field interaction or separated from this region by areas of finite conductivity, which well absorb radiation. Conditions (\ref{3.22}) are approximated by the finite-difference equations

\begin{eqnarray}
(B_{r})^{n+1/2}_{i,k+1/2} + (B_{r})^{n-1/2}_{i,k-1/2} &=&
- 2 \, (E_{\phi})^{n}_{i, k} , \label{3.23} \\
(B_{\phi})^{n+1/2}_{i-1/2,k+1/2} + (B_{\phi})^{n-1/2}_{i-1/2,k-1/2} &=&
2 \, (E_{r})^{n}_{i-1/2,k} , \label{3.24} \\
(E_{z})^{n}_{i,k+1/2} + (E_{z})^{n}_{i,k-1/2} &=& 0 , \label{3.25} \\
(B_{z})^{n+1/2}_{i-1/2,k+1/2} + (B_{z})^{n+1/2}_{i-1/2,k-1/2} &=& 0 .
\label{3.26}
\end{eqnarray}

Here the index k at the boundary is assumed to be fixed.

Engquist and Majda \cite{62} and Mur \cite{63} considered boundary conditions of the absorbing type for the case of an arbitrary position of the wave incident on a plane boundary.

Let, for instance, a plane wave having the velocity components c_x, c_y, and c_z such that $c_x^2+c_y^2+c_z^2=c^2$ be incident in the direction of decreasing of the coordinate x. Then for the plane $x = 0$ any component of the field W satisfies the wave equation

\begin{equation}
\left(\partial_x - \frac{\sqrt{1 - (c/c_y)^2 - (c/c_z)^2}}{c}
\partial_t \right) W |_{x = 0} = 0 . \label{3.27}
\end{equation}

Expanding the square root into a series, we obtain boundary conditions of the first order

\begin{equation}
\left(\partial_x - \frac{1}{c} \partial_t \right) W |_{x = 0} = 0 \label{3.28}
\end{equation}

and the second order

\begin{equation}
\left(\partial^2_{xt} - \frac{1}{c} \partial^2_{tt} + \frac{c}{2}
(\partial^2_{yy} + \partial^2_{yy})\right) W |_{x = 0} = 0. \label{3.29}
\end{equation}

These equations can be substantially simplified for a two-dimensional case. Consider, for instance, a field independent of z and an

E-polarization wave having the components E_z, H_x, and H_y only. From Maxwell's equations it follows that

$$\frac{\partial B_x}{\partial t} = - \frac{\partial E_z}{\partial y}. \label{3.30}$$

Substitution of Eq. (\ref{3.30}) into Eq. (\ref{3.28}) gives the absorption condition for $W=E_z$ in the form

$$\left(\frac{\partial E_z}{\partial x} - \frac{1}{c} \frac{\partial E_z}{\partial t} - \frac{c}{2} \frac{\partial B_x}{\partial y} \right)_{x = 0} = 0. \label{3.31}$$

Similar conditions can readily be deduced for an H-polarization wave.
Let us now write finite-difference approximations for the absorption conditions. For Eq. (\ref{3.27}) the difference scheme has the form

$$(E_z)^{n+1}_{0,j,k+1/2} = (E_z)^n_{1,j,k+1/2} + \frac{c\tau - h_x}{c\tau+h_x} [(E_z)^{n+1}_{1,j,k+1/2} - (E_z)^n_{0,j,k+1/2}] . \label{3.32}$$

Write the second-order scheme for Eq. (\ref{3.28}) assuming that $h_x=h_y=h_z=h$:

$$\begin{aligned}
(E_{z})^{n+1}_{0,j,k+1/2} &=& -(E_{z})^{n-1}_{1,j,k+1/2} + \frac{c \tau - h}{c \tau + h} [(E_{z})^{n+1}_{1,j,k+1/2} + (E_{z})^{n-1}_{0,j,k+1/2}] \nonumber \\
&+& \frac{2 \, h}{c \tau + h} [(E_{z})^{n}_{0,j,k+1/2} + (E_{z})^{n}_{1,j,k+1/2}] + \nonumber \\ &+&
\frac{(c \, \tau)^{2}}{c \tau + h} ((E_{z})^{n}_{0,j+1,k+1/2} - 2 (E_{z})^{n}_{0,j,k+1/2} + \nonumber \\
&+& (E_{z})^{n}_{0,j-1,k+1/2} + (E_{z})^{n}_{1,j+1,k+1/2} - 2 (E_{z})^{n}_{1,j,k+1/2} + \nonumber \\
&+& (E_{z})^{n}_{1,j-1,k+1/2} + (E_{z})^{n}_{0,j,k+3/2} - 2 (E_{z})^{n}_{0,j,k+1/2} + \nonumber \\
&+& (E_{z})^{n}_{0,j,k-1/2} + (E_{z})^{n}_{1,j,k+3/2} - 2 (E_{z})^{n}_{1,j,k+1/2} + \nonumber \\
&+& (E_{z})^{n}_{1,j-1,k-1/2}). \label{3.33}
\end{aligned}$$

For a two-dimensional case condition, Eq. (\ref{3.29}), is approximated by the scheme

$$\begin{aligned}
(E_{z})^{n+1}_{0,j} &=& (E_{z})^{n}_{1,j} + \frac{c \tau - h}{c \tau + h} ((E_{z})^{n+1}_{1,j} - (E_{z})^{n}_{0,j}) - \nonumber \\ &&
\frac{c}{2 (c \tau + h)} ((B_{x})^{n+1/2}_{0,j+1/2} - (B_{x})^{n+1/2}_{(),j-1/2} + \nonumber \\
&& (B_{x})^{n+1/2}_{1,j+1/2} - (B_{x})^{n+1/2}_{1,j-1/2}) . \label{3.34}
\end{aligned}$$

A somewhat different approach to the conditions at an open boundary segment was used by Weiland in the BCI code modeling axially symmetrical fields

\cite{51} and the TBCI code modeling azimuthally nonuniform oscillation modes \cite{52,53} Weiland considers a particular case of a wave propagating along an axially symmetrical tube and assumes that the field at a point z_0 at a time T is the same as that at a point z_0-ct at a time $t-\tau$, so that the B_ϕ component is approximated by

\begin{equation}
(B_{\phi})^{n+1/2}_{i-1/2,k} = (B_{\phi})^{n-m+1/2}_{i-1/2,k-1} ,
\quad mc\tau = h_{z} . \label{3.35}
\end{equation}

The equation approximating the E_r component will then take the form

\begin{equation}
(E_{r})^{n+1}_{i-1/2,k} = (E_{r})^{n}_{i-1/2,k} + \frac{\tau}{m \,h_{z}}
[(B_{\phi})^{n+1/2}_{i-1/2,k-1} - (B_{\phi})^{n-m+1/2}_{i-1/2,k-1}] . \label{3.36}
\end{equation}

The computation by this scheme requires storing an intermediate information for m layers. In deriving the finite-difference equations Weiland used the finite integration method where the volume or area (in a two-dimensional case) of the grid cell is considered as an element for integration. The finite-difference equations constructed in this case are the same as those we derived for the inner point of the grid. In the methodological respect, this approach is interesting in view that, when integrating for a near boundary element, one may introduce into the finite-difference scheme a factor equal to the cell volume (area) occupied by the field. Weiland, however, uses only a simplified version of this method where he distinguishes outer cells (occupied by metal) and inner cells (completely occupied by field) and sorts the near-boundary cells by running out a diagonal through a rectangular cell. Thus, if this diagonal approximates well the inclined segment of the boundary, only a correction factor of two may appear in the equations.

The construction of a finite-difference scheme of the second-order accuracy for the approximation of the boundary conditions for a boundary contour consisting of second-order curves was completely automated by Ivanov \cite{49} in an applied program package named MAXWELL-T For the axially symmetrical case, the boundary conditions have the form

\begin{eqnarray}
(E_{r})^{n+1}_{i-1/2,k} &=& (E_{r})^{n}_{i-1/2,k} +
\tau \sum_{j} P_{j} (B_{\phi})^{n+1/2}_{j} , \label{3.37} \\
(E_{z})^{n+1}_{i,k-1/2} &=& (E_{z})^{n}_{i,k-1/2} +
\tau \sum_{j} P_{j} (B_{\phi})^{n+1/2}_{j} , \label{3.38}
\end{eqnarray}

where P_j are the difference template coefficients and the whole summation is carried out by all grid points neighboring the near-boundary points.

This approach requires preliminarily marking the types of near-boundary points and storing the P_j coefficient for them.

Library of Classes for Object-Oriented Field Solver Code

The classes library which reflects the structure of the complete

PIC-code architecture is made up of base classes, visualization classes interface and contemplation classes. Note should be made that the library contains different implementations of the same concepts (classes): vftrix, matr, array, vector, etc. This has been done in order to provide for a search for the most optimal program implementation both during the creation stage of certain classes and while developing a method to approach them.

BASE CLASSES are considered here as class hierarchies describing the numerical model of a specific task. To new base data types should also belong data classes (structures) describing common base concepts: coordinate, axis, vector, etc.

VISUALIZATION CLASSES are required for graphical specification of the computational region, checking of the correctness of mesh generation and specification of the boundary conditions of the task, as well as graphical representation of the results.

INTERFACE CLASSES serve to create an advanced interactive data input under MS WINDOWS. Application of this approach helps to create a user-friendly interface. The entire class hierarchy is based on a standard class TDialog OWL, Borland Library.

COMPUTATIONAL CLASSES are developed to implement various numerical algorithms useful for solving electrostatic tasks. Thus, the emphasis in this report is made on developing straightforward method of solving Poisson equation.

\subsection{Base Classes}

Development of new data types (structures and classes) requires a detailed description of the inner structure of the abstract class as well as the interface of interaction of the objects of a given class with other data types in a specific implementation of numerical algorithms. The process of writing the base classes is, however, iterational. This is due to the fact that in object-oriented programming it is difficult to readily determine the final structure of the new type data and still more difficult it is to describe all possible methods of the interaction of objects of a given class with the other program data. Classification of the library classes used here allows us to separate major (base) data (structure) with further detailed development of the structure and methods. These classes form the ``lower'' level of the library.

\begin{verbatim}
struct:
    AXISTRUCT - describes the type of the coordinate axis
            for physical values
    pixel - determines 2D coordinates on the display
    vect - determines the physical value of the 3D coordinate
    index - determines the 2D index
    para - determines the 2D coordinates (TEMPLATE data type)
    point - determines the 2D coordinates
    array -
    TVector -

```
    TCell -
    TPhPoint -
class:
    axis -
    TBoundary -
    TElectrod -
    TAbstrRegion -
    TGeom -
    TGeomFrameWork -
    TAbstrMesh -
    TAbstrModel -
    TAbstrPhysVal -
    TEulerVal -
    TLagrangeVal -
    THistoricVal -
    TGridVal -
    TFunctionVal -
    TDoubleGridVal -
    TBeamVal -
\end{verbatim}
```

As a result of detailed elaboration of specific conditions of the numerical model each of the classes enumerated above gives rise to new type data inheriting both the data structure and the interactive behavior of the parent classes. These classes fill the upper level of the base classes describing the numerical model.

The following class hierarchy is used to describe the mesh type of the computational region:\vspace{12pt}

```
\noindent
\begin{minipage}[c]{5true cm}
\unitlength=1mm
\special{em:linewidth 0.4pt}
\linethickness{0.4pt}
\begin{picture}(45.00,32.00)
\put(0.00,24.00){\framebox(35.00,8.00)[]{}}
\put(35.00,28.00){\line(1,0){10.00}}
\put(10.00,00.00){\line(0,1){24.00}}
\put(45.00,00.00){\line(0,1){28.00}}
\put(10.00,0.00){\line(1,0){35.00}}
\put(10.00,8.00){\line(1,0){35.00}}
\put(10.00,16.00){\line(1,0){35.00}}
\put(10.00,28.00){\makebox(0,0)[lc]{\tt axis}}
\put(18.00,20.00){\makebox(0,0)[lc]{\tt TGeomAxis}}
\put(18.00,12.00){\makebox(0,0)[lc]{\tt TPhysAxis}}
\put(18.00,4.00){\makebox(0,0)[lc]{\tt TTimeAxis}}
\end{picture}
\end{minipage}\hfill \begin{minipage}[c]{10true cm}
This hierarchy is meant for the introduction into the numerical
model of objects subsequently used as components of physical values. Their
usage will simplify the display of physical values.\end{minipage}\\[2ex]
\begin{minipage}[c]{50true mm}
\unitlength=1mm
```

\special{em:linewidth 0.4pt}
\linethickness{0.4pt}
\begin{picture}(50.00,45.00)
\put(0.00,32.00){\framebox(40.00,8.00)[cc]{\tt TGeomFrameWork}}
\put(40.00,36.00){\line(1,0){10}}
\put(10.00,00.00){\line(0,1){32.00}}
\put(50.00,00.00){\line(0,1){36.00}}
\put(10.00,0.00){\line(1,0){40.00}}
\put(10.00,8.00){\line(1,0){40.00}}
\put(10.00,16.00){\line(1,0){40.00}}
\put(10.00,24.00){\line(1,0){40.00}}
\put(20.00,28.00){\makebox(0,0)[lc]{\tt TGeomXY}}
\put(20.00,20.00){\makebox(0,0)[lc]{\tt RGeomRZ}}
\put(20.00,12.00){\makebox(0,0)[lc]{\tt TGeomRTh}}
\put(20.00,4.00){\makebox(0,0)[lc]{\tt TGeomX1X2}}
\end{picture}
\end{minipage}\hfill \begin{minipage}[c]{10true cm}
The name of this group implies that it deals with determination of the geometry type of a numerical model. These classes should be added by {\tt TElectrod} and {\tt TGeom} for specification of a random computational region.
\end{minipage}\\[2ex]

This hierarchy of classes is used to describe the type of the computational region:

\noindent
\unitlength=1mm
\special{em:linewidth 0.4pt}
\linethickness{0.4pt}
\begin{picture}(55.00,40.00)
\put(0.00,24.00){\framebox(35.00,8.00)[]{}}
\put(35.00,28.00){\line(1,0){10.00}}
\put(10.00,8.00){\line(0,1){16.00}}
\put(45.00,8.00){\line(0,1){20.00}}
\put(10.00,16.00){\line(1,0){35.00}}
\put(10.00,8.00){\line(1,0){35.00}}
\put(45.00,12.00){\line(1,0){8.00}}
\put(7.00,28.00){\makebox(0,0)[lc]{\tt TAbstrRegion}}
\put(18.00,0.00){\line(0,1){8.00}}
\put(53.00,0.00){\line(0,1){12.00}}
\put(18.00,0.00){\line(1,0){35.00}}
\put(16.00,20.00){\makebox(0,0)[lc]{\tt TSampleRgn}}
\put(16.00,12.00){\makebox(0,0)[lc]{\tt TRectangleRgn}}
\put(26.00,4.00){\makebox(0,0)[lc]{\tt TRectRgnRZ}}
\end{picture}\vspace{12pt}

Classes hierarchy of the mesh of the computational region is needed to describe the type of the computational region, the knods template, etc. The abstract class TAbstrMesh incorporates two other abstract classes.
\newpage
\begin{figure*}[t]
\unitlength=1mm

```
\special{em:linewidth 0.4pt}
\linethickness{0.4pt}
\begin{picture}(140.00,60.00)
\put(0.00,42.00){\framebox(45.00,8.00)[cc]{}}
\put(0.00,54.00){\framebox(45.00,8.00)[cc]{}}
\put(45.00,58.00){\line(1,0){20.00}}
\put(45.00,46.00){\line(1,0){20.00}}
\put(65.00,46.00){\line(0,1){12.00}}
\put(65.00,52.00){\line(1,0){10.00}}
\put(75.00,48.00){\framebox(45.00,8.00)[cc]{}}
\put(120.00,52.00){\line(1,0){10.00}}
\put(90.00,40.00){\line(0,1){8.00}}
\put(130.00,40.00){\line(0,1){12.00}}
\put(90.00,40.00){\line(1,0){40.00}}
\put(130.00,44.00){\line(1,0){10.00}}
\put(100.00,32.00){\line(1,0){40.00}}
\put(100.00,24.00){\line(1,0){40.00}}
\put(100.00,16.00){\line(0,1){24.00}}
\put(140.00,16.00){\line(0,1){28.00}}
\put(90.00,16.00){\line(1,0){50.00}}
\put(130.00,12.00){\line(1,0){10.00}}
\put(90.00,8.00){\line(0,1){8.00}}
\put(130.00,8.00){\line(0,1){8.00}}
\put(100.00,0.00){\line(0,1){8.00}}
\put(100.00,0.00){\line(1,0){40.00}}
\put(140.00,0.00){\line(0,1){12.00}}
\put(90.00,8.00){\line(1,0){40.00}}
\put(5.00,58.00){\makebox(0,0)[lc]{\tt TAbstrRegions}}
\put(5.00,46.00){\makebox(0,0)[lc]{\tt TGeomFrameWork}}
\put(80.00,52.00){\makebox(0,0)[lc]{\tt TAbstrMesh}}
\put(95.00,44.00){\makebox(0,0)[lc]{\tt TMesh2D}}
\put(105.00,36.00){\makebox(0,0)[lc]{\tt TRegularMesh}}
\put(105.00,28.00){\makebox(0,0)[lc]{\tt TCellListMesh}}
\put(105.00,20.00){\makebox(0,0)[lc]{\tt TNonRectMesh}}
\put(95.00,12.00){\makebox(0,0)[lc]{\tt TMesh1D}}
\put(105.00,4.00){\makebox(0,0)[lc]{\tt TSurfaceMesh}}
\end{picture}
\end{figure*}
```

Classes describing the numerical model make use of the data of the computational region, boundary conditions, dimensions and symmetry, as well as the type of the \linebreak task/problem.

```
\begin{figure*}[bth]
\unitlength=1mm
\special{em:linewidth 0.4pt}
\linethickness{0.4pt}
\begin{picture}(140.00,50.00)
\put(100.00,0.00){\line(0,1){8.00}}
\put(150.00,0.00){\line(0,1){12.00}}
\put(130.00,12.00){\line(1,0){20.00}}
\put(100.00,0.00){\line(1,0){50.00}}
\put(105.00,4.00){\makebox(0,0)[lc]{\tt TElectrostaticModel}}
\put(90.00,8.00){\line(0,1){16.00}}
```

```
\put(130.00,8.00){\line(0,1){20.00}}
\multiput(90.00,8.00)(0,8){2}{\line(1,0){40.00}}
\put(120.00,28.00){\line(1,0){10.00}}
\put(95.00,12.00){\makebox(0,0)[lc]{\tt TSimple2DModel}}
\put(95.00,20.00){\makebox(0,0)[lc]{\tt TPiClassicModel}}
\put(75.00,24.00){\framebox(45.00,8.00)[cc]{}}
\put(80.00,28.00){\makebox(0,0)[lc]{\tt TAbstrModel}}
\put(65.00,28.00){\line(1,0){10.00}}
\put(65.00,10.00){\line(0,1){36.00}}
\multiput(45.00,10.00)(0,12){4}{\line(1,0){20.00}}
\multiput(0.00,6.00)(0,12){4}{\framebox(45.00,8.00)[cc]{}}
\put(5.00,46.00){\makebox(0,0)[lc]{\tt TAbstrRegions}}
\put(5.00,34.00){\makebox(0,0)[lc]{\tt TAbstrProcess}}
\put(5.00,22.00){\makebox(0,0)[lc]{\tt TAbstrPhysVal}}
\put(5.00,10.00){\makebox(0,0)[lc]{\tt TGeomFrameWork}}
\end{picture}
\end{figure*}

\subsection{Visualization Classes}

This part of the library is based on the GUI-interface of the OWL Library
compiled by Borland. The major classes developed to display in the child
windows are based on the abstract class {\tt TBasePlotswin}.

\noindent
\unitlength=1mm
\special{em:linewidth 0.4pt}
\linethickness{0.4pt}
\begin{picture}(70.00,45.00)
\put(25.00,0.00){\line(0,1){16.00}}
\put(65.00,0.00){\line(0,1){20.00}}
\multiput(25.00,0.00)(0,8){2}{\line(1,0){40.00}}
\put(55.00,20.00){\line(1,0){10.00}}
\put(15.00,16.00){\line(0,1){16.00}}
\put(55.00,16.00){\line(0,1){20.00}}
\multiput(15.00,16.00)(0,8){2}{\line(1,0){40.00}}
\put(40.00,36.00){\line(1,0){15.00}}
\put(0.00,32.00){\framebox(40.00,8.00)[cc]{}}
\put(5.00,36.00){\makebox(0,0)[lc]{\tt TBasePlotsWin}}
\put(20.00,28.00){\makebox(0,0)[lc]{\tt TSurfacePlot}}
\put(20.00,20.00){\makebox(0,0)[lc]{\tt TLinesPlot}}
\put(30.00,12.00){\makebox(0,0)[lc]{\tt TRegionPlot}}
\put(30.00,4.00){\makebox(0,0)[lc]{\tt TPointsPlot}}
\end{picture}
%\end{figure*}

\subsection{Dialog Classes}

  Dialog classes are necessary to create a friendly environment for the
user working with the software package. In view of the fact that the project
under developments meant for operation under MS WINDOWS we make use of the
OWL Library, v.2 compiled by Borland.

  The report contains the child classes from the class {\tt TDialog} OWL
```

Borland Library. They were developed for input and variation of data required to initialize new objects. In addition to the input dialog windows the library contains windows for systems messages and information windows.

{\bf Windows informing of the version and authors of the code}

```
\begin{figure*}[h]
\unitlength=1mm
\special{em:linewidth 0.4pt}
\linethickness{0.4pt}
\begin{picture}(140.00,20.00)
\put(85.00,0.00){\line(0,1){8.00}}
\put(125.00,0.00){\line(0,1){12.00}}
\put(85.00,0.00){\line(1,0){40.00}}
\put(115.00,12.00){\line(1,0){10.00}}
\put(0.00,0.00){\framebox(45.00,8.00)[cc]{\tt ProjectRCVersion}}
\put(45.00,4.00){\line(1,0){40.00}}
\put(75.00,8.00){\framebox(40.00,12.00)[cc]{}}
\put(90.00,4.00){\makebox(0,0)[lc]{\tt TAboutVersDlg}}
\put(95.00,15.00){\makebox(0,0)[cb]{\tt OWL-class}}
\put(95.00,13.00){\makebox(0,0)[ct]{\tt TDialog}}
\end{picture}
\end{figure*}
```

The dialog window ({\tt TInfoDlg1}) and the data structure ({\tt TInfoTrs1}) are to output information data on reserves of the computer, on computational model, etc. through the window of the class {\tt TInfoDlg1}:

```
\begin{figure*}[h]
\unitlength=1mm
\special{em:linewidth 0.4pt}
\linethickness{0.4pt}
\begin{picture}(140.00,20.00)
\put(85.00,0.00){\line(0,1){8.00}}
\put(125.00,0.00){\line(0,1){12.00}}
\put(85.00,0.00){\line(1,0){40.00}}
\put(115.00,12.00){\line(1,0){10.00}}
\put(0.00,0.00){\framebox(45.00,8.00)[cc]{\tt TInfoTrs1}}
\put(45.00,4.00){\line(1,0){40.00}}
\put(75.00,8.00){\framebox(40.00,12.00)[cc]{}}
\put(90.00,4.00){\makebox(0,0)[lc]{\tt TInfoDlg1}}
\put(95.00,15.00){\makebox(0,0)[cb]{\tt OWL-class}}
\put(95.00,13.00){\makebox(0,0)[ct]{\tt TDialog}}
\end{picture}
\end{figure*}
```

The dialog window ({\tt TRegionBndsDlg}) and the data structure ({\tt TRegionBndsTrsl}) are to input data into the class {\tt TAbstrRegion} and its child classes:

```
\noindent
\unitlength=1mm
\special{em:linewidth 0.4pt}
\linethickness{0.4pt}
```

```
\begin{picture}(140.00,25.00)
\put(85.00,0.00){\line(0,1){8.00}}
\put(125.00,0.00){\line(0,1){12.00}}
\put(85.00,0.00){\line(1,0){40.00}}
\put(115.00,12.00){\line(1,0){10.00}}
\put(0.00,0.00){\framebox(45.00,8.00)[cc]{\tt TRegionBndsTrs1}}
\put(45.00,4.00){\line(1,0){40.00}}
\put(75.00,8.00){\framebox(40.00,12.00)[cc]{}}
\put(90.00,4.00){\makebox(0,0)[lc]{\tt TRegionBndsDlg}}
\put(95.00,15.00){\makebox(0,0)[cb]{\tt OWL-class}}
\put(95.00,13.00){\makebox(0,0)[ct]{\tt TDialog}}
\end{picture}
%\end{figure*}

  {\bf A service class to check the input information in the dialog window} \linebreak
({\tt TRegionBndsDlg})

\noindent
\unitlength=1mm
\special{em:linewidth 0.4pt}
\linethickness{0.4pt}
\begin{picture}(140.00,24.00)
\put(10.00,0.00){\line(0,1){8.00}}
\put(60.00,0.00){\line(0,1){12.00}}
\put(10.00,0.00){\line(1,0){50.00}}
\put(50.00,12.00){\line(1,0){10.00}}
\put(0.00,8.00){\framebox(50.00,12.00)[cc]{}}
\put(15.00,4.00){\makebox(0,0)[lc]{\tt TAboutVersDlg}}
\put(20.00,15.00){\makebox(0,0)[cb]{\tt OWL-class}}
\put(20.00,13.00){\makebox(0,0)[ct]{\tt TFilterValidator}}
\end{picture}
%\end{figure}


\subsection{Computing Classes}

  Due to the fact that during the project implementation various
numerical algorithms are used to compute the same physical values and also to
reach the main target of the project -- REUSABILITY -- we attempted to
separate computational classes from base classes. Development of the class
hierarchy with inheritance is feasible only for families of certain numerical
algorithms having similar structure and identical sets of input and output
parameters. In this case the base class is used for late binding and
replacement of the numerical algorithms on the run. The library contains only
one computational class ({\tt TPoisson}) which is a prototype of the approach
described.

The service class providing links between the computational module and
the object-oriented base classes is to be made up by the classes-descendants
{\tt TAbstrProcess}.

\begin{verbatim}
   TAbstrProcess
    TDetermPrcs
```

TMomentPrcs
TStaticPrcs - TElectrostaticPrcs
TDynamicPrcs
THistoricPrcs
\end{verbatim}

However, simple and laconic algorythms can be fully encapsulated into respective classes of our main structure, e. g. the FFT algorithm can be implemented both in a member-function of the class {\tt TGridVal: Spectrum} ( ), and in a member-function of the class {\tt THarmonic Process::Spectrum (TPhysVal*)}.

\section{PiCsim3 code for Computation of Poisson Equation}
\subsection{Goal of Program}

To carry out testing of the library of OO classes we developed the PiCsim3 code based on the OO-PIC classes hierarchy. This software furnishes an illustration of interactions between the classes developed. Using the calculations of the test problems of the solution of Poisson equation as an example it was shown that the version developed is capable of doing computations of the problems in simple geometries.

The software is run under Windows 3.1 and Win32, therefore the diskette has both projects for the Compiler Borland C++ v.4.5 (ps3$_-$work.ide and ps$_-$32.ide). In the 32-register version there should be determined the global variable $_-$WIN$_-$32$_-$ to inhibit usage of the functions of the Toolhelp library, such as SysMemoryInfo ( ), HeapInfo ( ), GlobalCompact ( ).

The software program satisfies the requirements set for the codes with user-friendly graphical interfaces. This program makes use of the standard Windows MDI implemented as the Borland`s OWL.

Correct performance of the program is possible provided it is run on the 486DX computers with 8MB on-line memory.

\subsection{Installation of Program onto Hard Disk}

To ensure successful installation of the software product onto your hard disk the latter has to possess no less than 15 MB free space. In view of the fact that the version provided is of a de-bug character we chose not to develop the installation program unrolling the code of fields computation. We applied a simple way of transportation of the self-unpacking archives of the source modules of OO-PIC library of PiCsim3 and projects for the compiler Borland C++ v.4.5.

To port the program onto your hard disk is easy. You are to open a PICSIM catalogue on a disk with no less than 10 MB free space, e.g.:

\begin{verbatim}
C:\>MD PICSIM
\end{verbatim}

Then insert the work diskette into the disk drive and copy the self-unpacking archives with the program into the catalogue C:{$\backslash$}PICSIM

\begin{verbatim}
C:\>A:
A:\COPY PICSIM_A.EXE C:\PICSIM\PICSIM_A.EXE
\end{verbatim}

Upon transporting the archives on the disk of your computer take the diskette out. The next preparation step is to unpack the archive file. To do so the user is to transfer to the work disk in the catalogue PICSIM using the command

\begin{verbatim}
A:\>C
C:\>CD PICSIM
C:\PICSIM\>
\end{verbatim}

Then print in the command line the command

\begin{verbatim}
C:\PICSIM\>PICSIM_A.EXE
\end{verbatim}

Upon completing the archives unpacking your disk should display the directory tree:

\begin{verbatim}
C:\
|--------PICSIM\
|--------INC
|--------OBJ
|--------EXE
|--------HLP
|--------CPP
|---------BASE
|---------INTERFACE
|---------MAIN
|---------PLOTS
|---------SERVICE
\end{verbatim}

It is essential to make sure that all the files necessary for successful compilation of the program PiCsim3 have been duly unpacked. Check the file lists using the work subdirectiries against the lists cited in the report.

When unpacking has been finished you may delete the archive file from the hard disk, as its duplicate is on the work disk now.

\subsection{Working with Program}

The program should be started from OO WINDOWS, e.g. using the command:

\begin{verbatim}
C:\PICSIM\EXE\>WIN PICSIM3.EXE
\end{verbatim}

Your computer will display the main window of the program. Due to implementation of the MDI interface the main function creates the object of the {\tt TPICsimApp} class inheriting all the properties from the class {\tt Tapplication} of the class library OWL which calls the Run ( ) method. The class {\tt TPiCsimApp} initializes the main window of the program and, in accordance with the rules of building the MDI interface, these are two objects:

\begin{verbatim}
    TPiCsimFrameWin - inheriting from clan TDecoratedFrame
    TPiCsimClientWin - inheriting from clan TMDIClient
\end{verbatim}

In initializing the object of the class {\tt TPiCsimFrameWin} (SetupWindow ( )) there creates a window for problems and other managing elements of the window. Additionally the object checks the sequence of messages under Windows and calls Winhelp by pressing the F1 key, and when the message under WM$_-$NEWMODEL and WM$_-$CLOSEMODEL are input it changes the manner of operation with the computation model. This approach allows the user to change the main problem window at run time (menu, ControlBar, StatusBar, ...). For this purpose in the message processor under OCWindows use is made of the client substitution technique enabling, by simple means, addition of new models with their own user-friendly interfaces. A necessary condition for this approach is the inheritance of the model class from the class {\tt TPiCsimClientWin}.

Thus, in starting the program the computer displays the main window with the menu, status-line and control panel. This window is generated for the user to select the computation model. During implementation of the test program we used, as an example, the electrostatic model. Having selected pictogram $<\phi>$ you initialize the electrostatic model under your own window tuned to operate with this model type. The main window, in addition to the $<\phi>$ pictogram, has the $<?>$ pictogram which generates the message Windows and the F1 key which calls Winhelp. The $<\rm C>$ pictogram displays the information window ABOUT, and pressing the pictogram $<\rm Q>$ causes quitting of the program.

Having selected the electrostatic model, the program replaces the main window of the problem with its own menu by the control panel and status-line. As shown by experience of working with software complexes under Windows the simplest method of running the program is to use pictograms from the control panel. Therefore, developing the working scenario we paid a lot of attention to underlining the basic action of the control panel pictograms

\begin{tabular}{ccl}
  $<{\rm N}>$ & -- & opening new model\\
  $<{\rm O}>$ & -- & opening model stored on disk\\
  $<{\rm S}>$ & -- & saving working model\\
  $<{\rm R}>$ & -- & starting computation of potential\\
  $<{\rm St}>$ & -- & computing electrostatic fields\\
  $<{\rm Nx}>$ & -- & next computation step\\
  $<{\rm T}>$ & -- & opening graphics window with the results of potential ϕ(r,z)\\
  & & displayed as surface\\
  $<{\rm G}>$ & -- & opening graphics window with display of isolines of equal \\
  & & electric field strength\\
  $<{\rm E}>$ & -- & exit into main window of program selecting model type.

\end{tabular}

\subsection{Test Results}

The program allows test runs based on computation of the electric field potential and strength at a given distribution of the charge density under boundary conditions. Correctness of operation of the program was determined by comparing the initial charge distribution with the distribution of the charge density reconstructed from the calculated distribution of the electric field strength. The results we obtained in these tests were generelly good, since the error of the reconstructed charge density was of the order (h^2), which corresponds to the accuracy of the algorithm used.

Additionally, calculations by our program were compared with the data obtained by other authors \cite{66, 67}.

\subsubsection{Test Tasks}
Numerical model for the test tasks are formulated in 3 files: ``test1.mod", ``test2.mod" and ``test3.mod". To load the test task into the program and carry out the computation it is necessary to do the following:
\def\labelitemi{--}
\begin{itemize}
\item put the program on the run and select an electrostatic version of the task;
\item open the model by pushing the button $<{\rm O}>$;
\item choose the file ``test.mod", N 1, 2, 3;
\item push the button $<{\rm R}>$ and solve the Poisson equation for the given conditions; and
\item visualize the windows with graphics information on distridution of the charge density, the electric field potential and strength, having selected in the menu the items = Plots/2d Plots/Space Charge (Potential or Restore Source).
\end{itemize}
The program has the following options for controlling the graphics windows:
\begin{itemize}
\item scaling of the graphics information in the window and its frames;
\item addition of standard objects (text, boundary, of the calculation region and calculation mesh, symmetry axes, ...) to the information;
\item withdrawal of the above objects from the window and their subsequent editing;
\item while selecting ``edit" for the main physical value visualized graphically in the window the screen displays the dialog window within which one can define orientation of the surface with respect to the look-up plane, etc.
\end{itemize}

\subsubsection{Test run 1}

The most completely studied charge distribution was that of a single uniform sphere of charge. The data for the first run is set-up by the file ``test1.mod" and correspond to the data for the test run 1 reported in \cite{66}.

```
\begin{figure}[h]
\centering
\unitlength=0.24pt
\begin{picture}(1298,1003)
\put(0,1003){\special{em:graph test01.pcx}}
\end{picture}
\caption{}
\end{figure}
```

Figure 2 shows the input charge distribution and the calculated fields and associated errors. The calculation by our program agree well with the data obtained by Beard and Hockney using the POT 4A program \cite{66}.

```
\subsubsection{Test run 2}
```

In a more stringent test, we checked the accuracy of the program using an input charge density consisting of two equally but oppositely charged spheres of equal radii. The data for this run is set-up by the file ``test2.mod" and corresponds to the data for the test run 2 from Ref. \cite{66}.

```
\begin{figure}[t]
\centering
\unitlength=0.24pt
\begin{picture}(1298,1003)
\put(0,1003){\special{em:graph test02.pcx}}
\end{picture}
\caption{}
\end{figure}
```

These results are shown in Fig. 3. They agree well with the results obtained using the POT 4A program \cite{66}.

```
\subsubsection{Test run 3}
```

When using the test deck to verify our program the test deck will generate a potential distribution and obtain the source function from this potential. The potential is stored to provide a check on the accuracy of the solution. The generated potential can be an analytic function of x and y and is set-up by the file ``test3.mod." as

```
\begin{equation}
\phi(x,y)=X(x)Y(y).
\end{equation}
```

The form for the analytic depends on the boundary conditions is

```
\[ X(x)=\left\{\begin{array}{ll} \sin\Big(\displaystyle\frac{3\pi x}
{N\Delta x}\Big) & \rm for~ MBCX = 1,\\
\cos\Big(\displaystyle\frac{4\pi x}{N\Delta x}\Big) &
\rm for~ MBCX = 2,\end{array}\right. \]
\[ Y(y)=\left\{\begin{array}{ll} \sin\Big(\displaystyle\frac{3\pi y}
{N\Delta y}\Big) & \rm for~ MBCY = 1,\\
\cos\Big(\displaystyle\frac{4\pi y}{N\Delta y}\Big) &
\rm for~ MBCY = 2.\end{array}\right. \]

\newpage
```

\tolerance 5000
\begin{thebibliography}{nn}
\bibitem{1} L.D. Landau and E.M. Lifshitz, {\it The Field Theory}, Nauka, Moscow 1967.
\bibitem{2} L.D. Landau and E.M. Lifshitz, {\it The Medium Electrodynamics}, Gostekhizdat, Moscow 1957.
\bibitem{3} V.V. Nikolsky, {\it Electrodynamics and Radiowave propagation}, Nauka, Moscow 1973.
\bibitem{4} V.V. Nikolsky, {\it Variational Methods for Problems of Electrodynamics}, Nauka, Moscow 1967.
\bibitem{5} W.K. Panofsky and M. Philips, {\it Classical Electricity and Magnetism}, Addison-Wesley Publishing Company, Inc., Cambridge 1960.
\bibitem{6} S.R. de Groot and L.G. Suttorp, {\it Foundation of Electrodynamics}, North-Holland Publishing Company, Amsterdam 1972.
\bibitem{7} M.M. Bredov, V.V. Rumiantzev, and I.N. Toptygin, {\it The Classic Electrodynamics}, Nauka, Moscow 1985.
\bibitem{8} N.A. Khizhnyak, {\it Integral Equations of Macroscopic Electrodynamics}, Naukova dumka, Kiev 1986.
\bibitem{9} A.N. Kravchenko, {\it Boundary Characteristics in Electodynamics Problems}, Naukova dumka, Kiev 1989.
\bibitem{10} N.I. Doynikov, {\it Problem statement for numerical analysis of nonlinear magnetic system fields}, Rep. NIIEFA OB-8, Leningrad 1976.
\bibitem{11} C.W. Trowbridge, {\it Three-dimensional field computation}, Chilton, 1981, Rep. RL-81-054.
\bibitem{12} C.F. Iselin, {\it Review of recent developments in magnet computations}, Gene\-va, 1981, Rep. CERN/SPS/81-7.
\bibitem{13} R.J. Lary and L.R. Turner, ``Survey of Eddy Current Programs", IEEE Trans. Magn., {\bf 19:6} (1983) 2474.
\bibitem{14} T. Torchanoff, ``Survey of numerical methods in field calculations", IEEE Trans. Magn., {\bf 20:5} (1984) 1912.
\bibitem{15} M. Liese, K. Lenz, K. Senke, and J. Spiegl, ``Comparison of vector potential methods and true three-dimensional magnetostatic field calculations", IEEE Trans. Power Appar. Syst., {\bf PAS-103:6} (1984) 1339.
\bibitem{16} L.P. Bitsadze, ``Some solution of electric elasticity equation", In: {\it Some Problems of Elasticity Theory}. IPM Trans. (Tbilisi), {\bf 16} (1985) 20.
\bibitem{17} Zh.A. Rukhadze, ``About one problem of magnetothermoelasticity", In: {\it Some Problems of Elasticity Theory}. Ibid., 182.
%\bibitem{18} Šà ¢æ®¢ ,.,. ^-â¥£à «ì-ë¥ ãà ¢-¥-¨© ¢ § ¤ ç å ¤¨äà ªæ¨¨//, ª-.
% ,ëç¨á«¨â¥«ì-ë¥ ¬¥â®¤ë ¨ ¯à®£à ¬¬¨à®¢ -¨¥. 'à. Œƒ", 1966, ¢ë¯.
% 5.-'.260-293.
%\bibitem{19} Poggio A.J., Miller E.K. Solution three-dimensional scattering
% problems for electromagnetic waves based on integral equations
% //In: Computer Techniques for Electromagnetics. v.7, Pergamon
% Press, 1973.
\bibitem{20} O.V. Tozoni, {\it Secondary Source Method in Electrotechnics}, Energia, Moscow 1975.
\bibitem{21} I.D. Maergoiz, {\it Iterational Methods for Calculation Static Fields in Inhomogenous, Anisotropic and Nonlinear Media}, Naukova dumka, Kiev 1979.
\bibitem{22} M.G. Aleksandrov, A.N. Belyanin, V. Brukner, et al. {\it Computer Calculation of Electric Circuits and Electromagnetic Fields}, Radio

i svyaz, Moscow 1983.
\bibitem{23} P.A. Kurbatov and S.A. Arinchin, {\it Numerical Calculation of Electromagnetic Fields}, Energoatomizdat, Moscow 1984.
\bibitem{24} E.S. Koletchitsky, {\it Electric Field Calculation for High-Voltage Devices}, Energoatomizdat, Moscow 1983.
\bibitem{25} C.A. Brebbia, {\it Boundary Element Method for Engineers}, Penthch Press, London and Halstead Press, New York, 1978.
\bibitem{26} C.A. Brebbia, J.C. Telles, and L.C. Wrobel, {\it Boundary Element Techniques. Theory and Applications in Engineering}, Springer-Verlag, Berlin, 1984.
\bibitem{27} T.V. Hromadka and C. Lai, {\it The Complex Variable Boundary Element Method in Engineering Analysis}, Springer-Verlag, Berlin, 1987.
\bibitem{28} Y. Ose, T. Takagi, H. Sano, and K. Miki, ``3-D Electron Optics Simulation Method for Cathode Ray Tubes", IEEE Trans. Magn., {\bf 24:1} (1988) 552.
\bibitem{29} A.R. Mitchell and R. Wait, {\it The Finite Element Method in Partial Differential Equations}, John Wiley and Sons, Chichester, 1977.
\bibitem{30} O.C. Zienkiewicz and K. Morgan,{\it Finite Elements and Approxima\-tion}, John Wiley and Sons, New York, 1983.
\bibitem{31} P. Ciarlet, {\it The Finite Element Method for Elliptic Problems}, Nort-Holland Publ. Company, Amsterdam, 1978.
\bibitem{32} L.J. Segerlind, {\it Applied Finite Element Analysis}, John Wiley and Sons, New York, 1976.
\bibitem{33} R.H. Gallager, {\it Finite Element Analysis. Fundamentals}, Pren\-tice Hall, New Jersey, 1975.
\bibitem{34} D.H. Norrie and G. de Vries, {\it An Introduction to Finite Element Analysis}, Aca\-demic Press, New York, 1978.
\bibitem{35} K.-J. Bathe and E.L. Wilson, {\it Numerical Methods in Finite Element Analysis}, Prentice Hall, London, 1976.
\bibitem{36} {\it Finite Element Handbook}, Ed. by Kardestuncer, McGraw-Hill, New York, 1987.
\bibitem{37} P.P. Silvester and R.L. Ferrari, {\it Finite Elements for Electrical Engineering}, University Press, Cambridge, 1983.
%\bibitem{38} ¢ ç¥¢ ,.‹. Œ¥â®¤ë «£¥¡àë «®£¨ª¨ ¢ ¬ â¥¬ â¨ç¥áª®© ä¨§¨¥¥//
% Š¨¥¢: ãª®¢ ¤ã¬ª, 1974.-259á.
\bibitem{39} K.D. Paulsen, D.R. Lynch, and J.W. Strohbehn, ``Three-Dimensional Finite, Boundary and Hybrid Element Solutions of the Maxwell Equations for Lossy Dielectric Media" IEEE Trans. on Microwave Theory and Tech., {\bf 36:4} (1988) 682.
%\bibitem{40} ®è «ì .‘. Œ®¤¥«¨à®¢ -¨¥ ¯ãçª®¢ § àï¦¥--ëå ç áâ¨æ//Œ.:
% â®¬¨§¤ â, 1979.-224á.
%\bibitem{41} Hockney R.W., Eastwood J.W. Computer Simulation Using
% Particles//Mc\-Graw-Hill, New York, 1981.-559p.
\bibitem{42} K.S. Yee, ``Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media", IEEE Trans. Antennas Propagat., {\bf AP-14} (1966) 302.
\bibitem{43} E.J. Horowitz, D.E. Shumaker, and D.V. Anderson, ``QN3D: A Three-Dimensional Quasi-neutral Hybrid Particle-in-Cell Code with Applications to the Tilt Mode Instability in Field Reserved Configurations", J. Comput. Phys., {\bf 84} (1989) 279.
\bibitem{44} V.D. Danilov, ``Numerical solution of problem of interaction between charged partical bunch and resonator", In.: {\it Mathematical Methods in Physics and Cybernetics}. Atomizdat, Moscow 1974, Vol. 4, p. 8.

\bibitem{45} V.D. Danilov, ``Relativistic electron beam model, accounting irradiation field", In.: {\it Mathematical Methods in Physics and Cybernetics}. Atomizdat, Moscow 1974, Vol. 5, p. 12.
\bibitem{46} A. Taflove and M.E. Brodwin, ``Numerical Solution of Steady-State Electromag\-netic Scattering Problems Using the Time-Dependent Maxwell's Equations", IEEE Trans. Microwave and Thech., {\bf MTT-8:23}, (1975) 623.
\bibitem{47} N.S. Bakhvalov, V.S. Bondarenko, E.P. Zhidkov, et al. {\it Numerical Solution Maxwell's Equations in Inhomogenuous Region with Moving à §àë¢®¬ in the Right Part}, Preprint OIYaI R-11-8981, 1975.
\bibitem{48} R. Holland, L. Simpson, and K.S. Kunz, ``Finite-Difference Analysis of EMP Coupling to Lossy Dielectric Structures", IEEE Trans. Electromagn. Compat., {\bf EMC-22:3}, (1980) 203.
\bibitem{49} V.Ya. Ivanov, {\t Algorithms and Applied Program Packages for Electrodynamics Problems in 2-D and 3-D Regions}, Rep. ATI-MOD 320/10, ITMVT AN SSSR, Novosibirsk branch, 1982.
\bibitem{50} M. Chapman and E.M. Waisman, ``A Noise Suppression Algorithm for the Numerical Solution of Maxwell's Equations", J. of Comput. Phys., {\bf 58}, (1985) 44.
\bibitem{51} T. Weiland, ``Wake Force Computation in Time and Frequency Domain", IEEE Trans. Magn., {\bf MAG-16:6}, (1980) 1300.
\bibitem{52} T. Weiland, {\it Transverse Beam Cavity Interaction. P.I: Short Range Forces}, Report DESY 82-015.
\bibitem{53} T. Weiland, {\it Transverse Beam Cavity Interaction. P.II: Long Range Forces}, Report DESY 83-005.
\bibitem{54} T. Weiland, {\it RF Cavity Design and Codes}, Report DESY M-86-07.
\bibitem{55} R. Klatt, F. Krawczyk, W.-R. Novender, C. Palm, et al. ``MAFIA -- A Three-Dimensional Electromagnetic CAD System for Magnets, RF Structures, and Transient Wake-Field Calcula\-tions", Rep. Linac Accel. Conf., Stanford, USA, June 2-6, 1986.
\bibitem{56} F. Ebeling, R. Klatt, F. Krawczyk, et al. ``The 3-D MAFIA Group of Electromagnetic Codes", IEEE Trans. Magn., {\bf 25:4} (1989) 2962.
\bibitem{57} M.J. Browman, R. Cooper, and T. Weiland, ``Three-Dimensional Cavity Calculations", Rep. Linac Accel. Conf., Stanford, USA, June 2-6, 1986.
\bibitem{58} R. Klatt and T. Weiland, ``Wake Filed Calculations with Three-Dimensional BCI Code", Rep. Linac Accel. Conf., Stanford, USA, June 2-6, 1986.
\bibitem{59} J.H. Whealton, G.L. Chen, R.J. Raridon, et al. ``A 3D Analysis of Maxwell's Equations for Cavities of Arbitrary Shape", J. of Comput. Phys., {\bf 75} (1988) 168.
\bibitem{60} B. Goplen, L. Ludeking, J. McDonald, G. Warren, and R. Worl, {\it MAGIC user's manual}, Mission Res. Corp. Nevington VA. Rep. MRC/WDC-R-184, Sept. 1988.
\bibitem{61} P.V. Koteteshvili, P.V. Rybak, and V.P. Tarakanov, {\it KARAT --- Code for Calculational Experiment in Electrodynamics}, Preprint IOPh AH SSSR No.44, 1991.
\bibitem{62} R. Engquist and A. ``Majda Absorbihg boundary conditions for the numerical simulation of waves", Math. Comp., {\bf 31} (1977) 629.
\bibitem{63} G. Mur, ``Absorbing Boundary Conditions for the Finite-Difference Approximation of the Time-Domain Electromagnetic-Field Equations", IEEE Trans. Electromagn. Compat., {\bf 23:4} (1981) 377.

\bibitem{64} A. Ben-Menahem, ``Green's tensors and associated potentials for electromag\-netic waves in inhomogeneous material media", Proc. Roy. Soc. Lond., {\bf A 426} (1989) 79.
\bibitem{65} J.P. Boris, {\it The acceleration Calculations from a Scalar Potential}, Report MATT-152, Plasma Physics Laboratory, Princeton University, 1970.
\bibitem{66} S.J. Beard and R.W. Hockney, ``POT4A-A program for the direct solution of Poisson's equation in complex geometries", Comp. Phys. Comm., {\bf 36}, (1985) 25.
\bibitem{67} E.E. Kunhardt and P.F. Williams, ``Direct solution of Poisson's equation in cylindrically symmetric geometry: a fast algorithm", J. Comp. Phys., {\bf 57} (1985) 403.
\end{thebibliography}
\end{document}

Library of Classes
for Object-Oriented Particles Simulation Project

written by Dr. Igor Turchanovsky and Dr. Andrey Petkun
in conjunction with Principle Investigator
Dr. Victor Ryzhov
High Current Electronics Institute, Tomsk, Russia

for the OOPIC project funded by
AIR Force Office of Scientific Research

Contract SPC93 - 4035(F6170893W0612)

In the previous report we presented a general scheme of relations architecture of the base classes, which, upper levels, is capable of including various particles models. While developing the OO-model architecture we made account of the necessity to create methods of the object classes interaction not only inside the functional model but also that between the computation and GUI-classes. The present report views classes allowing us to create a model with a GUI interface operating under MS WINDOWS.

The classes library which reflects the structure of the complete PIC-code architecture is made up of base classes, visualization classes interface and contemplation classes. Note should be made that the library contains different implementations of the same concepts (classes): vftrix, matr, array, vector, etc. This has been done in order to provide for the search for the most optimal program implementation both during the creation stage of certain classes and while developing a method to approach them.

BASE CLASSES are considered here as class hierarchies describing the numerical model of a specific task. To new base data types should also belong data classes (structures) describing common base concepts: coordinate, axis, vector, etc.

VISUALIZATION CLASSES are required for graphical specification of the computational region, checking of the correctness of mesh generation and specification of the boundary conditions of the task, as well as graphical representation of the results.

INTERFACE CLASSES serve to create an advanced interactive data input under MS WINDOWS. Application of this approach helps to create a user-friendly interface. The entirely class hierarchy is based on a standard class TDialog OWL,Borland Library.

COMPUTATIONAL CLASSES are developed to implement various numerical algorythms useful for solving electrostatic tasks. Thus, the emphasis in this report is made on developing straightforward method of solving Poisson equation.

1. BASE CLASSES

Development of new data types (structures and classes) requires a detailed description of the inner structure of the abstract class as well as the interface of interaction of the objects of a given class with other data types in a specific implementation of numerical algorithms. The process of writing the base classes is , however, iterational. This is due to the fact that in object- oriented programming it is difficult to readily determine the final structure of the new type data and still more difficult it is to describe all possible methods of the interaction of objects of a given class with the other program data. Classification of the library classes used here allows us to separate major (base) data (structure) with further detailed development of the structure and methods. These classes form the "lower" level of the library.

stuct:

AXISTRUCT - describes the type of the coordinate axis for physical values
pixel - determines the 2D coordinates on the display
vect - determines the physical value of the 3D coordinate
index - determines the 2D index
para - determines the 2D coordinates (TEMPLATE data type)
point - determines the 2D coordinates
TPhPoint - determines the physical value of the 2D coordinate
TCell - determines the 2D coordinates on the mesh
array - determines the array physical values
TVector - determines the array physical values

classes :

axis
TBoundary
TElectrod
TAbstrRegion
TGeom
TGeomFrameWork
TAbstrMesh
TAbstrModel
TabstrPhysVal
TEulerVal
TLagrangeVal
THistoricVal
TGridVal
TFunctionVal
TDoubleGridVal
TBeamVal

As a result of detailed elaboration of specific conditions of the numerical model each of the classes enumerated above gives rise to new type data inheriting both the data structure and the interactive behavior of the parent classes. These classes fill the upper level of the base classes describing a particular

ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ axis ÃÄÄÄ¿
ÀÄÄÄÄÂÄÄÄÄÄÄÄÄÄÄÙ ³

```
³ TGeomAxis ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄ´
³ TPhysAxis ³
ÃÄÄÄÄÄÄÄÄÄÄÄÄ´
³ TTimeAxis ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

This hierarchy is meant for the introduction into the numerical model of objects subsequently used as components of physical values. Their usage will simplify the display of physical values.

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄ¿
³TGeomFrameWorkÃÄÄÄ¿
ÀÄÄÄÂÄÄÄÄÄÄÄÄÙ ³
  ³ TGeomXY  ³
  ÃÄÄÄÄÄÄÄÄÄÄÄÄ´
  ³ TGeomRZ  ³
  ÃÄÄÄÄÄÄÄÄÄÄÄÄ´
  ³ TGeomRTh  ³
  ÃÄÄÄÄÄÄÄÄÄÄÄÄ´
  ³ TGeomX1X2  ³
  ÀÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

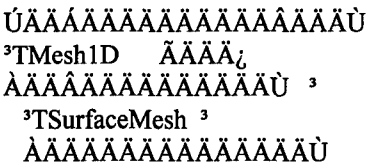
The name of this group implies that it deals with determination of the geometry type of a numerical model. These classes should be added by TElectrod and TGeom for specification of a random computational region.

The following class hierarchy is used to describe the mesh type of the computational region.

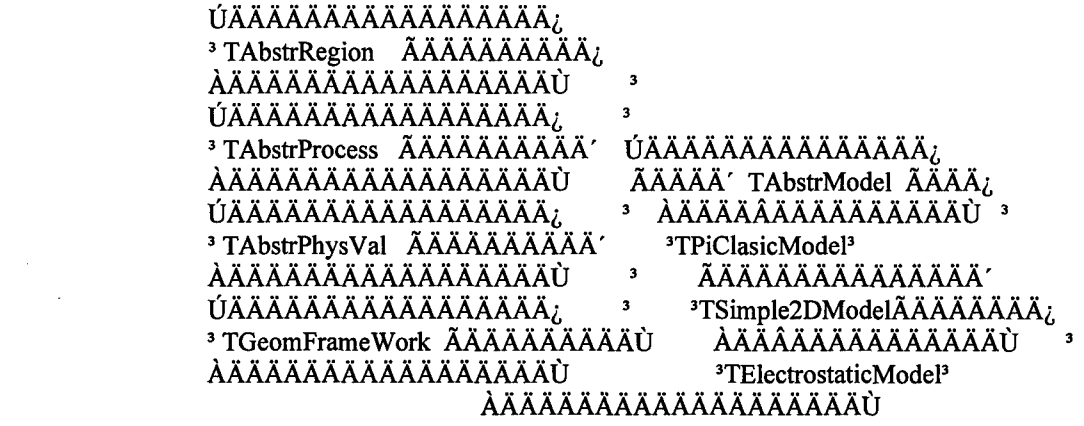
```
ÚÄÄÄÄÄÄÄÄÄÄÄÄ¿
³TAbstrRegion ÃÄÄÄ¿
ÀÄÄÄÂÄÄÄÄÄÄÄÄÙ ³
  ³TSampleRgn  ³
  ÃÄÄÄÄÄÄÄÄÄÄÄÄ´
  ³TRectangleRgnÃÄÄ¿
  ÀÄÄÄÂÄÄÄÄÄÄÄÄÙ ³
    ³TRectRgnRZ  ³
    ÀÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Classes hierarchy of the mesh of the computational region is needed to describe the type of the computational region, the knods template, etc. The abstract class TAbstrMesh incorporates two other abstract classes.

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ TAbstrRegions  ÃÄÄÄÄÄÄÄÄ¿  ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ      ÃÄÄÄÄ´ TAbstrMesh  ÃÄÄÄ¿
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿     ³  ÀÄÄÄÄÄÂÄÄÄÄÄÄÄÄÙ ³
³ TGeomFrameWork  ÃÄÄÄÄÄÄÄÄÙ       ³TMesh2D  ÃÄÄÄ¿
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ                 ÀÄÄÄÂÄÄÄÄÄÄÄÄÙ ³
                          ³TRegularMesh ³
                          ÃÄÄÄÄÄÄÄÄÄÄÄÄ´
                          ³TCellsListMesh³
                          ÃÄÄÄÄÄÄÄÄÄÄÄÄ´
                          ³TNonRectMesh ³
```



Classes describing the numerical model make use of the data of the computational region, boundary conditions, dimensions and symmetry, as well as the type of the task.




```
/*********************************************************************/
/*                                                                   */
/*              ObjectOriented PIC for C++                           */
/*                   Version 1.00                                    */
/*    Copyright (c) 1993, 1994 The Group Compuetr Simulation         */
/*          High Current Electronics Institute                       */
/*          Contract SPC93-4035(F6170893W0612)                       */
/*                                                                   */
/*                        Last write 10-20-94                        */
/*********************************************************************/
/*** PS_ARRAY.H ***/
#if !defined (_PS_ARRAY_H)
#define     _PS_ARRAY_H

#include "ps_defs.h"

void msg (int a, int t, int i, int n);

static int gc  = 0;
static int gca = 0;
static int mgc = 0;

/*=============================================================== array =*/
class array
{
 protected:
        num* e;
```

```
        int  step;
        int  N;
        int  i0;
        int  own;
        int typ;
   static int cont;
        int id;
        int chng;
        int tmp;
public:
 array ()      : i0 (0), N (0), typ(0), chng (0),tmp (0), own (0), step (1)
              { id=++cont; gca++; gc++; msg (1,typ,id,own); };
 array (int Size, int Tmp=0)   : i0 (0), typ(1), chng (0),tmp (Tmp), step (1)
              { e = new num [N=Size+1]; gca++; gc++;   own = N; id = ++cont;  msg (1,typ,id,own); };
 array (int Org, int End, int Tmp) : i0 (Org), typ(2), chng (0),tmp (Tmp), step (1)
              { e = new num [N=End-Org+1]; gca++; gc++; own = N; id= ++cont;  msg (1,typ,id,own); };
 array (const array& v, int Tmp=0);
 array (num* d, int Size, int Step=1, int Tmp=0) : i0 (0), typ(4), chng (0), tmp (Tmp), step (Step)
              { e = d; N = Size; own = 0; gca++; gc++; id= ++cont;  msg (1,typ,id,own); };
~array ()              { gca--; gc--; N = 0; if (own) delete [] e; e = 0;  msg (-1,typ,id,own); };
int  alloc  (int Size)     { if (own) return 0; e = new num [N=Size+1]; return own = N; };
void assign (num* p)       { e = p; };
     num&    operator [] (int i)   { return *(e+(i-i0)*step); };
     array&  operator  = (const array& v);
     array&  operator  = (num a);
     array&  operator += (const array& v);
     array&  operator *= (num a);
friend array   operator  + (array& v1, array& v2);
friend array   operator  * (num a,    array& v);
 void   add (array& v1, array& v2, num a = 1.0);
 void   add (array& v1, array& v2, array& v3, num a = 1.0);
 int  size ()               { return N-1; };
};
/*=============================================================== matrix =*/
class matrix
{
 public:
      num * data;
 public:
      int   Ni;
      int   Nj;
     indx   NN;
      int typ;
 static int cont;
      int id;
      int own;
      int chng;
      int tmp;
 matrix ()      : Ni(0), Nj(0), typ(0), chng(0), tmp(0), own(0)
             { mgc++;  gc++; id= ++cont;  msg (2,typ,id,own); };
 matrix (int N, int M, int Tmp = 0);
 matrix (int N);
 matrix (const matrix& a, int Tmp = 0);
 matrix (num * d, int N1, int N2, int Tmp = 0);
```

```
 ~matrix ();
      array & operator [] (int i);
      array & operator () (int j);
       num &  operator () (int i, int j)   { return *(data + i*Nj + j); };
      matrix &  operator =  (const matrix& a);
      matrix &  operator =  (num a);
      matrix &  operator += (const matrix& a);
      matrix &  operator *= (num a);
 friend matrix    operator +  (matrix& a1, matrix& a2);
 friend matrix    operator *  (num x,     matrix& a);
       void   assign (num* p)       { data = (typ==4)? p : 0; };  // !
       num &  elem (int i, int j)    { return *(data + i*Nj + j); };
       array  row  (int i)          { return array ((data+i*Nj), Nj); };
       array  col  (int j);
       void assign (int i, int j, num x)  { *(data + i*Nj + j) = x; };
};
/*********************************************************************  end */
#endif

#include <mem.h>

void msg (int a, int t, int i, int n)
{
}
/*_____________________________________________________  array __*/
int array :: cont = 0;

array :: array (const array& v, int Tmp)
{
   N = v.N;   i0 = v.i0;              gc++;  gca++;
                          e = new num [N];
 typ = 3;  chng = 0;  own = N;  tmp = Tmp;  id=++cont;  step = v.step;
  for (int i=0; i<N; i++) e[i]= v.e[i];
                          msg (1,typ,id,own);
}

array& array :: operator  = (const array& v)
{
  if ((step == 1) && (v.step == 1))
 memcpy ((void*)e, (void*)v.e,  N*sizeof(num));
      else
 for (int i=0; i<N; i++) *(e + i*step)  = *(v.e + i*v.step);
                          return (*this);
}

array& array :: operator  = (num a)
{
 for (int i=0; i<N; i++) *(e + i*step) = a;
                          return (*this);
}

array& array :: operator += (const array& v)
{
 for (int i=0; i<N; i++) *(e + i*step) += *(v.e + i*v.step);
```

```
                                return (*this);
}

array& array :: operator *= (num a)
{
 for (int i=0; i<N; i++) *(e + i*step) *= a;
                                return (*this);
}

void  array :: add (array& v1, array& v2, num a)
{
  for (int i=0; i<N; i++)
      *(e + i*step) = *(v1.e + i*v1.step) + a*(*(v2.e + i*v2.step));
}

void  array :: add (array& v1, array& v2, array& v3, num a)
{
  for (int i=0; i<N; i++)
  *(e + i*step) = *(v1.e + i*v1.step) + *(v2.e + i*v2.step) + a*(*(v3.e + i*v3.step));
};

//..... friends .....

array operator + (array& v1, array& v2)
{
array res (v1.N-1);
  for (int i=0; i < res.N; i++)
    *(res.e + i) = *(v1.e + i*v1.step) + *(v2.e + i*v2.step);
                                return (res);
};

array operator * (num a, array& v)
{
 array res (v.N-1);
  for (int i=0; i<res.N; i++)
    *(res.e + i) = a*(*(v.e + i*v.step));
                                return (res);
};
//---------------------------------------- num * data;
int matrix :: cont = 0;

matrix :: matrix (int N, int M, int Tmp)
{                                gc++; mgc++;
 Ni = N+1;  Nj = M+1;  NN = Nj*Ni;   data = new num [NN];
   typ = 1;  chng = 0;  own = NN;  tmp = Tmp; id=++cont;
 memset ((void*)data, 0, NN*sizeof(num));
                                msg (2,typ,id,own);
}

matrix :: matrix (int N)
{                                gc++;  mgc++;
 Ni = N+1;  Nj = N+1;  NN = Nj*Ni;   data = new num [NN];
   typ = 2;  chng = 0;  own = NN;   tmp = 0;  id=++cont;
 memset ((void*)data, 0, NN*sizeof(num));
```

```
                        msg (2,typ,id,own);
}

matrix :: matrix (const matrix& a, int Tmp)
{                                gc++; mgc++;
 Ni = a.Ni; Nj = a.Nj; NN = Nj*Ni;  data = new num [NN];
  typ = 3; chng = 0; own = NN;  tmp = Tmp; id=++cont;
 memcpy ((void*)data, (void*)a.data, NN*sizeof(num));
                        msg (2,typ,id,own);
}

matrix :: matrix (num * d, int N1, int N2, int Tmp)
{
 Ni = N1; Nj = N2; NN = Nj*Ni;  data = d;  gc++;  mgc++;
  typ = 4; chng = 0; own = 0;  tmp = Tmp; id=++cont;
                        msg (2,typ,id,own);
}

matrix :: ~matrix ()
{                    gc--; mgc--;
 if (own) delete [] data;
   data = 0;    Ni = 0; Nj = 0;
                        msg (-2,typ,id,own);
}

matrix& matrix :: operator  = (const matrix& a)
{
 memcpy ((void*)data, (void*)a.data, NN*sizeof(num));
                        return (*this);
}

matrix& matrix :: operator  = (num a)
{
 for (indx i=0; i<NN; i++) *(data+i) = a;
                        return (*this);
}

array & matrix :: operator [] ( int i)
{
       return (*new array ((data+i*Nj), Nj, 1, 1));
}

array & matrix :: operator () ( int j )
{
       return (*new array ((data+j), Ni, Nj, 1));
}

matrix& matrix :: operator += (const matrix& a)
{
 for (indx i=0; i < NN; i++) *(data+i) += *(a.data+i);
                        return (*this);
}

matrix& matrix :: operator *= (num a)
```

```
{
 for (indx i=0; i<NN; i++) *(data+i) *= a;
                              return (*this);
}

array  matrix :: col (int j)
{
   array res (Ni);
 for (int i=0; i<Ni; i++) res[i] = *(data + i*Nj + j);
                              return res;
}

//--------- friends ---------------------------

matrix operator + (matrix& a1, matrix& a2)
{
 matrix res (a1.Ni-1, a1.Nj-1);
     for (indx i=0; i < a1.NN; i++)
 res.data[i] = *(a1.data+i) + a2.data[i];
                              return (res);
}

matrix operator * (num x, matrix& a)
{
 matrix res (a.Ni-1, a.Nj-1);
     for (indx i=0; i < a.NN; i++)
  res.data[i] = x*(a.data[i]);
                          return (res);
};
//------------------------------------------------------------- end.

/*** PS_AXIS.H ***/
#if !defined (_PS_AXIS_H)
#define      _PS_AXIS_H

#include "ps_defs.h"

//---------------------------------------------------------------
#define NBoundaryTypes  7

enum TBoundaryType
  {
   ConductingBnd = 0,
   OpenBnd,
   AxisymmetricBnd,
   PeriodicBnd,
   DielectricBnd,
   MirrorsymmetricBnd,
   InternalBnd,
   InterRegionsBnd
  };
//---------------------------------------------------------------
#define NAxeTypes  8
```

```
enum TAxeType
 {
  XAxe = 0,
  YAxe,
  RAxe,
  ZAxe,
  ThAxe,
  x1Axe,
  x2Axe,
  FiAxe,
  tAxe
 };
//---------------------------------------------------------------
#define NUnitsTypes  10

enum TUnitsType
 {
  arbitrUnits = 0,
  mUnits,
  cmUnits,
  mmUnits,
  inchUnits,
  radUnits,
  dgrUnits,
  secUnits,
  nsUnits,
  msUnits,
  mcsUnits
 };
/*=============================================================  axis =*/
class axis
{
public:
   char  name  [LNGH_NAME];
   char  units [LNGH_UNITS];
   int   id;
   num   org, end;
   num   step, scale;
   int   N;
   int   uid;
   int   OwnDraw;
   int   OwnEdit;

 axis ( ):  id (0), uid (0), N (1)
       { org=0.0; end=1.0; scale=1.0; step=1.0; OwnDraw=OwnEdit=0; };
 axis ( char*, num, num=0.0 );
 axis ( const int at, const num x1, const num x2,
      const int n, const int unts = 0 );
 axis ( const int at, const num x1, const num x2,
          int n,  char * untstr );
 axis ( const axis& a );
~axis ( )      {};

      void Normalyze ( );
```

```
      void ScaleTo   ( const axis& x );
      void Transform ( const axis& x, num(*F)(num) );
virtual void Draw ( num Angle )          {};
virtual void Edit ( )                    {};

 num operator () (num i) { return org+i*step; };
protected:
    void axname  ( int nd );
    void axunits ( int ud );
};
/*=========================================================  TGeomAxis  =*/
class TGeomAxis : public axis
{
public:
   TBoundaryType bound1, bound2;
public:
     TGeomAxis () : axis ()      { };
     TGeomAxis ( const TGeomAxis& A );
     TGeomAxis ( int at, num x1, num x2,
             int bnd1, int bnd2, int n, int unts = 0 );
};
/*=========================================================  TPhysAxis  =*/
class TPhysAxis : public axis
{
public:
     TPhysAxis () : axis () {};
};
/*=========================================================  TTimeAxis  =*/
class TTimeAxis : public axis
{
public:
     TTimeAxis () : axis () {};
};
/*************************************************************************/
#endif

/*** PS_AXIS.CPP ***/
#include <string.h>
#include "ps_axis.h"

/*_____________________________________________________  axis __*/

axis :: axis (char* Nam, num En, num Or)
{
 id = 0; org = Or; end = En;  uid = 0;  N = 1;
                          axunits ( uid );
    strcpy (name, Nam);
  step =  (end > org)? ((end-org)/N) : -1.0;
 scale = 1.0 / step;
  OwnDraw = OwnEdit = 0;
};

axis :: axis (const axis& a)
{
```

```
 id = a.id;  org = a.org;    end = a.end;    uid = a.uid;
  N = a.N;  step = a.step; scale = a.scale;
     strcpy (name,  a.name);
     strcpy (units, a.units);
 OwnDraw = OwnEdit = 0;
};

axis :: axis ( const int at, const num x1, const num x2,
               const int n, const int unts )
{
 id = at; org = x1; end = x2;  N = n;  uid = unts;
                           axname ( at );  axunits ( uid );
    step =  (end > org)? ((end-org)/N) : -1.0;
    scale = 1.0 / step;
 OwnDraw = OwnEdit = 0;
};


axis :: axis ( const int at, const num x1, const num x2,
            int n,  char* untstr )
{
 id = at; org = x1; end = x2;  N = n;
                           axname ( at );
    strcpy (units, untstr);                uid = -1;
          if (stricmp (units, "arbitr"))   uid = 0;
          if (stricmp (units, "m"))        uid = 2;
          if (stricmp (units, "cm"))       uid = 3;
          if (stricmp (units, "mm"))       uid = 4;
          if (stricmp (units, "inchs"))    uid = 5;
          if (stricmp (units, "degrees"))  uid = 6;
          if (stricmp (units, "sec"))      uid = 7;
          if (stricmp (units, "ns"))       uid = 8;
          if (stricmp (units, "ms"))       uid = 9;
          if (stricmp (units, "mcs"))      uid = 10;
 step =  (end > org)? ((end-org)/N) : -1.0;
scale = 1.0 / step;
 OwnDraw = OwnEdit = 0;
};

void axis :: axname ( int nd )
{
        switch ( nd )
  { case 0:  strcpy (name, "X");   break;
    case 1:  strcpy (name, "Y");   break;
    case 2:  strcpy (name, "R");   break;
    case 3:  strcpy (name, "Z");   break;
    case 4:  strcpy (name, "Th");  break;
    case 5:  strcpy (name, "X1");  break;
    case 6:  strcpy (name, "X2");  break;
    case 7:  strcpy (name, "Fi");  break;
    case 8:  strcpy (name, "t");   break;
   default:      break;
  }
};
```

```
void axis :: axunits ( int ud )
{
        switch ( ud )
  { case 0:  strcpy (units, "arbitr");     break;
    case 1:  strcpy (units, "m");          break;
    case 2:  strcpy (units, "cm");         break;
    case 3:  strcpy (units, "mm");         break;
    case 4:  strcpy (units, "inchs");      break;
    case 5:  strcpy (units, "rad");        break;
    case 6:  strcpy (units, "degrees");    break;
    case 7:  strcpy (units, "sec");        break;
    case 8:  strcpy (units, "ns");         break;
    case 9:  strcpy (units, "ms");         break;
    case 10:  strcpy (units, "mcs");       break;
    default:                       break;
  }
};

void axis :: Normalyze ()
{
};

void axis :: ScaleTo (const axis& x)
{
};

void axis :: Transform (const axis& x, num(*F)(num))
{
};
//-------------------------------------------------------------
 TGeomAxis :: TGeomAxis ( const TGeomAxis & A )
        : axis ( A.id, A.org, A.end, A.N, A.uid )
{
  bound1 = A.bound1;
  bound2 = A.bound2;
};
 TGeomAxis :: TGeomAxis ( int at, num x1, num x2,
                    int bnd1, int bnd2, int n, int unts )
          : axis ( at, x1, x2, n, unts )
{
  bound1 = TBoundaryType ( bnd1 );
  bound2 = TBoundaryType ( bnd2 );
};
//------------------------------------------------------------ end.

/*** PS_GEOM.H ***/
#if !defined (_PS_GEOM_H)
#define      _PS_GEOM_H

#include "ps_defs.h"

template <class tip> struct para
{
```

```
    tip x1;
    tip x2;
  para ()            { x1 =  0;   x2 =  0; };
  para (para& p)       { x1 = p.x1;  x2 = p.x2; };
  para (tip a1, tip a2)  { x1 = a1;   x2 = a2; };
    para  operator =  (para a)  { x1 = a.x1;  x2 = a.x2; return (*this); };
    para  operator () (tip a1, tip a2)  { x1 = a1;  x2 = a2; return (*this); };
friend int  operator == (para p1, para p2)
         { return ((p1.x1==p2.x1)&&(p1.x2==p2.x2)) ? 1 : 0; };
};

struct point
{
    num x1;
    num x2;
  point ()            { x1 = 0.0;   x2 = 0.0;  };
  point (point& p)      { x1 = p.x1;   x2 = p.x2; };
  point (num a1, num a2)  { x1 = a1;    x2 = a2;   };
    point operator =  (point a)  { x1 = a.x1;  x2 = a.x2; return (*this); };
    point operator () (num a1, num a2)  { x1 = a1;  x2 = a2; return (*this); };
friend int  operator == (point p1, point p2)
         { return ((p1.x1==p2.x1)&&(p1.x2==p2.x2)) ? 1 : 0; };
};

struct pixel { int h;  int v; };
struct vect  { num x1; num x2; num x3; };
struct index { int i1; int i2; };

struct AXISTRUCT
{
  char   axtype;
  char * axname;
  char * units;
  char * lobound;
  char * upbound;
  char * bndtype;
};

/**** TGeomFrameWork ----------------------------------*/
class TGeomFrameWork
{
public:
    static int geom_assigned;
        int geom;
protected:
       num h1,   h2;       // Space steps for x1, x2
       num asph, asph2;    // h1/h2 & its square
       int N1,   N2;       // Dimensions
       int equih;          // =0, if h1,h2 - const(x1,x2)
public:
     TGeomFrameWork ( );
  virtual void x1 (AXISTRUCT& axe) = 0;
  virtual void x2 (AXISTRUCT& axe) = 0;
  virtual num  ds (point x, point h) = 0;
```

```
        void SetSteps (num dx1, num dx2, int eqx = 0)
                        { h1 = dx1; h2 = dx2; equih = eqx;
                          asph = h1/h2;  asph2 = asph*asph; };
  virtual void LaplaceShablon (point x, point h,
                num& x1m, num& x1, num& x1p,
                num& x2m, num& x2, num& x2p, num& gf) = 0;
};
/**** TGeomXY ----------------------------------------*/
class TGeomXY : public TGeomFrameWork
{
public:
    TGeomXY ( );
  virtual void x1 (AXISTRUCT& axe);
  virtual void x2 (AXISTRUCT& axe);
  virtual num  ds (point x, point h);
  virtual void LaplaceShablon (point x, point h,
                num& x1m, num& x1, num& x1p,
                num& x2m, num& x2, num& x2p, num& gf);
};
/**** TGeomRZ ----------------------------------------*/
class TGeomRZ : public TGeomFrameWork
{
public:
    TGeomRZ ( );
  virtual void x1 (AXISTRUCT& axe);
  virtual void x2 (AXISTRUCT& axe);
  virtual num  ds (point x, point h);
  virtual void LaplaceShablon (point x, point h,
                num& x1m, num& x1, num& x1p,
                num& x2m, num& x2, num& x2p, num& gf);
};
/**** TGeomRTh ----------------------------------------*/
class TGeomRTh : public TGeomFrameWork
{
public:
    TGeomRTh ( );
  virtual void x1 (AXISTRUCT& axe);
  virtual void x2 (AXISTRUCT& axe);
  virtual num  ds (point x, point h);
  virtual void LaplaceShablon (point x, point h,
                num& x1m, num& x1, num& x1p,
                num& x2m, num& x2, num& x2p, num& gf);
};
/**** TGeomX1X2 ----------------------------------------*/
class TGeomX1X2 : public TGeomFrameWork
{
public:
    TGeomX1X2 ( );
  virtual void x1 (AXISTRUCT& axe);
  virtual void x2 (AXISTRUCT& axe);
  virtual num  ds (point x, point h);
  virtual void LaplaceShablon (point x, point h,
                num& x1m, num& x1, num& x1p,
                num& x2m, num& x2, num& x2p, num& gf);
```

```
};
//-----------------------------------------------------*
//  TGeomFrameWork* GFW;
//-----------------------------------------------------*
#endif
/*** PS_GEOM.CPP ***/
#include "ps_geom.h"

/**** TGeomFrameWork ----------------------------------*/
  TGeomFrameWork::TGeomFrameWork ()
{
   h1 = 1.0;  h2 = 1.0;  equih = 0; asph = 1.0; asph2 = 1.0; geom = -1;
};

 int TGeomFrameWork::geom_assigned = -1;

/**** TGeomXY -----------------------------------------*/
    TGeomXY::TGeomXY () : TGeomFrameWork ()
{
     geom = 1;
};

 void TGeomXY :: x1 (AXISTRUCT& axe)
{
    axe.axtype  = 'x';
    axe.axname  = "Length";
    axe.units   = "cm";
    axe.lobound = "Left";
    axe.upbound = "Right";
    axe.bndtype = "Open";
};

 void TGeomXY :: x2 (AXISTRUCT& axe)
{
    axe.axtype  = 'x';
    axe.axname  = "Length";
    axe.units   = "cm";
    axe.lobound = "Left";
    axe.upbound = "Right";
    axe.bndtype = "Open";
};

 num TGeomXY :: ds (point x, point h)
{
      return (h.x1 * h.x2);
};

 void TGeomXY :: LaplaceShablon (point x, point h,
                      num& x1m, num& x1, num& x1p,
                      num& x2m, num& x2, num& x2p,
                                  num& gfr )
{
 x1m = 1.0;   x1 = 2.0;      x1p = 1.0;
 x2m = asph2;  x2 = 2.0*asph2;  x2p = asph2;
```

```
    gfr = h1*h1;
};

/**** TGeomRZ ------------------------------------------*/
    TGeomRZ::TGeomRZ () : TGeomFrameWork ()
{
     geom = 2;
};

 void TGeomRZ :: x1 (AXISTRUCT& axe)
{
    axe.axtype  = 'x';
    axe.axname  = "Length";
    axe.units   = "cm";
    axe.lobound = "Left";
    axe.upbound = "Right";
    axe.bndtype = "Open";
};

 void TGeomRZ :: x2 (AXISTRUCT& axe)
{
    axe.axtype  = 'x';
    axe.axname  = "Length";
    axe.units   = "cm";
    axe.lobound = "Left";
    axe.upbound = "Right";
    axe.bndtype = "Open";
};

 num TGeomRZ :: ds (point x, point h)
{
      return (h.x1 * h.x2);
};

 void TGeomRZ :: LaplaceShablon (point x, point h,
                      num& x1m, num& x1, num& x1p,
                      num& x2m, num& x2, num& x2p,
                                  num& gfr )
{
         asph = h.x1 / h.x2;  asph2 = asph * asph;
 x1m = (x.x1-0.5*h.x1)*asph2/x.x1;
 x1p = (x.x1+0.5*h.x1)*asph2/x.x1;
 x1  = x1m + x1p + 2;
      gfr = h.x1*h.x1*eps0;
 x2m = 1.0;
 x2p = 1.0;
 x2  = x2m + x2p;
};

/**** TGeomRTh -----------------------------------------*/
    TGeomRTh::TGeomRTh () : TGeomFrameWork ()
{
     geom = 3;
};
```

```
 void TGeomRTh :: x1 (AXISTRUCT& axe)
{
    axe.axtype  = 'x';
    axe.axname  = "Length";
    axe.units   = "cm";
    axe.lobound = "Left";
    axe.upbound = "Right";
    axe.bndtype = "Open";
};

 void TGeomRTh :: x2 (AXISTRUCT& axe)
{
    axe.axtype  = 'x';
    axe.axname  = "Length";
    axe.units   = "cm";
    axe.lobound = "Left";
    axe.upbound = "Right";
    axe.bndtype = "Open";
};

 num TGeomRTh :: ds (point x, point h)
{
      return (x.x1 * h.x1 * h.x2);          // r*dr*dth
};

 void TGeomRTh :: LaplaceShablon (point x, point h,
                      num& x1m, num& x1, num& x1p,
                      num& x2m, num& x2, num& x2p,
                                  num& gfr )
{
 x1m = 1.0;   x1 = 2.0;      x1p = 1.0;
 x2m = asph2; x2 = 2.0*asph2; x2p = asph2;
   gfr = h1*h1;
};
/**** TGeomX1X2 ----------------------------------------*/
   TGeomX1X2::TGeomX1X2 () : TGeomFrameWork ()
{
     geom = 4;
};

 void TGeomX1X2 :: x1 (AXISTRUCT& axe)
{
    axe.axtype  = 'x';
    axe.axname  = "Length";
    axe.units   = "cm";
    axe.lobound = "Left";
    axe.upbound = "Right";
    axe.bndtype = "Open";
};

 void TGeomX1X2 :: x2 (AXISTRUCT& axe)
{
    axe.axtype  = 'x';
```

```
    axe.axname  = "Length";
    axe.units   = "cm";
    axe.lobound = "Left";
    axe.upbound = "Right";
    axe.bndtype = "Open";
};

 num TGeomX1X2 :: ds (point x, point h)
{
      return (x.x1 * h.x1 * h.x2);          // r*dr*dth
};

 void TGeomX1X2 :: LaplaceShablon (point x, point h,
                        num& x1m, num& x1, num& x1p,
                        num& x2m, num& x2, num& x2p,
                                        num& gfr )
{
 x1m = 1.0;   x1 = 2.0;       x1p = 1.0;
 x2m = asph2; x2 = 2.0*asph2; x2p = asph2;
   gfr = h1*h1;
};
/*** PS_MESH.H ***/
#if !defined (_PS_MESH_H)
#define      _PS_MESH_H

#include "ps_regn.h"

class TGridVal;
class TAbstrProcess;

typedef  void (*TApplicator) (int, int);
typedef  void (TAbstrProcess::*TPrcsApplicator) (int, int);

/*_____________________________________________________  TAbstrMesh _____*/

class TAbstrMesh
{
 public:
    TGeomAxis   *X1, *X2;
    TGeomFrameWork *GFW;
    TAbstrRegion   *Rgn;
 static int id_Mesh;
   int Status;
   int ic, jc;       // Current Node index
   num x1, x2;       // Boundaries for radius
   num y1, y2;       // Boundaries for  z
   num h1, h2;       // Space steps
   num asph, asph2;
   int Nx, Ny;       // Dimensions
   int N1, N2;       // Dimensions
   int Ms;
   int OwnDraw;
   int CanEdit;
public:
```

```
        TAbstrMesh ()      {};
        TAbstrMesh (int);
        TAbstrMesh (const TAbstrMesh& aMesh, int Shift);
  virtual ~TAbstrMesh ();

        void Prepare     ()      {};

  virtual void AllocateVal   ( TGridVal& A, long& MeshSize, int& incr ) = 0;
  virtual void DeAllocateVal ( TGridVal& A, long& MeshSize) = 0;

  virtual point operator () ( int i, int j ) = 0;

  virtual unsigned int Offset  ( int i, int j ) = 0;

  virtual void ForAllNodes      ( TApplicator ) = 0;
  virtual void ForAllNodes      ( TAbstrProcess&, TPrcsApplicator ) = 0;
  virtual void ForAllInnerNodes ( TApplicator ) = 0;
  virtual void ForAllInnerNodes ( TAbstrProcess&, TPrcsApplicator ) = 0;
  virtual void ForRow           ( num, TApplicator ) = 0;
  virtual void ForRow           ( TAbstrProcess&, num, TPrcsApplicator ) = 0;
  virtual void LeapFrogNodes    ( TApplicator ) = 0;
  virtual void LeapFrogNodes    ( TAbstrProcess&, TPrcsApplicator ) = 0;

  virtual void ResetIndex () { ic = jc = 0; };
  virtual int  NextIndex ( int& i, int& j ) { i = ic;  j = ++jc; return 0;};
  virtual int  PrevIndex ( int& i, int& j ) { i = ic;  j = --jc; return 0;};
  virtual int  TopIndex  ( int& i, int& j ) { i = ++ic; j = jc;  return 0;};
  virtual int  DownIndex ( int& i, int& j ) { i = --ic; j = jc;  return 0;};

  virtual void DrawMesh ( ) {};
  virtual void EditMesh ( ) {};
  virtual void SaveMesh ( ) {};
  virtual void OpenMesh ( ) {};
};
/*_____________________________________________________ TMesh2D _____________*/

class TMesh2D : public TAbstrMesh
{
public:
        TMesh2D (): TAbstrMesh ()    {};
        TMesh2D ( TGeomAxis* x1, TGeomAxis* x2 );
        TMesh2D ( TAbstrRegion* aRgn );

  virtual unsigned int Offset  ( int i, int j ) { return (i*(N2-N1)+j)/*sizeof(num)*/;};

  virtual num& Centre ( num * base ) { return *(base + Offset(ic,  jc  )); };
  virtual num& West   ( num * base ) { return *(base + Offset(ic-1,jc  )); };
  virtual num& East   ( num * base ) { return *(base + Offset(ic+1,jc  )); };
  virtual num& North  ( num * base ) { return *(base + Offset(ic,  jc+1)); };
  virtual num& South  ( num * base ) { return *(base + Offset(ic,  jc-1)); };

  virtual void Shablon ( int& left, int& right, int& top, int& bottom )
                { left=ic-1; right=ic+1; top=jc+1; bottom=jc-1; };
```

```
  virtual point operator () ( int i, int j ) { return point((*X1)(i),(*X2)(j));};
        num Coord1 (int i )  { return (*X1)(i); };
        num Coord2 (int i )  { return (*X2)(i); };
  virtual num Interpolate ( const TGridVal& F, point x );
  virtual int NearestRow  ( num x ) = 0;
  virtual int NearestCol  ( num x ) = 0;

  virtual int CreateMesh  ( )  { return 0; };
};
/*_____________________________________________________ TRegularMesh ____*/

class TRegularMesh : public TMesh2D
{
 public:
     TRegularMesh (): TMesh2D () {};
     TRegularMesh (TGeomAxis* x1, TGeomAxis*x2): TMesh2D (x1,x2) {};
//                override pure abstract methods
  virtual void AllocateVal   ( TGridVal& A, long& MeshSize, int& incr );
  virtual void DeAllocateVal ( TGridVal& A, long& MeshSize );
  virtual int  NearestRow  ( num y ) { return (int)((y - y1)/h2); };
  virtual int  NearestCol  ( num x ) { return (int)((x - x1)/h1); };

  virtual void ForAllNodes     ( TApplicator Scheme );
  virtual void ForAllNodes     ( TAbstrProcess& p, TPrcsApplicator Scheme );
  virtual void ForAllInnerNodes ( TApplicator Scheme );
  virtual void ForAllInnerNodes ( TAbstrProcess& p, TPrcsApplicator Scheme );
  virtual void ForRow          ( num Xrow, TApplicator Scheme );
  virtual void ForRow          ( TAbstrProcess& p, num Xrow, TPrcsApplicator Scheme );
  virtual void LeapFrogNodes   ( TApplicator Scheme );
  virtual void LeapFrogNodes   ( TAbstrProcess& p, TPrcsApplicator Scheme );
};
/*_____________________________________________________ TCellsListMesh __*/

class TCellsListMesh : public TMesh2D
{
 public:
     TCellsListMesh (): TMesh2D () {};
};
/*_____________________________________________________ TNonRectMesh   __*/

class TNonRectMesh : public TMesh2D
{
 public:
     TNonRectMesh (): TMesh2D () {};
};
/*_____________________________________________________ TComplexMesh   __*/

class TComplexMesh : public TNonRectMesh
{
 public:
     TComplexMesh (): TNonRectMesh () {};
};
/*_____________________________________________________ T1DMesh        __*/
```

```
class T1DMesh : public TAbstrMesh
{
 public:
    T1DMesh (): TAbstrMesh () {};
};
/*_____________________________________________________ T3DMesh      __*/

class T3DMesh : public TAbstrMesh
{
 public:
    T3DMesh (): TAbstrMesh () {};
};
/*_____________________________________________________ TPhaseSpaceMesh _*/

class TPhaseSpaceMesh : public TAbstrMesh
{
 public:
    TPhaseSpaceMesh (): TAbstrMesh () {};
};
/*_____________________________________________________ TSurfacesMesh  __*/

class TSurfaceMesh : public T1DMesh
{
 public:
    TSurfaceMesh (): T1DMesh () {};
};
/*_____________________________________________________ TInterlaceMesh __*/

class TInterlaceMesh : public TRegularMesh
{
 public:
    TInterlaceMesh (): TRegularMesh () {};
};
/*------------------------------------------------------------  end  --*/
#endif
/*** PS_MESH.CPP ***/
#include "ps_phval.h"
#include "ps_mesh.h"
#include "ps_procs.h"

/********** TAbstrMesh ----------------------------------*/
 TAbstrMesh::TAbstrMesh (int)
{
}
 TAbstrMesh::TAbstrMesh (const TAbstrMesh& aMesh, int Shift)
{
}
 TAbstrMesh::~TAbstrMesh () {}

//////////////////////////////////////////////////////////////
 TMesh2D::TMesh2D (TGeomAxis* x1, TGeomAxis* x2)
{
  X1 = x1;  N1 = Nx = (X1->N);  h1 = (X1->step);
  X2 = x2;  N2 = Ny = (X2->N);  h2 = (X2->step);
```

```
    asph = (h2 > 0.0)? (h1/h2) : (1.0);  asph2 = asph * asph;
 int k = N2; int m = 0;
          while (k > 1) { k /= 2; ++m; }
     Ms = ((1 << m) == N2)? (m) : 0;
}
//---------------------------------------------------------
 TMesh2D::TMesh2D (TAbstrRegion* aRgn)
{
          X1 = new TGeomAxis ( );
          X2 = new TGeomAxis ( );
}
num TMesh2D :: Interpolate (const TGridVal& F, point x)
{
return 0;
}
//////////////////////////////////////////////////////////////////////
//---------------------------------------------------------
 void TRegularMesh :: AllocateVal  ( TGridVal& A, long& MeshSize, int& incr )
{
 A.data = new num [MeshSize = (N1+1)*(N2+1)];  incr = N1+1;
}
 void TRegularMesh :: DeAllocateVal( TGridVal& A, long& )
{
 delete [] A.data;
}
//---------------------------------------------------------
 void TRegularMesh :: ForAllNodes ( TApplicator Scheme )
{
 for (int i=0; i<=N1; i++)
 for (int j=0; j<=N2; j++)
                  Scheme (i,j);
}
 void TRegularMesh :: ForAllNodes ( TAbstrProcess& p, TPrcsApplicator Scheme )
{
 for (int i=0; i<=N1; i++)
 for (int j=0; j<=N2; j++)
                  (p.*Scheme) (i,j);
}
 void TRegularMesh :: ForAllInnerNodes ( TApplicator Scheme )
{
 for (int i=1; i<N1; i++)
 for (int j=1; j<N2; j++)
                  Scheme (i,j);
}
 void TRegularMesh :: ForAllInnerNodes ( TAbstrProcess& p, TPrcsApplicator Scheme )
{
 for (int i=1; i<N1; i++)
 for (int j=1; j<N2; j++)
                 (p.*Scheme) (i,j);
}
 void TRegularMesh :: ForRow ( num Xrow, TApplicator Scheme )
{
 int i0 = NearestRow (Xrow);
 for (int j=1; j<N2; j++)
```

```
                Scheme (i0,j);
}
 void TRegularMesh :: ForRow ( TAbstrProcess& p, num Xrow, TPrcsApplicator Scheme )
{
 int i0 = NearestRow (Xrow);
 for (int j=1; j<N2; j++)
              (p.*Scheme) (i0,j);
}
 void TRegularMesh :: LeapFrogNodes ( TApplicator Scheme )
{
 for (int i=1;  i < N1;  i+=2)
 for (int j=1;  j < N2;  j+=2)
                Scheme (i,j);
}
 void TRegularMesh :: LeapFrogNodes ( TAbstrProcess& p, TPrcsApplicator Scheme )
{
 for (int i=1;  i < N1;  i+=2)
 for (int j=1;  j < N2;  j+=2)
              (p.*Scheme) (i,j);
}
/*** PS_PHVAL.H ***/
#if !defined (_PS_PHVAL_H)
#define     _PS_PHVAL_H

#include "ps_axis.h"
#include "ps_array.h"
#include "ps_geom.h"

class TAbstrMesh;

//   TAbstrPhysVal ----------------------------------
class TAbstrPhysVal
{
public:
    TAbstrPhysVal * Next;
    TPhysAxis    * Axis;
    num Org, End;
    num Min, Max;
    int  N;
    int OwnDraw;
public:
        TAbstrPhysVal ();
        TAbstrPhysVal ( const TPhysAxis& Ax );
        TAbstrPhysVal ( const TAbstrPhysVal& P );
 virtual ~TAbstrPhysVal ()                    { };

        void Create ()  {};
        void Destroy ()  {};
        void Update ()  {};

        void Redraw ()  {};
        void Save   ();
        void Restore ();
   virtual int PlotsStep () { return 1; };
```

```
    virtual int PlotsSize () { return N; };

   virtual TAbstrPhysVal  operator = ( const TAbstrPhysVal& A );
   virtual TAbstrPhysVal  operator + ( const TAbstrPhysVal& A );
   virtual TAbstrPhysVal  operator - ( const TAbstrPhysVal& A );
    friend TAbstrPhysVal  operator * ( num coef, const TAbstrPhysVal& A );
};

//   TEulerVal ----------------------------------
class TEulerVal : public TAbstrPhysVal
{
public:
  TAbstrMesh * Mesh;
public:
    TEulerVal () : TAbstrPhysVal () { };

 num operator () ( point x )      { return 0; };
 num operator () ( int i, int j )  { return 0; };
};

//   TLagrangeVal -------------------------------
class TLagrangeVal : public TAbstrPhysVal
{
public:
    TLagrangeVal () : TAbstrPhysVal () { };
};

//   THistoricVal -------------------------------
class THistoricVal : public TAbstrPhysVal
{
public:
    THistoricVal () : TAbstrPhysVal () { };
};

//   TGridVal -----------------------------------
class TGridVal : public TEulerVal
{
public:
       num * data;
public:
    TGridVal () : TEulerVal () { };
};

//   TFunctionVal -------------------------------
class TFunctionVal : public TEulerVal
{
public:
    TFunctionVal () : TEulerVal () { };
};


//   TDoubleGridVal -----------------------------
class TDoubleGridVal : public TGridVal
{
```

```
public:
    TDoubleGridVal () : TGridVal () { };
};
//   TBeamVal ------------------------------------
class TBeamVal : public TLagrangeVal
{
public:
    TBeamVal () : TLagrangeVal () { };
};
//-----------------------------------------------------
#endif
/*** PS_PHVAL.CPP ***/
#include "ps_phval.h"

/*------------------------------------------------------ TAbstrPhysVal --*/
 TAbstrPhysVal :: TAbstrPhysVal ()
{
};

 TAbstrPhysVal :: TAbstrPhysVal ( const TPhysAxis& Ax )
{
};

 TAbstrPhysVal :: TAbstrPhysVal ( const TAbstrPhysVal& P )
{
};

void TAbstrPhysVal :: Save ( )
{
};

void TAbstrPhysVal :: Restore ( )
{
};
TAbstrPhysVal TAbstrPhysVal::operator = ( const TAbstrPhysVal& A )
{
return *this;
};
TAbstrPhysVal TAbstrPhysVal::operator + ( const TAbstrPhysVal& A )
{
return *this;
};
TAbstrPhysVal TAbstrPhysVal::operator - ( const TAbstrPhysVal& A )
{
return *this;
};
TAbstrPhysVal /* friend */  operator * ( num coef, const TAbstrPhysVal& A )
{
TAbstrPhysVal tmp;
return tmp;
};
/*** PS_REGN.H ***/
#if !defined (_PS_REGN_H)
#define      _PS_REGN_H
```

```
#include "ps_geom.h"
#include "ps_axis.h"

/**** TBoundary ------------------------------------*/

class TBoundary
{
  static TBoundary * CurrentBnd;
public:
        point   Org;
        point   End;
    TBoundaryType  BndType;
        int   OwnDraw;
     TBoundary * PrevBnd;
     TBoundary * NextBnd;
public:
      TBoundary ()  { Org = point (0.0, 0.0);
                  End = point (1.0, 1.0); };
      TBoundary ( point x1, point x2,
              TBoundaryType bt = ConductingBnd );
      TBoundary ( point x1, point x2,
              TBoundaryType bt, TBoundary * nxt );
      TBoundary ( point x2,
              TBoundaryType bt, TBoundary * nxt );
      TBoundary ( TBoundary* frm, point x2,
              TBoundaryType bt = ConductingBnd );
  int ConnectWith ( TBoundary * b1, TBoundary * b2 );
};

/**** TAbstrRegion ---------------------------------*/

class TAbstrRegion
{
public:
       int Type;
       int OwnDraw;
       int CanEdit;
       int NBounds;
     TBoundary * BoundaryList;
   TAbstrRegion * Next;
public:
          TAbstrRegion () { Next = 0; Type = 0; };
          TAbstrRegion ( num, num, num, num );
  virtual int IsInner (point x) { return 0; };
  virtual TGeomAxis* X1 ( )    { return 0; };
  virtual TGeomAxis* X2 ( )    { return 0; };
};

/**** TRectangleRgn --------------------------------*/

class TRectangleRgn : public TAbstrRegion
{
protected:
```

```
        TGeomAxis * x1;
        TGeomAxis * x2;
        num x1min, x1max;   // Boundaries for x1
        num x2min, x2max;   // Boundaries for x2
private:
        num hx1,  hx2;     // Space steps
        num asph,  asph2;
        int Nx1,  Nx2;     // Dimensions
public:
      TRectangleRgn () : TAbstrRegion () { Type = 10; };
      TRectangleRgn ( num, num, num, num );
      TRectangleRgn ( TGeomAxis * ax1, TGeomAxis * ax2 );
  virtual int IsInner (point x) { return ( (x.x1 > x1min)&&
                                (x.x1 < x1max)&&
                                (x.x2 > x2min)&&
                                (x.x2 < x2max) ); };
        int AssignBounds ( TGeomAxis * ax1, TGeomAxis * ax2 );
   TGeomAxis* X1 ( )   { return x1; };
   TGeomAxis* X2 ( )   { return x2; };
};

/**** TRectRgnRZ ------------------------------------*/

class TRectRgnRZ : public TRectangleRgn
{
protected:
        num R1, R2;       // Boundaries for radius
        num Z1, Z2;       // Boundaries for z
        num hr, hz;       // Space steps
        int Nr, Nz;       // Dimensions
public:
            TRectRgnRZ () : TRectangleRgn () { Type = 12; };
            TRectRgnRZ ( num, num, num, num );
      num rad ( int j ) { return R1 + j*hr; };
      num rad ( num x ) { return R1 + x*hr; };
};

/**** TSimpleRegion ---------------------------------*/

class TSampleRgn  : public TAbstrRegion
{
public:
        TSampleRgn () : TAbstrRegion() { Type = 20; };
};
//------------------------------------------------- end
#endif
/*** PS_REGN.CPP ***/
#include "ps_regn.h"

/********* TBoundary ----------------------------------*/
 TBoundary* TBoundary::CurrentBnd = 0;

 TBoundary::TBoundary ( point x1, point x2, TBoundaryType bt )
{
```

```
 Org = x1;  End = x2; BndType = bt;
    PrevBnd = CurrentBnd;
    NextBnd = 0;
  CurrentBnd = this;
}

TBoundary::TBoundary ( point x1, point x2, TBoundaryType bt, TBoundary * nxt )
{
 Org = x1;  End = x2; BndType = bt;
    PrevBnd = CurrentBnd;
    NextBnd = nxt;
  CurrentBnd = this;
}

TBoundary::TBoundary ( point x2, TBoundaryType bt, TBoundary * nxt )
{
 Org = CurrentBnd->End;  End = x2; BndType = bt;
    PrevBnd = CurrentBnd;
    NextBnd = nxt;
  CurrentBnd = this;
}

TBoundary::TBoundary ( TBoundary *frm, point x2, TBoundaryType bt )
{
 Org = frm->End;  End = x2; BndType = bt;
    PrevBnd = frm;
    NextBnd = 0;
  CurrentBnd = this;
}

int TBoundary::ConnectWith ( TBoundary *bt1, TBoundary *bt2 )
{
    PrevBnd = bt1;
    NextBnd = bt2;
return ( (bt1->End == Org) && (bt2->Org == End) );
}

/************ TAbstrRegion ----------------------------------*/
TAbstrRegion::TAbstrRegion ( num, num, num, num )
{
 OwnDraw = 0;
}
/************ TRectangleRgn ---------------------------------*/
TRectangleRgn::TRectangleRgn ( num, num, num, num )
        :TAbstrRegion ()
{
}
TRectangleRgn::TRectangleRgn ( TGeomAxis * ax1, TGeomAxis *ax2 )
        :TAbstrRegion ()
{      OwnDraw = CanEdit = 0;
 x1 = new TGeomAxis ( *ax1 );
 x2 = new TGeomAxis ( *ax2 );
x1min = ax1->org;    x1max = ax1->end;
x2min = ax2->org;    x2max = ax2->end;
```

```
    NBounds = 4;
//  BoundaryList = new TBoundary
//     ( point (x1min, x2min), point (x1max, x2min),
//        TBoundaryType( ax2->bound1) );

//              new TBoundary
//     ( point (x1max, x2max), PeriodicBnd,
//              new TBoundary
//     ( point (x1min, x2max), ConductingBnd,
//              new TBoundary
//     ( point (x1min, x2min), PeriodicBnd,
//                              BoundaryList ))));
}
 int TRectangleRgn::AssignBounds ( TGeomAxis * ax1, TGeomAxis *ax2 )
{
 x1min = ax1->org;    x1max = ax1->end;
 x2min = ax2->org;    x2max = ax2->end;
    NBounds = 4;
  BoundaryList = new TBoundary
    ( point (x1min, x2min), point (x1max, x2min), AxisymmetricBnd,
            new TBoundary
    ( point (x1max, x2max), PeriodicBnd,
            new TBoundary
    ( point (x1min, x2max), ConductingBnd,
            new TBoundary
    ( point (x1min, x2min), PeriodicBnd,
                            BoundaryList ))));
 return 0;
}
/************  TRectRgnRZ ----------------------------------------*/
 TRectRgnRZ::TRectRgnRZ ( num, num, num, num )
        :TRectangleRgn ()
{
}
//---------------------------------------------------------- end.
```

2. VISUALIZATION CLASSES

This part of the library is base on the GUI-interface of the OWL Library compiled at Borland. The major classes developed to display in the child windows are based on the abstract class TBasePlotswin.

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³TBasePlotsWinÃÄÄÄ¿
ÀÄÄÄÂÄÄÄÄÄÄÄÄÄÙ ³
  ³TSurfacePlot ³
  ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ´
  ³TLinesPlot  ÃÄÄ¿
  ÀÄÄÄÂÄÄÄÄÄÄÄÄÄÙ ³
    ³TRegionPlot ³
    ÃÄÄÄÄÄÄÄÄÄÄÄÄÄ´
    ³TPointsPlot ³
```

ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ

```
/*********************************************************************/
/*                                                                   */
/*                ObjectOriented PIC for C++                         */
/*                    Version 1.00                                   */
/*    Copyright (c) 1993, 1994 The Group Compuetr Simulation         */
/*           High Current Electronics Institute                      */
/*           Contract SPC93-4035(F6170893W0612)                      */
/*                                                                   */
/*                          Last write 10-20-94                      */
/*********************************************************************/
/* PS_GRAPH.H   */
//                      Graphics window object header
#ifndef _PS_GRAPH_H
#define _PS_GRAPH_H

class _OWLCLASS TFrameWindow;
class _OWLCLASS TStatic;
class _OWLCLASS TPen;

#include "ps_plots.h"
#include "ps_phval.h"
#include "ps_regn.h"
#include "ps_idres.rh"

const int WM_PW_CLOSED = WM_USER + 100;

/*------- TBaseGraphWin ---------------------------------------------- */
//                Abstract Base class for Graphic Control.
//          Separated to easy change general ancestor Window class,
//       e.g. - TFrameWindow -> TMDIChild, TDecoratedFrame or TTinyCaption
//
class TBaseGraphWin : public TFrameWindow
{
public:
        TBaseGraphWin (TWindow* parent, char* title,
                TModule* inst = 0 );
  virtual LPSTR GetClassName () { return "BaseGraphWin"; };
  virtual void  TimerTick ()  {};          //-- default placeholder
  virtual void  Animate ()    {};          //-- default placeholder
        void  EvDestroy ();              //-- post message
         int   pwid;
protected:
       TWindow *mastwin;
       TModule *mastapp;

DECLARE_RESPONSE_TABLE ( TBaseGraphWin );
};

/*------- TGraphWin -------------------------------------------------- */

class TGraphWin : public TBaseGraphWin
{
```

```
protected:
     BOOL    Iconized;
     BOOL    Scaled;
     BOOL    Isotropic;
     BOOL    Decorated;
     BOOL    Paused;
public:
    TRect      Clnt;
    TPlotsList * PlotsList;            // Plots Elements
protected:
     int    pwLengthX, pwLengthY;      // Window Size in pixels
     int    pwLeft,   pwRight;         // Viewport coord rel Window
     int    pwTop,    pwBottom;        //      " " "
public:
          TGraphWin ( TWindow* parent, char* title,
                TModule* inst = 0 );
 virtual    ~TGraphWin ();
 virtual void  GetWindowClass( WNDCLASS& WndClass );
 virtual void  SetupWindow  ();
 virtual LPSTR GetClassName () { return "GraphicWindow"; };

protected:
   virtual void  ScaleTo ( int Xs, int Ys );

   virtual void  Paint ( TDC& dc, BOOL erase, TRect& rect );
   virtual void  EvSysCommand  ( UINT id, TPoint& pt);
   virtual void  EvLButtonDown ( UINT, TPoint& );
   virtual void  EvRButtonDown ( UINT, TPoint& );
   virtual void  EvSize       ( UINT, TSize& size );
private:
        void  DlgSettings ();
        void  DlgAddPlot  ();
        void  DlgDelPlot  ();
public:
        void  AssignList (TPlotsList * pl) { PlotsList = pl; };
        void  LinkPlot   (TBasePlot * bp)  { PlotsList -> Ins (bp); };
   virtual void  TimerTick ();
   virtual void  Animate ();

DECLARE_RESPONSE_TABLE (TGraphWin);
};

/*------- TLinesGraphWin ---------------------------------------------- */

class TLinesGraphWin : public TGraphWin
{
protected:
     int    pwNx,     pwNy;           // Axis Draw digits
     char   cwXfmt  [LNGH_FMT];       //  and its format
     char   cwYfmt  [LNGH_FMT];       //    " "
     num    rwHx,     rwHy;          //
     num    rwScaleX, rwScaleY;
     num    rwSizeX,  rwSizeY;
     num    rwStepX,  rwStepY;
```

```
    num    rwOrgX,   rwOrgY;        // Geometry limits in
    num    rwEndX,   rwEndY;        //   physical units
    char   cwName  [LNGH_TITLE];
    char   cwXname [LNGH_AXIS];
    char   cwYname [LNGH_AXIS];
    char   cwXunits [LNGH_UNITS];
    char   cwYunits [LNGH_UNITS];
public:
          TLinesGraphWin ( TWindow* parent, char* title,
                           axis* hrz,   axis* vrt,
                           TModule* inst = 0 );
 virtual    ~TLinesGraphWin ();

 virtual void DrawMesh     ( TDC& dc );
 virtual void DrawAxis     ( TDC& dc );
 virtual void DrawBounds   ( TDC& dc );
 virtual void DrawComments  ( TDC& dc );
protected:
  int  Icx ( num x )   { return (int)(rwScaleX*(x - rwOrgX)+pwLeft); };
  int  Icy ( num y )   { return (int)(rwScaleY*(rwEndY - y)+pwTop); };
  num  Xci ( int i )   { return (rwOrgX + rwStepX*(i-pwLeft)); };
  num  Yci ( int i )   { return (rwEndY - rwStepY*(i-pwTop)); };
   virtual void  ScaleTo ( int Xs, int Ys );

   virtual void  Paint ( TDC& dc, BOOL erase, TRect& rect );
   virtual void  EvLButtonDown ( UINT, TPoint& );
   virtual void  EvSize       ( UINT, TSize& size );
public:
   virtual void  TimerTick ();
   virtual void  Animate ();
protected:
      long   TextClr,
             NameClr,
             MeshClr,
             PlotClr;
      TPen  * TextPen,
            * MeshPen,
            * PlotPen;
      char   txt   [LNGH_COMMENT];
     TStatic * StaticControl;
      int    TextHeight;

DECLARE_RESPONSE_TABLE (TLinesGraphWin);
};

/*------- TRegionGraphWin ------------------------------------------------ */

class TRegionGraphWin : public TLinesGraphWin
{
private:
     int    pwMeshX,  pwMeshY;        // Mesh Dimension
 TBoundaryType pwLeftBnd, pwRightBnd,      //
           pwTopBnd,  pwBottomBnd;     //
public:
```

```
    TRegionGraphWin (TWindow* parent, char* title,
                TGeomAxis* hrz,  TGeomAxis* vrt,
                TModule* inst = 0 );
    TRegionGraphWin (TWindow* parent, char* title,
                TRectangleRgn* rgn,
                TModule* inst = 0 );
  virtual LPSTR GetClassName () { return "RegionPlot"; };

       void ExamplePaint  ( TDC& dc );
  virtual void DrawMesh     ( TDC& dc );
  virtual void DrawAxis     ( TDC& dc );
  virtual void DrawBounds   ( TDC& dc );
  virtual void DrawComments  ( TDC& dc );
protected:
         TPen* Pen (TBoundaryType ind);
         long  BndColor (TBoundaryType ind);
    virtual void  Paint ( TDC& dc, BOOL erase, TRect& rect );
protected:
      long   BndOpenClr,
             BndMetalClr,
             BndSymClr,
             BndPeriodClr;
      TPen  * BndOpenPen,
            * BndMetalPen,
            * BndSymPen,
            * BndPeriodPen;
};
//---------------------------------------------------------------------
#endif                                  // ifndef _PS_GRAPH_H
// ****** PS_GRAPH.CPP ******
#if defined (_SPRT_PLOTS_PCH)
////////////////////////////////////
#pragma hdrfile "d:\picsim\obj\_plots.csm"
 #include <stdlib.h>
 #include <stdio.h>
 #include <math.h>
 #include <typeinfo.h>
 #include <owl\dialog.h>
 #include <owl\checkbox.h>
 #include <owl\static.h>
 #include <owl\combobox.h>
 #include <owl\edit.h>
 #include <owl\groupbox.h>
 #include <owl\radiobut.h>
 #include <owl\dc.h>
 #include <owl\framewin.h>
 #include <owl\applicat.h>
#pragma hdrstop

#else
////////////////////////////////////
#pragma hdrfile "f:\picsim\obj\_allpch.csm"
 #define _ALLPCH
 #include <_allpch.h>
```

```
#pragma hdrstop

#endif
////////////////////////////////////

#include "ps_idres.rh"
#include "ps_dlg01.rh"
#include "ps_menu1.rh"
#include "ps_str01.rh"

#include "ps_graph.h"

static int WinCount = 0;
static int Debug = 0;

//*************************************************************************
struct TGraphDlgStruct1
{
   BOOL svDecor;
   BOOL svScale;
   BOOL svIsotr;
};
//-------------------------------------------------------------------------
class TGraphDlg1 : public TDialog
{
public:
    TGraphDlg1 (TWindow* parent, int resId, TGraphDlgStruct1 & ts);
};
 TGraphDlg1 :: TGraphDlg1 (TWindow* parent, int resId, TGraphDlgStruct1 & ts)
         : TDialog (parent, resId),  TWindow (parent)
{
  new TCheckBox (this, idc_check1, 0);
  new TCheckBox (this, idc_check2, 0);
  new TCheckBox (this, idc_check3, 0);
    TransferBuffer = (void far*)&ts;
};

//*************************************************************************
struct TGraphDlgStruct2
{
   TComboBoxData  svPlots;
};
//-------------------------------------------------------------------------
class TGraphDlg2 : public TDialog
{
public:
    TGraphDlg2 (TWindow* parent, int resId, TGraphDlgStruct2 & ts);
};
 TGraphDlg2 :: TGraphDlg2 (TWindow* parent, int resId, TGraphDlgStruct2 & ts)
         : TDialog (parent, resId),  TWindow (parent)
{
  new TComboBox (this, idc_combo1, MXCOMBO);
    TransferBuffer = (void far*)&ts;
};
```

```
/*************** TBaseGraphWin ---------------------------------- */

DEFINE_RESPONSE_TABLE1 (TBaseGraphWin,TWindow)
  EV_WM_DESTROY,
END_RESPONSE_TABLE;

//----------------------------------------------------------------
    TBaseGraphWin :: TBaseGraphWin (TWindow* parent, char* title,
                          TModule* inst )
              : TFrameWindow ( parent, title, 0, FALSE, inst )
{
 mastwin = parent;  mastapp = inst;   pwid = ++WinCount;
};
//----------------------------------------------------------------
void TBaseGraphWin::EvDestroy ()
{
 Parent->PostMessage ( WM_PW_CLOSED, pwid );
 TFrameWindow::EvDestroy();
};

//*********************************************************************

#define cms_Settings   201
#define cms_AddPlot    202
#define cms_DelPlot    203
#define cms_Revision   209

/*************** TGraphWin ---------------------------------- */

DEFINE_RESPONSE_TABLE1 (TGraphWin, TBaseGraphWin)
  EV_WM_SIZE,
  EV_WM_SYSCOMMAND,
  EV_WM_LBUTTONDOWN,
  EV_WM_RBUTTONDOWN,
END_RESPONSE_TABLE;

//----------------------------------------------------------------
TGraphWin :: TGraphWin (TWindow* parent, char* title,
                    TModule* inst )
        : TBaseGraphWin (parent, title, inst),
              PlotsList ( 0 )
{
 Attr.Style = WS_VISIBLE | WS_POPUP | WS_OVERLAPPEDWINDOW;
 Attr.X =  70 + 15*WinCount;
 Attr.Y =  50 + 10*WinCount;
 Attr.W = 500;
 Attr.H = 300;
Iconized = Scaled = Isotropic = Paused  = FALSE;  Decorated = TRUE;
pwLeft = 8*7;                pwRight = 8*9;
pwTop = 16*2;                pwBottom = 16*3;
  pwLengthX = Attr.W - (pwLeft + pwRight);              // work area width
  pwLengthY = Attr.H - (pwTop  + pwBottom);              // work area height
Clnt.left = pwLeft;    Clnt.right  = pwLeft + pwLengthX;
```

```
 Clnt.top  = pwTop;    Clnt.bottom = pwTop  + pwLengthY;
};
//---------------------------------------------------------------------
TGraphWin :: ~TGraphWin ()
{
 if (PlotsList)     delete PlotsList;
};
//---------------------------------------------------------------------
void TGraphWin::GetWindowClass( WNDCLASS& WndClass )
{
 TBaseGraphWin::GetWindowClass( WndClass );
 WndClass.hbrBackground = (HBRUSH) GetStockObject (BLACK_BRUSH);
// WndClass.hIcon = 0;
// WndClass.hIcon = mastapp -> LoadIcon ("idiIconPiCpp1");
 WndClass.hIcon = HICON (TIcon (HINSTANCE(*mastapp), idiIconPiCpp2));
};
//---------------------------------------------------------------------
void TGraphWin::SetupWindow ()
{
  HMENU hSysMnu;
  char captio [25];
  char numero [10];
//  int lc;
                    TBaseGraphWin::SetupWindow ();
     itoa ( WinCount, numero, 10 );
/* lc = */ GetWindowText (captio, 24);
  strcat (captio, " Win=");
  strcat (captio, numero);
                  SetWindowText (captio);
 hSysMnu = GetSystemMenu ( );
  AppendMenu (hSysMnu, MF_SEPARATOR, 0, NULL);
  AppendMenu (hSysMnu, MF_STRING, cms_Settings, "&Settings...");
  AppendMenu (hSysMnu, MF_SEPARATOR, 0, NULL);
  AppendMenu (hSysMnu, MF_STRING, cms_AddPlot,  "&Add Plot...");
  AppendMenu (hSysMnu, MF_STRING, cms_DelPlot,  "&Delete Plot...");
  AppendMenu (hSysMnu, MF_STRING, cms_Revision, "&Revision...");
};
//---------------------------------------------------------------------
void TGraphWin::EvSize( UINT sizeType, TSize& size )
{
 int NewXmax, NewYmax;
 TRect clnt = GetClientRect ();
              TBaseGraphWin::EvSize (sizeType, size);
 Invalidate (TRUE);
             NewXmax = clnt.right  - (pwLeft + pwRight);
             NewYmax = clnt.bottom - (pwTop  + pwBottom);
 if (sizeType == SIZE_MINIMIZED)
   {  if (!Iconized)   { Iconized = TRUE; }
   }
 else
   {  if (Iconized)  Iconized = FALSE;
      if (Scaled)    ScaleTo (NewXmax, NewYmax);
   };
 Clnt.left = pwLeft;   Clnt.right  = pwLeft + pwLengthX;
```

```
 Clnt.top  = pwTop;    Clnt.bottom = pwTop  + pwLengthY;
};
//---------------------------------------------------------------
void TGraphWin::EvSysCommand (UINT id, TPoint& pt)
{
          switch (id)
 {
 case cms_Settings:  DlgSettings ();          break;
 case cms_AddPlot :  DlgAddPlot  ();           break;
 case cms_DelPlot :  DlgDelPlot  ();          break;
 case cms_Revision:
   {
    if (PlotsList)          PlotsList -> Revision ();
     else   MessageBox ("PlotsList is Empty!","",MB_OK);
                            break;
   }
 default:  TBaseGraphWin::EvSysCommand (id, pt);
 }
};
//---------------------------------------------------------------
void TGraphWin::DlgAddPlot ()
{
 TGraphDlgStruct2 ds2;
       ds2.svPlots.AddStringItem ("Text Element",     0);
       ds2.svPlots.AddStringItem ("Axis Element",     1);
       ds2.svPlots.AddStringItem ("Boundary Element", 2);
       ds2.svPlots.AddStringItem ("Frame Element",    3);
       ds2.svPlots.AddStringItem ("Region Element",   4);
       ds2.svPlots.AddStringItem ("Mesh Element",     5);
       ds2.svPlots.AddStringItem ("Line Element",     6);
       ds2.svPlots.AddStringItem ("Func Element",     7);
       ds2.svPlots.AddStringItem ("GridValue Element", 8);
       ds2.svPlots.Select (0);
 if ( TGraphDlg2 (this, iddGraph2, ds2).Execute () == IDOK )
  {
    TBasePlot * NP = 0;
     int Sel = ds2.svPlots.GetSelIndex();
              switch ( Sel )
   { case 0 :  NP = new TTextPlot (this);        break;
     case 1 :  NP = new TAxisPlot (this);        break;
   }
        if ( NP ) PlotsList -> Ins (NP);
  }
};
//---------------------------------------------------------------
void TGraphWin::DlgDelPlot ()
{
 TGraphDlgStruct2 ds2;      int k = 0;     char buf [80];
        while (++k <= (PlotsList ->Ne))
   {       PlotsList -> Element ( buf, k );
       ds2.svPlots.AddStringItem ( buf, k-1 );
   }
       ds2.svPlots.Select (0);
 if ( TGraphDlg2 (this, iddGraph2, ds2).Execute () == IDOK )
```

```
  {
     int Sel = ds2.svPlots.GetSelIndex() + 1;
       if ( Sel > 0 ) PlotsList -> Del ( Sel );
  }
};
//---------------------------------------------------------------
void TGraphWin::DlgSettings ()
{
 TGraphDlgStruct1 ds1;
            ds1.svDecor = Decorated;
            ds1.svScale = Scaled;
            ds1.svIsotr = Isotropic;
  if ( TGraphDlg1 (this, iddGraph1, ds1).Execute () == IDOK )
   {
     Decorated  = ds1.svDecor;
     Scaled     = ds1.svScale;
     Isotropic  = ds1.svIsotr;
                  TRect clnt = GetClientRect ();
     int NewXmax = clnt.right  - (pwLeft + pwRight);
     int NewYmax = clnt.bottom - (pwTop  + pwBottom);
        if (Scaled) ScaleTo (NewXmax, NewYmax);
                       Invalidate (TRUE);
   }
};
//---------------------------------------------------------------
void TGraphWin::ScaleTo ( int Xs, int Ys )
{
                       if (Xs > 0) pwLengthX = Xs;
                       if (Ys > 0) pwLengthY = Ys;
 if (PlotsList) PlotsList -> ScaleTo ( pwLengthX, pwLengthY );
};
//---------------------------------------------------------------
void TGraphWin :: EvLButtonDown ( UINT, TPoint& )
{
 Paused = !Paused;
};
//---------------------------------------------------------------
void TGraphWin :: EvRButtonDown ( UINT, TPoint& )
{
 Scaled = !Scaled;
};
//---------------------------------------------------------------
void TGraphWin::TimerTick ()
{
// HDC dc;
   if (!Paused)
    {  // dc = GetDC (HWindow);
       if (PlotsList) PlotsList -> Draw ( TClientDC (HWindow) );
      //  ReleaseDC (HWindow, dc);
    }
};
//---------------------------------------------------------------
void TGraphWin::Animate ()
{
```

```
 HDC dc;
   if (!Iconized)
    { //  dc = GetDC (HWindow);
        if (PlotsList) PlotsList -> Redraw (TClientDC (HWindow));
//          ReleaseDC (HWindow, dc);
    }
};
//---------------------------------------------------------------
void TGraphWin::Paint ( TDC& dc, BOOL erase, TRect& rect )
{
 TBaseGraphWin::Paint (dc, erase, rect);
            dc.SetBkMode   (TRANSPARENT);
            dc.SetBkColor  (RGB(0,0,0));
            dc.SetTextAlign (TA_CENTER | TA_BASELINE | TA_NOUPDATECP );
//            dc.SetTextColor ( TextClr );
 if (!Iconized)
      {
  if (PlotsList) PlotsList -> Draw ( dc, 0 );
      }
};

//*********************************************************************
/**************** TLinesGraphWin ---------------------------------- */

DEFINE_RESPONSE_TABLE1 (TLinesGraphWin, TGraphWin)
  EV_WM_SIZE,
  EV_WM_LBUTTONDOWN,
END_RESPONSE_TABLE;

//---------------------------------------------------------------
TLinesGraphWin :: TLinesGraphWin (TWindow* parent, char* title,
                  axis * hrz,  axis * vrt,  TModule* inst )
          : TGraphWin (parent, title, inst)
{
  TextClr    = RGB ( 255, 255, 255 );
  NameClr    = RGB ( 127, 127,  0 );
  MeshClr    = RGB ( 127, 127, 127 );
  PlotClr    = RGB ( 255, 255, 127 );
          TextPen    = new TPen (TextClr);
          MeshPen    = new TPen (MeshClr,    1, PS_SOLID);
          PlotPen    = new TPen (PlotClr,    1, PS_SOLID);
if (!hrz) MessageBox ("Horiz Axis NULL pointer","Error");
if (!vrt) MessageBox ("Vert Axis NULL pointer","Error");
    if ((hrz)&&(vrt))
 {
rwOrgX = hrz->org;          rwOrgY = vrt->org;
rwEndX = hrz->end;          rwEndY = vrt->end;
 strcpy (cwXname, hrz->name);   strcpy (cwYname, vrt->name);
 strcpy (cwXunits, hrz->units);   strcpy (cwYunits, vrt->units);
    pwNx = ((hrz->N > 0)&&(hrz->N < 51))? hrz->N : -1;
    pwNy = ((vrt->N > 0)&&(vrt->N < 51))? vrt->N : -1;
 }
     else
 {
```

```
rwOrgX =  0.0;            rwOrgY =  0.0;
rwEndX = 10.0;            rwEndY = 10.0;
 strcpy (cwXname, "Îñü X");    strcpy (cwYname, "Îñü Y");
 strcpy (cwXunits,"cm");       strcpy (cwYunits,"m");
   pwNx = -1;      pwNy = -1;
 }
rwSizeX = rwEndX - rwOrgX;     rwSizeY = rwEndY - rwOrgY;
       if (rwSizeX <= 0.0) rwSizeX = 1.0;  //d
       if (rwSizeY <= 0.0) rwSizeY = 1.0;  //dd
pwLeft = 8*7;                pwRight = 8*9;
pwTop = 16*2;                pwBottom = 16*3;
 pwLengthX = Attr.W - (pwLeft + pwRight);            // work area width
 pwLengthY = Attr.H - (pwTop  + pwBottom);           // work area height
rwScaleX = pwLengthX/rwSizeX;  rwScaleY = pwLengthY/rwSizeY;
rwStepX = 1.0/rwScaleX;       rwStepY = 1.0/rwScaleY;
rwHx = rwSizeX / pwNx;        rwHy = rwSizeY / pwNy;
 strcpy (cwXfmt, "%.2f");     strcpy (cwYfmt, "%.2f");
 strcpy (cwName, " Lines Plot ");
StaticControl = new TStatic (this, 100, cwName, 10, 10, 10, 10, 0);
};
//---------------------------------------------------------------------
TLinesGraphWin :: ~TLinesGraphWin ()
{
 delete StaticControl;
// if (PlotsList) delete PlotsList;
};
//---------------------------------------------------------------------
void TLinesGraphWin::EvSize( UINT sizeType, TSize& size )
{
 TRect clnt = GetClientRect ();
             TGraphWin::EvSize (sizeType, size);
 Invalidate (TRUE);
 if ((StaticControl)&&(!Iconized))
    StaticControl -> MoveWindow ( 0, 0, clnt.right, 16, TRUE );
};
//---------------------------------------------------------------------
void TLinesGraphWin::ScaleTo ( int Xs, int Ys )
{
      TGraphWin :: ScaleTo ( Xs, Ys );
rwScaleX = pwLengthX/rwSizeX;  rwScaleY = pwLengthY/rwSizeY;
rwStepX = 1.0/rwScaleX;       rwStepY = 1.0/rwScaleY;
};
//---------------------------------------------------------------------
void TLinesGraphWin :: EvLButtonDown ( UINT, TPoint& )
{
 delete StaticControl;
 strcpy (txt, " Ìîëîäåö !");
 StaticControl = new TStatic(this, 100, txt,10,10,10,10,0);
 Paused = !Paused;
};
//---------------------------------------------------------------------
void TLinesGraphWin::TimerTick ()
{
 if (!Paused) TGraphWin :: TimerTick ();
```

```
};
//---------------------------------------------------------------
void TLinesGraphWin::Animate ()
{
 if (!Paused) TGraphWin :: Animate ();
};
//---------------------------------------------------------------
void TLinesGraphWin::DrawBounds ( TDC& dc )
{
                        dc.MoveTo ( pwLeft, pwTop );
 dc.SelectObject (*TextPen);
    dc.LineTo ( pwLeft+pwLengthX, pwTop );
    dc.LineTo ( pwLeft+pwLengthX, pwTop+pwLengthY );
    dc.LineTo ( pwLeft, pwTop+pwLengthY );
    dc.LineTo ( pwLeft, pwTop );
 dc.RestorePen ();
};
//---------------------------------------------------------------
void TLinesGraphWin::DrawAxis ( TDC& dc )
{
 char txt [20];
            dc.SetTextAlign (TA_CENTER | TA_TOP );
/* H(m) */    sprintf (txt,"%s (%s)", cwXname, cwXunits);
  dc.TextOut (pwLeft+pwLengthX/2, pwTop+pwLengthY+22, txt, strlen(txt));
/* -5.0 */    sprintf (txt, cwXfmt, rwOrgX);
  dc.TextOut (pwLeft+1, pwTop+pwLengthY+4, txt, strlen(txt));
/* 10.0 */    sprintf (txt, cwXfmt, rwEndX);
  dc.TextOut (pwLeft+pwLengthX-1, pwTop+pwLengthY+4, txt, strlen(txt));
/*  0 */  if (rwOrgX*rwEndX < 0.0)
  dc.TextOut (Icx(0.0), pwTop+pwLengthY+4, "0.0", 3);
            dc.SetTextAlign (TA_LEFT | TA_BASELINE );
/* V */    sprintf (txt,"%s", cwYname);
  dc.TextOut (pwLeft+pwLengthX+12, pwTop+pwLengthY/2, txt, strlen(txt));
/* (m) */    sprintf (txt,"(%s)", cwYunits);
  dc.TextOut (pwLeft+pwLengthX+6, pwTop+pwLengthY/2+16, txt, strlen(txt));
            dc.SetTextAlign (TA_RIGHT | TA_BASELINE );
/* -1 */    sprintf (txt, cwYfmt, rwOrgY);
  dc.TextOut (pwLeft-3, pwTop+pwLengthY, txt, strlen(txt));
/*  1 */    sprintf (txt, cwYfmt, rwEndY);
  dc.TextOut (pwLeft-3, pwTop, txt, strlen(txt));
/*  0 */  if (rwOrgY*rwEndY < 0.0)
  dc.TextOut (pwLeft-3, Icy(0.0), "0.0", 3);
            dc.SetTextAlign (TA_CENTER | TA_BASELINE | TA_NOUPDATECP );
};
//---------------------------------------------------------------
void TLinesGraphWin::DrawMesh ( TDC& dc )
{
 int i;
                        dc.SelectObject (*MeshPen);
   for ( i=1; i < pwNy; i++)
 {      dc.MoveTo ( pwLeft,          Icy (rwOrgY + i*rwHy) );
        dc.LineTo ( pwLeft+pwLengthX, Icy (rwOrgY + i*rwHy) );
 }
   for ( i=1; i < pwNx; i++)
```

```
 {      dc.MoveTo ( Icx (rwOrgX + i*rwHx), pwTop );
        dc.LineTo ( Icx (rwOrgX + i*rwHx), pwTop+pwLengthY );
 }
                        dc.RestorePen ();
};
//---------------------------------------------------------------
void TLinesGraphWin::DrawComments ( TDC& dc )
{
 char txt [] = "1D ";
 double d, w, a, h;
 num x, y;
// TRect rct (0, 12, pwLeft+pwRight+pwLengthX, 28);
//          dc.SetTextAlign (TA_CENTER);
//          dc.SetTextColor (NameClr);
//       strcat (txt, cwName);
// dc.ExtTextOut (1,1, ETO_OPAQUE, &rct, txt, strlen(txt));

    h = rwSizeX / pwLengthX;
    a = rwSizeY * 0.25;
    w = 6.28 * 8 /rwSizeX;
                    dc.MoveTo ( pwLeft, Icy (0.0) );
 dc.SelectObject (*PlotPen);
   for ( int i=1; i < pwLengthX; i++)
 {                       a *=  0.9;
                        x = num (rwOrgX + i * h);
                        y = num (a * sin (w * x));
 dc.LineTo ( Icx ( x ), Icy ( y ) );
 }
 dc.RestorePen ();
};
//---------------------------------------------------------------
void TLinesGraphWin::Paint ( TDC& dc, BOOL erase, TRect& rect )
{
 TGraphWin::Paint (dc, erase, rect);
            dc.SetBkMode    (TRANSPARENT);
            dc.SetBkColor   (RGB(0,0,0));
            dc.SetTextAlign (TA_CENTER | TA_BASELINE | TA_NOUPDATECP );
            dc.SetTextColor ( TextClr );
 if (!Iconized)
    {
                DrawBounds ( dc );
                DrawAxis ( dc );
                DrawMesh ( dc );
     if (Isotropic) DrawComments ( dc );
// - - - - - - - - - - - - - - - - - - - - - - - - - -
     if (PlotsList) PlotsList -> Draw (dc);
    }
};
//*********************************************************************
/**************** TRegionGraphWin ---------------------------------- */
//---------------------------------------------------------------

TRegionGraphWin :: TRegionGraphWin (TWindow* parent, char* title,
                        TGeomAxis * hrz, TGeomAxis * vrt,
```

```
                                TModule* inst )
            : TLinesGraphWin (parent, title, hrz, vrt, inst)
{
   BndOpenClr  = RGB (  0,   0, 255 );
   BndMetalClr = RGB ( 255,  0,  0 );
   BndSymClr   = RGB ( 127, 255,  0 );
   BndPeriodClr = RGB (  0, 127, 127 );
          BndOpenPen  = new TPen (BndOpenClr,  1, PS_DASH);
          BndMetalPen = new TPen (BndMetalClr, 2, PS_SOLID);
          BndSymPen   = new TPen (BndSymClr,   1, PS_DASHDOT);
          BndPeriodPen = new TPen (BndPeriodClr, 1, PS_SOLID);
   if ((hrz) && (vrt))
  {
pwMeshX    = (hrz->N);                pwMeshY    = (vrt->N);
pwLeftBnd  = TBoundaryType(hrz->bound1);  pwRightBnd = TBoundaryType (hrz->bound2);
pwBottomBnd = TBoundaryType(vrt->bound1);  pwTopBnd  = TBoundaryType (vrt->bound2);
   }
      else
   {
 pwMeshX    = 20;     pwMeshY    = 10;
 pwLeftBnd  =  1;     pwRightBnd =  3;
 pwBottomBnd =  2;     pwTopBnd  =  4;
   }
      pwNx = ((pwMeshX > 0)&&(pwMeshX < 151))? pwMeshX : 10;
      pwNy = ((pwMeshY > 0)&&(pwMeshY < 151))? pwMeshY : 10;
 rwHx = rwSizeX / pwNx;        rwHy = rwSizeY / pwNy;
  strcpy (cwName, "Two Axis and Homogeneous Mesh");
    wsprintf (txt," Regular Mesh = %d x %d", pwMeshX, pwMeshY);
StaticControl = new TStatic(this, 100, txt,10,10,10,10,0);
};
//---------------------------------------------------------------
TRegionGraphWin :: TRegionGraphWin (TWindow* parent, char* title,
                        TRectangleRgn * rgn, TModule* inst )
         : TLinesGraphWin (parent, title, rgn->X1(), rgn->X2(), inst )
{
 TGeomAxis * hrz = rgn -> X1();
 TGeomAxis * vrt = rgn -> X2();
   BndOpenClr  = RGB (  0,   0, 255 );
   BndMetalClr = RGB ( 255,  0,  0 );
   BndSymClr   = RGB ( 127, 255,  0 );
   BndPeriodClr = RGB (  0, 127, 127 );
          BndOpenPen  = new TPen (BndOpenClr,  1, PS_DASH);
          BndMetalPen = new TPen (BndMetalClr, 2, PS_SOLID);
          BndSymPen   = new TPen (BndSymClr,   1, PS_DASHDOT);
          BndPeriodPen = new TPen (BndPeriodClr, 1, PS_SOLID);
   if ((hrz) && (vrt))
  {
pwMeshX    = hrz->N;                pwMeshY    = vrt->N;
pwLeftBnd  = TBoundaryType(hrz->bound1);  pwRightBnd = TBoundaryType (hrz->bound2);
pwBottomBnd = TBoundaryType(vrt->bound1);  pwTopBnd  = TBoundaryType (vrt->bound2);
   }
      else
   {
 pwMeshX    = 20;    pwMeshY    = 10;
```

```
 pwLeftBnd  =  1;   pwRightBnd =  3;
 pwBottomBnd =  2;   pwTopBnd  =  4;
  }
     pwNx = ((pwMeshX > 0)&&(pwMeshX < 151))? pwMeshX : 10;
     pwNy = ((pwMeshY > 0)&&(pwMeshY < 151))? pwMeshY : 10;
 rwHx = rwSizeX / pwNx;        rwHy = rwSizeY / pwNy;
  strcpy (cwName, "Rectangle Region and Homogeneous Mesh");
   wsprintf (txt," Rectangle Mesh = %d x %d", pwMeshX, pwMeshY);
StaticControl = new TStatic(this, 100, txt,10,10,10,10,0);
};
//---------------------------------------------------------------
long TRegionGraphWin::BndColor ( TBoundaryType ind )
{              switch (ind)
  { case ConductingBnd:    return (BndMetalClr);
    case OpenBnd:          return (BndOpenClr);
    case AxisymmetricBnd:  return (BndSymClr);
    case PeriodicBnd:      return (BndPeriodClr);
     default:              return (TextClr);
  }
};
//---------------------------------------------------------------
TPen* TRegionGraphWin::Pen ( TBoundaryType ind )
{              switch (ind)
  { case ConductingBnd:    return (BndMetalPen);
    case OpenBnd:          return (BndOpenPen);
    case AxisymmetricBnd:  return (BndSymPen);
    case PeriodicBnd:      return (BndPeriodPen);
     default:              return (TextPen);
  }
};
//---------------------------------------------------------------
void TRegionGraphWin::ExamplePaint ( TDC& dc )
{
 TRect rct = GetClientRect();
     rct.Inflate(-2, 0);
 if (!Iconized)
       dc.DrawText(txt, strlen(txt), rct, DT_WORDBREAK);
                    dc.MoveTo ( 10, 30 );
 dc.SelectObject (*BndSymPen);    dc.LineTo ( pwLengthX-10,         30 ); // dc.RestorePen ();
 dc.SelectObject (*BndOpenPen);   dc.LineTo ( pwLengthX-10, pwLengthY-10 ); // dc.RestorePen ();
 dc.SelectObject (*BndMetalPen);  dc.LineTo (         10, pwLengthY-10 ); // dc.RestorePen ();
 dc.SelectObject (*BndPeriodPen); dc.LineTo (         10,         30 ); // dc.RestorePen ();
 dc.RestorePen ();
};
//---------------------------------------------------------------
void TRegionGraphWin::DrawBounds ( TDC& dc )
{
                    dc.MoveTo ( pwLeft, pwTop );
 dc.SelectObject (*Pen (pwTopBnd));
    dc.LineTo ( pwLeft+pwLengthX, pwTop );          dc.RestorePen ();
 dc.SelectObject (*Pen (pwRightBnd));
    dc.LineTo ( pwLeft+pwLengthX, pwTop+pwLengthY );  dc.RestorePen ();
 dc.SelectObject (*Pen (pwBottomBnd));
    dc.LineTo ( pwLeft, pwTop+pwLengthY );          dc.RestorePen ();
```

```
 dc.SelectObject (*Pen (pwLeftBnd));
    dc.LineTo ( pwLeft, pwTop );                    dc.RestorePen ();
};
//-----------------------------------------------------------------
void TRegionGraphWin::DrawAxis ( TDC& dc )
{
 char txt [20];
            dc.SetTextAlign (TA_CENTER | TA_TOP );
/* H(m) */    sprintf (txt,"%s (%s)", cwXname, cwXunits);
  dc.TextOut (pwLeft+pwLengthX/2, pwTop+pwLengthY+22, txt, strlen(txt));
/* -5.0 */    sprintf (txt, cwXfmt, rwOrgX);
  dc.TextOut (pwLeft+1, pwTop+pwLengthY+10, txt, strlen(txt));
/* 10.0 */    sprintf (txt, cwXfmt, rwEndX);
  dc.TextOut (pwLeft+pwLengthX-1, pwTop+pwLengthY+10, txt, strlen(txt));
/*  0 */  if (rwOrgX*rwEndX < 0.0)
  dc.TextOut (Icx(0.0), pwTop+pwLengthY+10, "0.0", 3);
            dc.SetTextAlign (TA_LEFT | TA_BASELINE );
/* V */    sprintf (txt,"%s", cwYname);
  dc.TextOut (pwLeft+pwLengthX+24, pwTop+pwLengthY/2, txt, strlen(txt));
/* (m) */    sprintf (txt,"(%s)", cwYunits);
  dc.TextOut (pwLeft+pwLengthX+18, pwTop+pwLengthY/2+16, txt, strlen(txt));
            dc.SetTextAlign (TA_RIGHT | TA_BASELINE );
/* -1 */    sprintf (txt, cwYfmt, rwOrgY);
  dc.TextOut (pwLeft-3, pwTop+pwLengthY-4, txt, strlen(txt));
/*  1 */    sprintf (txt, cwYfmt, rwEndY);
  dc.TextOut (pwLeft-3, pwTop+2, txt, strlen(txt));
/*  0 */  if (rwOrgY*rwEndY < 0.0)
  dc.TextOut (pwLeft-3, Icy(0.0), "0.0", 3);
            dc.SetTextAlign (TA_CENTER | TA_BASELINE | TA_NOUPDATECP );
};
//-----------------------------------------------------------------
void TRegionGraphWin::DrawMesh ( TDC& dc )
{
 int i;
                        dc.SelectObject (*MeshPen);
   for ( i=1; i < pwNy; i++)
 {      dc.MoveTo ( pwLeft,         Icy (rwOrgY + i*rwHy) );
        dc.LineTo ( pwLeft+pwLengthX, Icy (rwOrgY + i*rwHy) );
 }
   for ( i=1; i < pwNx; i++)
 {      dc.MoveTo ( Icx (rwOrgX + i*rwHx), pwTop );
        dc.LineTo ( Icx (rwOrgX + i*rwHx), pwTop+pwLengthY );
 }
                        dc.RestorePen ();
};
//-----------------------------------------------------------------
void TRegionGraphWin::DrawComments ( TDC& dc )
{
 char txt [20];
 TRect rct (0, 0, pwLeft+pwRight+pwLengthX, pwTop/2);

 strcpy (txt, string (*GetModule(), ids_boundary00+pwTopBnd).c_str () );
  dc.SetTextColor ( BndColor (pwTopBnd) );        // Top
dc.TextOut (pwLeft+(pwLengthX/2), pwTop, txt, strlen(txt));
```

```
  strcpy (txt, string (*GetModule(), ids_boundary00+pwBottomBnd).c_str () );
   dc.SetTextColor ( BndColor (pwBottomBnd) );     // Bottom
dc.TextOut (pwLeft+pwLengthX/2, pwTop+pwLengthY+10, txt, strlen(txt));
            dc.SetTextAlign (TA_LEFT | TA_BASELINE );
  strcpy (txt, string (*GetModule(), ids_boundary00+pwLeftBnd).c_str () );
   dc.SetTextColor ( BndColor (pwLeftBnd) );       // Left
dc.TextOut (2, pwTop+pwLengthY/2-16, txt, strlen(txt));
  strcpy (txt, string (*GetModule(), ids_boundary00+pwRightBnd).c_str () );
   dc.SetTextColor ( BndColor (pwRightBnd) );      // Right
dc.TextOut (pwLeft+pwLengthX-12, pwTop+pwLengthY/2-16, txt, strlen(txt));
//          dc.SetTextAlign (TA_CENTER);
//          dc.SetTextColor (NameClr);
//dc.ExtTextOut (1,1, ETO_OPAQUE, &rct, cwName, strlen(cwName));
};
//----------------------------------------------------------------
void TRegionGraphWin::Paint ( TDC& dc, BOOL erase, TRect& rect )
{
 TLinesGraphWin::Paint (dc, erase, rect);
            dc.SetBkMode    (TRANSPARENT);
            dc.SetBkColor   (RGB(0,0,0));
            dc.SetTextAlign (TA_CENTER | TA_BASELINE | TA_NOUPDATECP );
            dc.SetTextColor ( TextClr );
 if (!Iconized)
    {
//              DrawBounds ( dc );
//              DrawAxis ( dc );
//              DrawMesh ( dc );
   if (Decorated) DrawComments ( dc );
    }
};
//*****************************************************************
/*  PS_PLOTS.H  */
//                      Plot Element objects header
//
#ifndef _PS_PLOTS_H
#define _PS_PLOTS_H

#include <owl\point.h>

#include "ps_phval.h"
#include "ps_regn.h"
#include "ps_idres.rh"

class _OWLCLASS TDC;
class _OWLCLASS TPen;
class _OWLCLASS TStatic;
struct TFrameStruct;
class _OWLCLASS TWindow;

/*------- TBasePlot ---------------------------------------------- */

class TBasePlot
{
protected:
```

```
    int   pwX1,    pwY1;          // Viewport Org coord rel Window
    int   pwX2,    pwY2;        // Viewport End coord rel Window
    int   pwLengthX, pwLengthY;       // Viewport Size
    num    rwScaleX,  rwScaleY;
    num    rwSizeX,   rwSizeY;
    num    rwStepX,   rwStepY;
    num    rwOrgX,    rwOrgY;         // Phys limits in
    num    rwEndX,    rwEndY;         // physical units
public:
       TBasePlot ( TRect& x, TBasePlot* p = 0 );
       TBasePlot ( );
 virtual ~TBasePlot ( );
         int    id;
         char   ClassName [80];
protected:
  int  Icx ( num x )   { return (int)(rwScaleX*(x - rwOrgX)+pwX1); };
  int  Icy ( num y )   { return (int)(rwScaleY*(rwEndY - y)+pwY1); };
  num  Xci ( int i )   { return (rwOrgX + rwStepX*(i-pwX1)); };
  num  Yci ( int i )   { return (rwEndY - rwStepY*(i-pwY1)); };
public:
 virtual void  ScaleTo ( int Xs, int Ys );
 virtual void  Draw  ( TDC & dc )          { };
 virtual void  Erase ( TDC & dc )          { };
       void  Adjust ( int Adj = 0 );
 virtual void  IdClassMessage ( );
 virtual void  PrivateInfo (char* info)      { };
              BOOL    Static;
            TBasePlot * next;
              long    Color;
              TPen  * Pen;
DECLARE_CASTABLE;
};

/*--- TTextPlot ---------------------------------------------------- */
class TTextPlot : public TBasePlot
{
     char   Text   [LNGH_COMMENT];
    TStatic * sctl;
     int    Htxt,  Ltxt;
public:
       TTextPlot ( char* text, TRect& x, TBasePlot* p = 0 );
       TTextPlot ( TWindow* parent, TBasePlot* p = 0 );
 virtual ~TTextPlot ( ) { if (sctl) delete sctl; TBasePlot::~TBasePlot (); };

 virtual void  Draw  ( TDC & dc );
 virtual void  Erase ( TDC & dc );
 virtual void  PrivateInfo (char* info);

DECLARE_CASTABLE;
};

/*--- TAxisPlot ---------------------------------------------------- */
class TAxisPlot : public TBasePlot
{
```

```
public:
    axis * Axe;
    int   Orient;
protected:
    int    pwN;                 // Axe Draw digits
    char   cwFmt   [LNGH_FMT];     // and its format
    num    rwSize, rwH;
    char   cwAname [LNGH_AXIS];
    char   cwUnits [LNGH_UNITS];
public:
    TAxisPlot ( axis* e, TRect& x, int ori = 0, TBasePlot* p = 0 );
    TAxisPlot ( TWindow* parent, TBasePlot* p = 0 );
 virtual void  Draw  ( TDC & dc );
 virtual void  Erase ( TDC & dc );
 virtual void  ScaleTo ( int Xs, int Ys );
 virtual void  PrivateInfo (char* info);

DECLARE_CASTABLE;
};

/*--- TBoundsPlot ---------------------------------------------------- */
class TBoundsPlot : public TAxisPlot
{
public:
    TBoundaryType * Bnd;
public:
    TBoundsPlot ( axis* e, TRect& x, int ori = 0, TBasePlot* p = 0 );
 virtual void  Draw  ( TDC & dc );
 virtual void  Erase ( TDC & dc );
 virtual void  PrivateInfo (char* info);

DECLARE_CASTABLE;
};

/*--- TLinesPlot ----------------------------------------------------- */
class TLinesPlot : public TBasePlot
{
protected:
    TAbstrPhysVal  * PhVal;
        int    pwN;
public:
    TLinesPlot ( TAbstrPhysVal* e, TRect& x, TBasePlot* p = 0 );
    TLinesPlot ( TWindow* parent, TBasePlot* p = 0 );
 virtual void  Draw  ( TDC & dc );
 virtual void  Erase ( TDC & dc );
 virtual void  PrivateInfo (char* info);

DECLARE_CASTABLE;
};

/*--- TPointsPlot ---------------------------------------------------- */
class TPointsPlot : public TLinesPlot
{
public:
```

```
    TPointsPlot ( TAbstrPhysVal* e, TRect& x, TBasePlot* p = 0 )
                    : TLinesPlot ( e, x, p )  { };
 virtual void  Draw  ( TDC & dc );
 virtual void  Erase ( TDC & dc );

DECLARE_CASTABLE;
};

/*--- TFuncPlot ------------------------------------------------- */
class TFuncPlot : public TLinesPlot
{
    array A;
public:
    TFuncPlot ( TFunctionVal* e, TRect& x, TBasePlot* p = 0 );
    TFuncPlot ( TWindow* parent, TBasePlot* p = 0 );
 virtual void  Draw  ( TDC & dc );
 virtual void  Erase ( TDC & dc );

DECLARE_CASTABLE;
};

/*--- TArrayPlot ------------------------------------------------- */
class TArrayPlot : public TLinesPlot
{
    array A;
public:
    TArrayPlot ( TGridVal* e, TRect& x, TBasePlot* p = 0 );
    TArrayPlot ( TWindow* parent, TBasePlot* p = 0 );
 virtual void  Draw  ( TDC & dc );
 virtual void  Erase ( TDC & dc );

DECLARE_CASTABLE;
};

/*--- TFramePlot ------------------------------------------------- */
class TFramePlot : public TBasePlot
{
protected:
    int   pwNx,    pwNy;          // Axis Draw digits
    char  cwXfmt  [LNGH_FMT];     //  and its format
    char  cwYfmt  [LNGH_FMT];     //   " "
    int   pwMeshX, pwMeshY;       // Mesh Dimension
    num   rwHx,    rwHy;          //
    char  cwName  [LNGH_TITLE];
    char  cwXname [LNGH_AXIS];
    char  cwYname [LNGH_AXIS];
    char  cwXunits [LNGH_UNITS];
    char  cwYunits [LNGH_UNITS];
public:
      TFramePlot ( TFrameStruct* e, TRect& x, TBasePlot* p = 0 );
      TFramePlot ( TWindow* parent, TBasePlot* p = 0 );
 virtual ~TFramePlot ();

 virtual void  Draw  ( TDC & dc );
```

```
 virtual void  Erase ( TDC & dc );
 virtual void  PrivateInfo (char* info);
protected:
      long   FramClr,
             AxisClr,
             MeshClr,
             TextClr;
      TPen  * FramPen,
            * AxisPen,
            * MeshPen;

DECLARE_CASTABLE;
};

/*--- TRegionPlot ---------------------------------------------------- */
class TRegionPlot : public TBasePlot
{
public:
    TAbstrRegion * Rgn;
protected:
    int    pwNx,     pwNy;          // Axis Draw digits
    char   cwXfmt  [LNGH_FMT];      //  and its format
    char   cwYfmt  [LNGH_FMT];      //    " "
    num    rwHx,     rwHy;         //
    char   cwName  [LNGH_TITLE];
    char   cwXname  [LNGH_AXIS];
    char   cwYname  [LNGH_AXIS];
    char   cwXunits [LNGH_UNITS];
    char   cwYunits [LNGH_UNITS];
public:
       TRegionPlot ( TAbstrRegion* e, TRect& x, TBasePlot* p = 0 );
       TRegionPlot ( TWindow* parent, TBasePlot* p = 0 );
 virtual ~TRegionPlot ();

 virtual void  Draw  ( TDC & dc );
 virtual void  Erase ( TDC & dc );
 virtual void  PrivateInfo (char* info);
protected:
        TPen* BndPen (TBoundaryType ind);
        long  BndColor (TBoundaryType ind);
protected:
      long   BndOpenClr,
             BndMetalClr,
             BndSymClr,
             BndPeriodClr;
      TPen  * BndOpenPen,
            * BndMetalPen,
            * BndSymPen,
            * BndPeriodPen;
private:
    int    pwMeshX,  pwMeshY;     // Mesh Dimension
 TBoundaryType pwLeftBnd, pwRightBnd,   //
          pwTopBnd,  pwBottomBnd;  //
```

```
DECLARE_CASTABLE;
};

/*--- TMeshPlot ------------------------------------------------- */
class TMeshPlot : public TRegionPlot
{
public:
          TAbstrMesh * Mesh;
     TMeshPlot ( TAbstrMesh* e, TRect& x, TBasePlot* p = 0 );
     TMeshPlot ( TWindow* parent, TBasePlot* p = 0 );
  virtual void  Draw  ( TDC & dc );
  virtual void  Erase ( TDC & dc );
  virtual void  PrivateInfo (char* info);

DECLARE_CASTABLE;
};

////////////////////////////////////////////////////////////////////
/*------- TPlotsList ----------------------------------------------- */

class TPlotsList
{
    TBasePlot * First;
public:
         TPlotsList (): First ( 0 )   { Ne = 0; };
         TPlotsList (TBasePlot * frst);
        ~TPlotsList ();
  void  Ins ( TBasePlot * Plot );
  void  Del ( TBasePlot * Plot );
  void  Del ( int k );
  void  ScaleTo ( int Xs, int Ys );
  void  Draw   ( TDC & dc, BOOL NonStaticOnly = 1 );
  void  Redraw ( TDC & dc, BOOL NonStaticOnly = 1 );
  void  Revision  ( BOOL NonStaticOnly = 0 );
   int  Element ( char* item, int k );
        int  Ne;
private:
  void  ForEach ( void (*Procedure) (TBasePlot * ) );
};

////////////////////////////////////////////////////////////////////
//   Structures for Transfer Data between Dialog, Plot and Base classes
//*********************************************************************
struct TAxisStruct
{
 TAxisStruct ( );
 TAxisStruct ( axis & A );
void  GetData ( axis & A );
   char svName  [LNGH_AXIS];       // chr.
   char svOrg   [LNGH_FLOAT];      // num.
   char svEnd   [LNGH_FLOAT];      // num.
   char svN     [LNGH_INT];       // int. Axis Draw digits
   char svM     [LNGH_INT];       // int. Axis Draw mesh
   char svFmt   [LNGH_FMT];       // chr.  and its format
```

```
    char svH    [LNGH_FLOAT];     // num.
    char svUnits [LNGH_UNITS];     // chr.
    char svX1   [LNGH_INT];       // int.  Win coords in pics
    char svY1   [LNGH_INT];       // int.    "
    char svX2   [LNGH_INT];       // int.    "
    char svY2   [LNGH_INT];       // int.    "
};
//*************************************************************************
struct TFrameStruct
{
 TFrameStruct ();
 TFrameStruct ( axis & Hrz, axis & Vrt );
 void GetData  ( axis & Hrz, axis & Vrt );
    TAxisStruct X;
    TAxisStruct Y;
    char svName [LNGH_TITLE];
};

//-------------------------------------------------------------------------//
#endif  // ifndef _PS_PLOTS_H
// *********  PS_PLOTS.CPP ***********
#if defined (_SPRT_PLOTS_PCH)
////////////////////////////////////
#pragma hdrfile "d:\picsim\obj\_plots.csm"
 #include <stdlib.h>
 #include <stdio.h>
 #include <math.h>
 #include <typeinfo.h>
 #include <owl\dialog.h>
 #include <owl\checkbox.h>
 #include <owl\static.h>
 #include <owl\combobox.h>
 #include <owl\edit.h>
 #include <owl\groupbox.h>
 #include <owl\radiobut.h>
 #include <owl\dc.h>
 #include <owl\framewin.h>
 #include <owl\applicat.h>
#pragma hdrstop

#else
////////////////////////////////////
#pragma hdrfile "f:\picsim\obj\_allpch.csm"
 #define _ALLPCH
 #include <_allpch.h>
#pragma hdrstop

#endif
////////////////////////////////////

#include "ps_idres.rh"
//#include "ps_menu1.rh"
#include "ps_str01.rh"
#include "ps_dlg01.rh"
```

```
#include "ps_phval.h"
#include "ps_mesh.h"

#include "ps_plots.h"

////////////////////////////////////////////////////////////////////
//*****************************************************************
struct TTextPlotStr
{
 TTextPlotStr ();
  char svText [LNGH_COMMENT];
  char svX [5], svY [5];
  BOOL svAx1,  svAxc,  svAx2;
  BOOL svAy1,  svAyc,  svAy2;
  char svR [4], svG [4], svB [4];
};
 TTextPlotStr :: TTextPlotStr ()
{
 svText [0] = 0;  strcpy (svX, "5"); strcpy (svY, "5");
strcpy (svR, "255"); strcpy (svG, "255"); strcpy (svB, "255");
 svAx1 =  svAx2 = svAyc = svAy2 = 0;  svAxc = svAy1 = 1;
};
//-----------------------------------------------------------------
class TTextPlotDlg : public TDialog
{
public:
    TTextPlotDlg (TWindow* p, TTextPlotStr & ts, int resId = iddPlot01);
};
 TTextPlotDlg :: TTextPlotDlg (TWindow* p, TTextPlotStr & ts, int resId)
         : TDialog (p, resId)
{
     new TEdit (this, idc_edit01,  LNGH_COMMENT);
     new TEdit (this, idc_edit02,  5);
     new TEdit (this, idc_edit03,  5);
 TGroupBox * ivGroup1 = new TGroupBox (this, idc_group1);
               new TRadioButton (this, idc_radio1,  ivGroup1);
               new TRadioButton (this, idc_radio2,  ivGroup1);
               new TRadioButton (this, idc_radio3,  ivGroup1);
 TGroupBox * ivGroup2 = new TGroupBox (this, idc_group2);
               new TRadioButton (this, idc_radio4,  ivGroup2);
               new TRadioButton (this, idc_radio5,  ivGroup2);
               new TRadioButton (this, idc_radio6,  ivGroup2);
     new TEdit (this, idc_edit04,  4);
     new TEdit (this, idc_edit05,  4);
     new TEdit (this, idc_edit06,  4);

 TransferBuffer = (void far*)&ts;
};
//*****************************************************************
////////////////////////////////////////////////////////////////////

IMPLEMENT_CASTABLE1 ( TBasePlot,   TWindow );
```

```
IMPLEMENT_CASTABLE1 ( TTextPlot,    TBasePlot );
IMPLEMENT_CASTABLE1 ( TAxisPlot,    TBasePlot );
IMPLEMENT_CASTABLE1 ( TLinesPlot,   TBasePlot );
IMPLEMENT_CASTABLE1 ( TRegionPlot,  TBasePlot );
IMPLEMENT_CASTABLE1 ( TFramePlot,   TBasePlot );

IMPLEMENT_CASTABLE1 ( TBoundsPlot,  TAxisPlot );
IMPLEMENT_CASTABLE1 ( TPointsPlot,  TLinesPlot );
IMPLEMENT_CASTABLE1 ( TFuncPlot,    TLinesPlot );
IMPLEMENT_CASTABLE1 ( TArrayPlot,   TLinesPlot );
IMPLEMENT_CASTABLE1 ( TMeshPlot,    TRegionPlot );

/*************** TBasePlot ---------------------------------------- */
 static int PlotsCount = 0;
//-------------------------------------------------------------------
   TBasePlot :: TBasePlot ( TRect& x, TBasePlot* p )
                          : next ( p ), Static ( 1 )
{ pwX1 = x.left;  pwX2 = x.right;   pwLengthX = pwX2 - pwX1;
  pwY1 = x.top;   pwY2 = x.bottom;  pwLengthY = pwY2 - pwY1;
        Color = RGB ( 255, 255, 255 );
        Pen   = new TPen (Color);
 rwScaleX = rwSizeX = rwStepX = rwOrgX = rwEndX = 0.0;
 rwScaleY = rwSizeY = rwStepY = rwOrgY = rwEndY = 0.0;
                          id = ++PlotsCount;
 strcpy (ClassName,"Abstract Base Plot");
};
//-------------------------------------------------------------------
   TBasePlot :: TBasePlot ( )
{                 next = 0;  Static = 1;
  pwX1 = 0;   pwX2 = 100;   pwLengthX = pwX2 - pwX1;
  pwY1 = 0;   pwY2 = 100;   pwLengthY = pwY2 - pwY1;
        Color = RGB ( 255, 255, 255 );
        Pen   = new TPen (Color);
 rwScaleX = rwSizeX = rwStepX = rwOrgX = rwEndX = 0.0;
 rwScaleY = rwSizeY = rwStepY = rwOrgY = rwEndY = 0.0;
                          id = ++PlotsCount;
 strcpy (ClassName,"Abstract Base Plot");
};
//-------------------------------------------------------------------
   TBasePlot ::~TBasePlot ( ) { delete Pen; };
//-------------------------------------------------------------------
void TBasePlot :: ScaleTo ( int Xs, int Ys )
{
      if (Xs > 0) { pwLengthX = Xs; pwX2 = pwX1 + pwLengthX; }
      if (Ys > 0) { pwLengthY = Ys; pwY2 = pwY1 + pwLengthY; }
 rwScaleX = (rwSizeX > 0.0) ? (pwLengthX/rwSizeX) : -1;
  rwStepX = 1.0/rwScaleX;
 rwScaleY = (rwSizeY > 0.0) ? (pwLengthY/rwSizeY) : -1;
  rwStepY = 1.0/rwScaleY;
};
//-------------------------------------------------------------------
void TBasePlot :: Adjust ( int Adj )
{
     switch ( Adj )
```

```
 { case 0:
   {  rwSizeX = rwEndX - rwOrgX;     rwSizeY = rwEndY - rwOrgY;
 pwLengthX = pwX2-pwX1; if (pwLengthX <= 0) { pwLengthX=pwX2=100; pwX1=0; }
 pwLengthY = pwY2-pwY1; if (pwLengthY <= 0) { pwLengthY=pwY2=100; pwY1=0; }
    if (rwSizeX <= 0.0) { rwSizeX=1.0; rwOrgX=0; rwEndX = rwOrgX+rwSizeX; }
    if (rwSizeY <= 0.0) { rwSizeY=1.0; rwOrgY=0; rwEndY = rwOrgY+rwSizeY; }
 rwScaleX = pwLengthX/rwSizeX;  rwScaleY = pwLengthY/rwSizeY;
 rwStepX  = 1.0/rwScaleX;       rwStepY  = 1.0/rwScaleY;
 rwScaleX = (rwSizeX > 0.0) ? (pwLengthX/rwSizeX) : -1;
  rwStepX = 1.0/rwScaleX;
 rwScaleY = (rwSizeY > 0.0) ? (pwLengthY/rwSizeY) : -1;
  rwStepY = 1.0/rwScaleY;                          break;
    }
  default:
   {  rwSizeX = rwEndX - rwOrgX;     rwSizeY = rwEndY - rwOrgY;
 pwLengthX = pwX2-pwX1; if (pwLengthX <= 0) { pwLengthX=pwX2=100; pwX1=0; }
 pwLengthY = pwY2-pwY1; if (pwLengthY <= 0) { pwLengthY=pwY2=100; pwY1=0; }
    if (rwSizeX <= 0.0) { rwSizeX=1.0; rwOrgX=0; rwEndX = rwOrgX+rwSizeX; }
    if (rwSizeY <= 0.0) { rwSizeY=1.0; rwOrgY=0; rwEndY = rwOrgY+rwSizeY; }
 rwScaleX = pwLengthX/rwSizeX;  rwScaleY = pwLengthY/rwSizeY;
 rwStepX  = 1.0/rwScaleX;       rwStepY  = 1.0/rwScaleY;
 rwScaleX = (rwSizeX > 0.0) ? (pwLengthX/rwSizeX) : -1;
  rwStepX = 1.0/rwScaleX;
 rwScaleY = (rwSizeY > 0.0) ? (pwLengthY/rwSizeY) : -1;
  rwStepY = 1.0/rwScaleY;
    }
  }
};
//---------------------------------------------------------------------
void TBasePlot :: IdClassMessage ( )
{
 char txt [255],
     buf [ 80],
   captio[ 20];
            strcpy (txt, "  Client work area:\n");
 sprintf (buf," X1 = %d, X2 = %d,  Length X = %d\n", pwX1, pwX2, pwLengthX);
                  strcat (txt, buf);
 sprintf (buf," Y1 = %d, Y1 = %d,  Length Y = %d\n", pwY1, pwY2, pwLengthY);
                  strcat (txt, buf);
                  strcat (txt,"  Phys. Area: \n");
 sprintf (buf," x1 = %.2f, x2 = %.2f, Size X = %.2f, 1/hx = %.2f, hx = %.2f\n",
               rwOrgX, rwEndX, rwSizeX, rwScaleX, rwStepX);
                  strcat (txt, buf);
 sprintf (buf," y1 = %.2f, y2 = %.2f, Size Y = %.2f, 1/hy = %.2f, hy = %.2f\n",
               rwOrgY, rwEndY, rwSizeY, rwScaleY, rwStepY);
                  strcat (txt, buf);
                  strcat (txt,"     Private info: \n");
  PrivateInfo ( buf );     strcat (txt, buf);
        itoa ( id, buf, 10 ); strcpy (captio, buf); strcat (captio, " : ");
                  strcat (captio, ClassName);
   ::MessageBox (0, txt, captio, MB_ICONINFORMATION | MB_OK);
};

/*************** TTextPlot ---------------------------------------- */
```

```
//---------------------------------------------------------------------------
    TTextPlot :: TTextPlot ( char* text, TRect& x, TBasePlot* p )
            : TBasePlot ( x, p )
{
 strcpy (Text, text);   Htxt = 16;   Ltxt = strlen (Text);
    pwX2 = pwX1 + 8*Ltxt;  pwLengthX = pwX2 - pwX1;
    pwY2 = pwY1 + Htxt;    pwLengthY = pwY2 - pwY1;
     strcpy ( ClassName, typeid (this).name() );
}
//---------------------------------------------------------------------------
    TTextPlot :: TTextPlot ( TWindow* parent, TBasePlot* p )
            : TBasePlot ( )
{
 strcpy (Text, "  text  ");   Htxt = 16;   Ltxt = strlen (Text);
           TTextPlotStr tds;
 if ( TTextPlotDlg (parent, tds).Execute () == IDOK )
  {
     strcpy (Text, tds.svText);
       Htxt = 16;   Ltxt = strlen (Text);
  pwX1 = atoi (tds.svX);  pwX2 = pwX1 + 8*Ltxt;  pwLengthX = pwX2 - pwX1;
  pwY1 = atoi (tds.svY);  pwY2 = pwY1 + Htxt;    pwLengthY = pwY2 - pwY1;
     Color = RGB (atoi(tds.svR), atoi(tds.svG), atoi(tds.svB));
             delete Pen;  Pen    = new TPen (Color);
  }
     strcpy ( ClassName, typeid (this).name() );
}
//---------------------------------------------------------------------------
void TTextPlot :: Draw ( TDC & dc )
{
   dc.SelectObject (*Pen);
   dc.SetTextAlign (TA_CENTER | TA_TOP );
   dc.TextOut (pwX1+pwLengthX/2, pwY1, Text, Ltxt);
   dc.RestorePen ();
};
//---------------------------------------------------------------
void TTextPlot :: Erase ( TDC & dc )
{
};
//---------------------------------------------------------------
void TTextPlot :: PrivateInfo (char* info)
{  char buf [80];
 sprintf (buf," Text = %s, H = %d, L = %d\n", Text, Htxt, Ltxt);
 strcpy (info, buf);
};
/*************** TAxisPlot ---------------------------------------- */
//---------------------------------------------------------------------------
    TAxisPlot :: TAxisPlot ( axis * e, TRect& x, int ori, TBasePlot* p )
            : TBasePlot ( x, p ), Axe ( e )
{
    Orient = (Axe != NULL) ? ori : -1;
           switch ( ori )
  {
 case 0 :                    /*---- horizontal ----*/
      rwOrgX = e->org;   rwOrgY  = 0.0;
```

```
      rwEndX = e->end;   rwEndY  = 1.0;
        Color  = RGB ( 255,  0,  0 );          break;
 case 1 :                  /*---- vertical  ----*/
      rwOrgX =  0.0;     rwOrgY  = e->org;
      rwEndX =  1.0;     rwEndY  = e->end;
        Color  = RGB (  0, 255,  0 );          break;
 case 2 :                  /*---- arbitrary  ----*/
      rwOrgX =  0.0;     rwOrgY  = e->org;
      rwEndX =  1.0;     rwEndY  = e->end;
        Color  = RGB (  0,  0, 255 );          break;
 default:
      rwOrgX =  0.0;     rwOrgY  =  0.0;
      rwEndX =  1.0;     rwEndY  =  1.0;
        Color  = RGB ( 127, 127, 127 );
    ::MessageBox (0,"Axis NULL pointer","Error",MB_OK);
  }
      pwX1   = x.left+2;   pwX2    = x.right-2;
      pwY1   = x.bottom-2; pwY2    = x.bottom+2;
rwSizeX = rwEndX - rwOrgX;     rwSizeY = rwEndY - rwOrgY;
       if (rwSizeX <= 0.0) rwSizeX = 1.0;
       if (rwSizeY <= 0.0) rwSizeY = 1.0;
rwScaleX = pwLengthX/rwSizeX;  rwScaleY = pwLengthY/rwSizeY;
rwStepX  = 1.0/rwScaleX;       rwStepY  = 1.0/rwScaleY;
            delete Pen; Pen    = new TPen (Color);
strcpy (cwAname, e->name);       strcpy (cwUnits, e->units);
    pwN = ((e->N > 0)&&(e->N < 151)) ? (e->N) : -1;
         switch ( Orient )
   {
 case 0:  rwSize = rwSizeX;  break;
 case 1:  rwSize = rwSizeY;  break;
 case 2:  rwSize = sqrt(rwSizeY*rwSizeY + rwSizeX*rwSizeX);  break;
 default: rwSize = 1.0;
   }
    rwH = rwSize / pwN;
strcpy (cwFmt, "%.2f");
    strcpy ( ClassName, typeid (this).name() );
 ::MessageBox (0,"ClassName defined","Info",MB_OK);
 IdClassMessage ();
}
//---------------------------------------------------------------------------
   TAxisPlot :: TAxisPlot ( TWindow * parent, TBasePlot* p )
         : TBasePlot ( )
{                          Axe = NULL;
 Orient =  -1;
 rwOrgX =  0.0;     rwOrgY  =  0.0;
 rwEndX =  1.0;     rwEndY  =  1.0;
      Color  = RGB ( 127, 127, 127 );
rwSizeX = rwEndX - rwOrgX;     rwSizeY = rwEndY - rwOrgY;
rwScaleX = pwLengthX/rwSizeX;  rwScaleY = pwLengthY/rwSizeY;
rwStepX  = 1.0/rwScaleX;       rwStepY  = 1.0/rwScaleY;
            delete Pen; Pen    = new TPen (Color);
strcpy (cwAname," e->name ");  strcpy (cwUnits," e->units");
    pwN = 1; rwSize = 1.0;
    rwH = rwSize / pwN;
```

```
strcpy (cwFmt, "%.2f");
 ::MessageBox (0,"Axis NULL pointer","Error",MB_OK);
    strcpy ( ClassName, typeid (this).name() );
 ::MessageBox (0,"ClassName defined","Info",MB_OK);
 IdClassMessage ();
}
//---------------------------------------------------------------------
void TAxisPlot :: Draw ( TDC & dc )
{ num x = rwOrgX;
 num y = rwOrgY;
    dc.SelectObject (*Pen);
                dc.MoveTo ( pwX1,  pwY1 );
                dc.LineTo ( pwX2,  pwY2 );
 for ( int i=1;  i < pwN;  i++ )
   {
      dc.MoveTo ( Icx ( x ), Icy ( y ) );
//    dc.LineTo ( Icx ( x ), Icy ( y ) );
              x+=rwSizeX/pwN;
   }
  dc.RestorePen ();
};
//-------------------------------------------------------------
void TAxisPlot :: Erase ( TDC & dc )
                        { Draw ( dc ); };
//-------------------------------------------------------------
void TAxisPlot :: ScaleTo ( int Xs, int Ys )
{
   TBasePlot :: ScaleTo ( Xs, Ys );
        switch ( Orient )
   {
 case 0:  rwSize = rwSizeX;  break;
 case 1:  rwSize = rwSizeY;  break;
 case 2:  rwSize = sqrt(rwSizeY*rwSizeY + rwSizeX*rwSizeX);  break;
 default: rwSize = 1.0;
   }
};
//-------------------------------------------------------------
void TAxisPlot :: PrivateInfo (char* info)
{  char buf [80];
 sprintf (buf," Name = %s, N = %d, h = %.3f, Fmt = %s, Orient = %d \n",
              cwAname, pwN, rwH, cwFmt, Orient);
 strcpy (info, buf);
// if (Axe) strcpy (buf, typeid (Axe).name());
//    else  strcpy (buf, "NULL pointer");
// strcat (info,"Axe: "); strcat (info, buf);
};

/*************** TBoundsPlot ---------------------------------------- */
//---------------------------------------------------------------------
   TBoundsPlot :: TBoundsPlot ( axis * e, TRect& x, int ori, TBasePlot* p )
          : TAxisPlot ( e, x, ori, p )
{     strcpy ( ClassName, typeid (this).name() );
}
//---------------------------------------------------------------------
```

```
void TBoundsPlot :: Draw ( TDC & dc )
{
                dc.SelectObject (*Pen);
                dc.RestorePen ();
};
//---------------------------------------------------------------
void TBoundsPlot :: Erase ( TDC & dc )
                        { Draw ( dc ); };
//---------------------------------------------------------------
void TBoundsPlot :: PrivateInfo (char* info)
{  char buf [80];
   TAxisPlot :: PrivateInfo ( info );
 sprintf (buf," BounaryType:  %d\n", (int)Bnd);
 strcat (info, buf);
};

/*************** TLinesPlot ---------------------------------------- */
//---------------------------------------------------------------------
   TLinesPlot :: TLinesPlot ( TAbstrPhysVal * e, TRect& x, TBasePlot* p )
          : TBasePlot ( x, p ), PhVal ( e )
{ if ( e )
      { rwOrgX = e->Org;  rwOrgY = e->Min;
        rwEndX = e->End;  rwEndY = e->Max;
      }
  else
      { rwOrgX =  0.0;    rwOrgY =  0.0;
        rwEndX = 10.0;    rwEndY = 10.0;
   ::MessageBox (0,"PhysVal NULL pointer","Error",MB_OK);
      }
 rwSizeX = rwEndX - rwOrgX;     rwSizeY = rwEndY - rwOrgY;
        if (rwSizeX <= 0.0) rwSizeX = 1.0;
        if (rwSizeY <= 0.0) rwSizeY = 1.0;
 rwScaleX = pwLengthX/rwSizeX;  rwScaleY = pwLengthY/rwSizeY;
 rwStepX = 1.0/rwScaleX;        rwStepY = 1.0/rwScaleY;
         Color  = RGB ( 255, 255, 255 );
         Pen    = new TPen (Color);
 Static = 0;
     strcpy ( ClassName, typeid (this).name() );
}
//---------------------------------------------------------------------
   TLinesPlot :: TLinesPlot ( TWindow * parent, TBasePlot* p )
          : TBasePlot ( ), PhVal ( 0 )
{
     strcpy ( ClassName, typeid (this).name() );
}
//---------------------------------------------------------------------
void TLinesPlot :: Draw ( TDC & dc )
{
 char txt [] = "1D ";
 double d, w, a, h;
 num x, y;
    h = rwSizeX / pwLengthX;
    a = rwSizeY * 0.25;
    w = 6.28 * 8 /rwSizeX;
```

```
                    dc.MoveTo ( pwX1, Icy (0.0) );
 dc.SelectObject (*Pen);
   for ( int i=1; i < pwLengthX; i++)
  {                      a *=  0.9;
                         x = num (rwOrgX + i * h);
                         y = num (a * sin (w * x));
 dc.LineTo ( Icx ( x ), Icy ( y ) );
  }
 dc.RestorePen ();
};
//-------------------------------------------------------------
void TLinesPlot :: Erase ( TDC & dc )
{
 char txt [] = "1D ";
 double d, w, a, h;
 num x, y;
    h = rwSizeX / pwLengthX;
    a = rwSizeY * 0.25;
    w = 6.28 * 8 /rwSizeX;
                    dc.MoveTo ( pwX1, Icy (0.0) );
 dc.SelectObject (*Pen);
   for ( int i=1; i < pwLengthX; i++)
  {                      a *=  0.9;
                         x = num (rwOrgX + i * h);
                         y = num (a * sin (w * x));
 dc.LineTo ( Icx ( x ), Icy ( y ) );
  }
 dc.RestorePen ();
};
//-------------------------------------------------------------
void TLinesPlot :: PrivateInfo (char* info)
{  char buf [80];
 sprintf (buf," N = %d\n", pwN);
 strcpy  (info, buf);
 strcpy  (buf, typeid (PhVal).name());
 strcat  (info,"PhysVal: "); strcat (info, buf);
};

/*************** TPointsPlot ---------------------------------------- */
//-------------------------------------------------------------------
void TPointsPlot :: Draw ( TDC & dc )
{ num x = rwOrgX;
  num y = rwOrgY;
     dc.SelectObject (*Pen);
                    dc.MoveTo ( pwX1,  pwY1 );
 for ( int i=1;  i < pwN;  i++ )
    {
        dc.MoveTo ( Icx ( x ), Icy ( y ) );
//      dc.LineTo ( Icx ( x ), Icy ( y ) );
                    x+=rwSizeX;  //pwN;
    }
  dc.RestorePen ();
};
//-------------------------------------------------------------
```

```
void TPointsPlot :: Erase ( TDC & dc )
                        { Draw ( dc ); };

/**************** TFuncPlot ---------------------------------------- */
//-------------------------------------------------------------------
   TFuncPlot :: TFuncPlot ( TFunctionVal* e, TRect& x, TBasePlot* p )
         : TLinesPlot ( e, x, p )
{
    strcpy ( ClassName, typeid (this).name() );
};
//-------------------------------------------------------------------
   TFuncPlot :: TFuncPlot ( TWindow * parent, TBasePlot* p )
         : TLinesPlot ( parent, p )
{
    strcpy ( ClassName, typeid (this).name() );
};
//-------------------------------------------------------------------
void TFuncPlot :: Draw ( TDC & dc )
{ num x = rwOrgX;
 num y = rwOrgY;
    dc.SelectObject (*Pen);
                dc.MoveTo ( pwX1,  pwY1 );
for ( int i=1;  i < pwN;  i++ )
   {
      dc.MoveTo ( Icx ( x ), Icy ( y ) );
//    dc.LineTo ( Icx ( x ), Icy ( y ) );
                x += rwSizeX;   //pwN;
   }
 dc.RestorePen ();
};
//-------------------------------------------------------------
void TFuncPlot :: Erase ( TDC & dc )
                        { Draw ( dc ); };

/**************** TArrayPlot ---------------------------------------- */
//-------------------------------------------------------------------
   TArrayPlot :: TArrayPlot ( TGridVal* e, TRect& x, TBasePlot* p )
         : TLinesPlot ( e, x, p )
{
//  array A ( e->data, e->Mesh->N1, e->Mesh->N2 );
//  array A = matrix X ( e->data, e->Mesh->N1, e->Mesh->N2 ).row (k);
    strcpy ( ClassName, typeid (this).name() );
};
//-------------------------------------------------------------------
   TArrayPlot :: TArrayPlot ( TWindow * parent, TBasePlot* p )
         : TLinesPlot ( parent, p )
{
    strcpy ( ClassName, typeid (this).name() );
}
//-------------------------------------------------------------------
void TArrayPlot :: Draw ( TDC & dc )
{ num x=rwOrgX;
 num y=rwOrgY;
    dc.SelectObject (*Pen);
```

```
                dc.MoveTo ( pwX1,  pwY1 );
 for ( int i=1;  i < pwN;  i++ )
    {
        dc.MoveTo ( Icx ( x ), Icy ( y ) );
//      dc.LineTo ( Icx ( x ), Icy ( y ) );
                x+=rwSizeX;  //pwN;
    }
  dc.RestorePen ();
};
//---------------------------------------------------------------
void TArrayPlot :: Erase ( TDC & dc )
                        { Draw ( dc ); };

/*************** TFramePlot ---------------------------------------- */
//-------------------------------------------------------------------
   TFramePlot :: TFramePlot ( TFrameStruct* e, TRect& x, TBasePlot* p )
         : TBasePlot ( x, p )
{ char * erp;
   FramClr = RGB (  0,  0, 255 );
   AxisClr = RGB ( 255,  0,  0 );
   MeshClr = RGB ( 127, 255,  0 );
   TextClr = RGB (  0, 127, 127 );
        FramPen  = new TPen (FramClr,  1, PS_DASH);
        AxisPen  = new TPen (AxisClr,  2, PS_SOLID);
        MeshPen  = new TPen (MeshClr,  1, PS_DASHDOT);
   pwMeshX = atoi (e->X.svM);       pwMeshY = atoi (e->Y.svM);
   pwNx    = atoi (e->X.svN);       pwNy    = atoi (e->Y.svN);
 strcpy (cwXname,  e->X.svName);   strcpy (cwYname,  e->Y.svName);
 strcpy (cwXunits, e->X.svUnits);  strcpy (cwYunits, e->Y.svUnits);
 strcpy (cwXfmt,   e->X.svFmt);    strcpy (cwYfmt,   e->Y.svFmt);
         strcpy (cwName,   e->svName);
   rwOrgX  = (num) strtod (e->X.svOrg,  &erp );
   rwEndX  = (num) strtod (e->X.svEnd,  &erp );
   rwOrgY  = (num) strtod (e->Y.svOrg,  &erp );
   rwEndY  = (num) strtod (e->Y.svEnd,  &erp );
 rwSizeX = rwEndX - rwOrgX;    rwSizeY = rwEndY - rwOrgY;
       if (rwSizeX <= 0.0) rwSizeX = 1.0;
       if (rwSizeY <= 0.0) rwSizeY = 1.0;
 rwScaleX = pwLengthX/rwSizeX;  rwScaleY = pwLengthY/rwSizeY;
 rwStepX  = 1.0/rwScaleX;      rwStepY  = 1.0/rwScaleY;
 rwHx    = rwSizeX/pwMeshX;    rwHy    = rwSizeY/pwMeshY;
    strcpy ( ClassName, typeid (this).name() );
}
//-------------------------------------------------------------------
   TFramePlot :: TFramePlot ( TWindow* parent, TBasePlot* p )
         : TBasePlot ( )
{
char * erp;
   FramClr = RGB (  0,  0, 255 );
   AxisClr = RGB ( 255,  0,  0 );
   MeshClr = RGB ( 127, 255,  0 );
   TextClr = RGB (  0, 127, 127 );
        FramPen  = new TPen (FramClr,  1, PS_DASH);
        AxisPen  = new TPen (AxisClr,  2, PS_SOLID);
```

```
        MeshPen  = new TPen (MeshClr,  1, PS_DASHDOT);
    strcpy ( ClassName, typeid (this).name() );
}
//---------------------------------------------------------------
   TFramePlot :: ~TFramePlot ( )
{                          TBasePlot ::~TBasePlot ();
 delete  FramPen;  delete  MeshPen;  delete  AxisPen;
};
//---------------------------------------------------------------
void TFramePlot :: Draw ( TDC & dc )
{
    dc.MoveTo  ( pwX1,         pwY1 );
 dc.SelectObject (*FramPen);
    dc.LineTo  ( pwX1+pwLengthX,  pwY1 );
    dc.LineTo  ( pwX1+pwLengthX,  pwY1+pwLengthY );
    dc.LineTo  ( pwX1,         pwY1+pwLengthY );
    dc.LineTo  ( pwX1,         pwY1 );
 dc.RestorePen ();
};
//---------------------------------------------------------------
void TFramePlot :: Erase ( TDC & dc )
                          { Draw ( dc ); };
//---------------------------------------------------------------
void TFramePlot :: PrivateInfo (char* info)
{  char buf [80];
 sprintf (buf," Name = %s\n", cwName);
 strcpy  (info, buf);
// strcpy  (buf, typeid (PhVal).name());
// strcat  (info,"PhysVal: "); strcat (info, buf);
};

/*************** TRegionPlot ---------------------------------------- */
//---------------------------------------------------------------------
   TRegionPlot :: TRegionPlot ( TAbstrRegion* e, TRect& x, TBasePlot* p )
          : TBasePlot ( x, p ), Rgn ( e )
{                          Rgn = e;
 TGeomAxis * hrz = Rgn -> X1 ();
 TGeomAxis * vrt = Rgn -> X2 ();
  BndOpenClr  = RGB (  0,  0, 255 );
  BndMetalClr = RGB ( 255,  0,  0 );
  BndSymClr   = RGB ( 127, 255,  0 );
  BndPeriodClr = RGB (  0, 127, 127 );
        BndOpenPen  = new TPen (BndOpenClr,  1, PS_DASH);
        BndMetalPen = new TPen (BndMetalClr, 2, PS_SOLID);
        BndSymPen   = new TPen (BndSymClr,   1, PS_DASHDOT);
        BndPeriodPen = new TPen (BndPeriodClr, 1, PS_SOLID);
  if ((hrz) && (vrt))
 {
rwOrgX = hrz->org;          rwOrgY = vrt->org;
rwEndX = hrz->end;          rwEndY = vrt->end;
  strcpy (cwXname, hrz->name);   strcpy (cwYname, vrt->name);
  strcpy (cwXunits, hrz->units);  strcpy (cwYunits, vrt->units);
    pwNx = ((hrz->N > 0)&&(hrz->N < 51))? hrz->N : -1;
    pwNy = ((vrt->N > 0)&&(vrt->N < 51))? vrt->N : -1;
```

```
pwMeshX    = hrz->N;                pwMeshY    = vrt->N;
pwLeftBnd  = TBoundaryType(hrz->bound1);  pwRightBnd = TBoundaryType (hrz->bound2);
pwBottomBnd = TBoundaryType(vrt->bound1);  pwTopBnd  = TBoundaryType (vrt->bound2);
  }
    else
  {
 ::MessageBox (0,"Horiz Or Vert Axis NULL pointer","Error",MB_OK);
rwOrgX =  0.0;          rwOrgY =  0.0;
rwEndX = 10.0;          rwEndY = 10.0;
  strcpy (cwXname, "Îñü X");    strcpy (cwYname, "Îñü Y");
  strcpy (cwXunits,"cm");      strcpy (cwYunits,"m");
    pwNx = -1;     pwNy = -1;
pwMeshX =  20;          pwMeshY =  10;
pwLeftBnd =  1;  pwRightBnd = 3;
pwBottomBnd = 2;  pwTopBnd =  4;
  }
    pwNx = ((pwMeshX > 0)&&(pwMeshX < 151))? pwMeshX : 10;
    pwNy = ((pwMeshY > 0)&&(pwMeshY < 151))? pwMeshY : 10;
 rwHx = rwSizeX / pwNx;       rwHy = rwSizeY / pwNy;
    strcpy ( ClassName, typeid (this).name() );
 Adjust ();
}
//------------------------------------------------------------------------
   TRegionPlot :: TRegionPlot ( TWindow* parent, TBasePlot* p )
          : TBasePlot ( )
{
  strcpy ( ClassName, typeid (this).name() );
 Adjust ();
}
//-------------------------------------------------------------------
   TRegionPlot :: ~TRegionPlot ( )
{                        TBasePlot ::~TBasePlot ();
 delete  BndOpenPen;  delete  BndMetalPen;
 delete  BndSymPen;   delete  BndPeriodPen;
};
//-------------------------------------------------------------------
long TRegionPlot :: BndColor ( TBoundaryType ind )
{            switch (ind)
  { case ConductingBnd:    return (BndMetalClr);
    case OpenBnd:          return (BndOpenClr);
    case AxisymmetricBnd:  return (BndSymClr);
    case PeriodicBnd:      return (BndPeriodClr);
     default:              return (Color);
  }
};
//-------------------------------------------------------------------
TPen* TRegionPlot :: BndPen ( TBoundaryType ind )
{            switch (ind)
  { case ConductingBnd:    return (BndMetalPen);
    case OpenBnd:          return (BndOpenPen);
    case AxisymmetricBnd:  return (BndSymPen);
    case PeriodicBnd:      return (BndPeriodPen);
     default:              return (Pen);
  }
```

```
};
//-------------------------------------------------------------------------
void TRegionPlot :: Draw ( TDC & dc )
{
                        dc.MoveTo ( pwX1, pwY1 );
 dc.SelectObject (*BndPen (pwTopBnd));
    dc.LineTo ( pwX1+pwLengthX, pwY1 );          dc.RestorePen ();
 dc.SelectObject (*BndPen (pwRightBnd));
    dc.LineTo ( pwX1+pwLengthX, pwY1+pwLengthY );  dc.RestorePen ();
 dc.SelectObject (*BndPen (pwBottomBnd));
    dc.LineTo ( pwX1, pwY1+pwLengthY );          dc.RestorePen ();
 dc.SelectObject (*BndPen (pwLeftBnd));
    dc.LineTo ( pwX1, pwY1 );                  dc.RestorePen ();
};
//-------------------------------------------------------------------
void TRegionPlot :: Erase ( TDC & dc )
                              { Draw ( dc ); };
//-------------------------------------------------------------------
void TRegionPlot :: PrivateInfo (char* info)
{  char buf [80];
 sprintf (buf," Name = %s\n", cwName);
 strcpy  (info, buf);
// strcpy  (buf, typeid (Rgn).name());
// strcat  (info,"Rgn: "); strcat (info, buf);
};

/**************** TMeshPlot ---------------------------------------- */
//-------------------------------------------------------------------------
   TMeshPlot :: TMeshPlot ( TAbstrMesh* e, TRect& x, TBasePlot* p )
         : TRegionPlot ( (e->Rgn), x, p ), Mesh ( e )
{
    strcpy ( ClassName, typeid (this).name() );
}
//-------------------------------------------------------------------------
   TMeshPlot :: TMeshPlot ( TWindow* parent, TBasePlot* p )
         : TRegionPlot ( parent, p )
{
    strcpy ( ClassName, typeid (this).name() );
}
//-------------------------------------------------------------------------
void TMeshPlot :: Draw ( TDC & dc )
{  int i;
                        dc.SelectObject (*Pen);
  for ( i=1; i < pwNy; i++)
 {      dc.MoveTo ( pwX1,          Icy (rwOrgY + i*rwHy) );
        dc.LineTo ( pwX1+pwLengthX, Icy (rwOrgY + i*rwHy) );
 }
  for ( i=1; i < pwNx; i++)
 {      dc.MoveTo ( Icx (rwOrgX + i*rwHx), pwY1 );
        dc.LineTo ( Icx (rwOrgX + i*rwHx), pwY1+pwLengthY );
 }
                        dc.RestorePen ();
};
//-------------------------------------------------------------------
```

```
void TMeshPlot :: Erase ( TDC & dc )
                              { Draw ( dc ); };
//-------------------------------------------------------------------
void TMeshPlot :: PrivateInfo (char* info)
{  char buf [80];
    TRegionPlot :: PrivateInfo ( info );
 sprintf (buf," Name = %s\n", cwName);
 strcpy  (info, buf);
// strcpy  (buf, typeid (Rgn).name());
// strcat  (info,"Rgn: "); strcat (info, buf);
};

//_____________________________________________________________________
////////////////////////////////////////////////////////////////
//            TPlotsList
//*****************************************************************
    TPlotsList :: TPlotsList ( TBasePlot * frst )
{
            First = frst;    Ne = 1;
 TBasePlot * p = First;
   while ( p->next )
    {    p = p->next;
                 ++Ne;
    }
};
//-------------------------------------------------------------------
    TPlotsList :: ~TPlotsList ( )
{
 TBasePlot *c, * p = First;
   while ( p )
    {    c = p;  p = p->next;
       delete c;             --Ne;
    }
};
//-------------------------------------------------------------------
 void TPlotsList :: Ins ( TBasePlot * Plot )
{
 if ( Ne == 0 )  First = Plot;
  else
    {  TBasePlot * p = First;
  while ( p->next ) p = p->next;
                p->next = Plot;
    }
 Plot->next = 0;    ++Ne;
};
//-------------------------------------------------------------------
 void TPlotsList :: Del ( TBasePlot * Plot )
{
 if ( Ne == 0 )  return;
  TBasePlot *c, * p = First;
   if (Plot != First)
     {       while ( p )
       { if ((p->next) == Plot)
            {  c = p->next;
```

```
                p->next = c->next;
          delete c;  --Ne;  break;
              }
         p = p->next;
        }
      }
    else
      {    c = First->next;
        delete First;
            First = c;       --Ne;
      }
};
//-----------------------------------------------------------------
 void TPlotsList :: Del ( int k )
{
 if ( Ne == 0 )  return;
  TBasePlot *c, * p = First;    int j = 1;
   if ( k != 1 )
     {       while ( p )
       { if ( k == ++j )
            {  c = p->next;
                p->next = c->next;
          delete c;  --Ne;  break;
              }
         p = p->next;
        }
      }
    else
      {    c = First->next;
        delete First;
            First = c;       --Ne;
      }
};
//-----------------------------------------------------------------
 void TPlotsList :: ScaleTo ( int Xs, int Ys )
{
 if ( Ne == 0 )  return;
       TBasePlot * plot = First;
    while ( plot )
  {   plot -> ScaleTo ( Xs, Ys );
      plot = (plot->next);
  }
};
//-----------------------------------------------------------------
 void TPlotsList :: Draw ( TDC & dc, BOOL NonStaticOnly )
{
 if ( Ne == 0 )  return;
       TBasePlot * plot = First;
    while ( plot )
  {
 if (!(plot -> Static) || (!NonStaticOnly))
      plot -> Draw ( dc );
      plot = (plot->next);
  }
```

```
};
//-----------------------------------------------------------------
 void TPlotsList :: Redraw ( TDC & dc, BOOL NonStaticOnly )
{
 if ( Ne == 0 )  return;
       TBasePlot * plot = First;
    while ( plot )
   {
  if (!(plot -> Static) || (!NonStaticOnly))
     {  plot -> Erase ( dc );
        plot -> Draw  ( dc );
     }
        plot = (plot->next);
   }
};
//-----------------------------------------------------------------
 void TPlotsList :: Revision (BOOL NonStaticOnly )
{ char txt [80];
     if (NonStaticOnly) strcpy (txt," NonStatic ");
         else         strcpy (txt,"  Certain ");
             strcat (txt, " Plots List ");
 if ( Ne == 0 )
          {   strcat (txt, "is Empty !");
  ::MessageBox (0, txt, "Revision", MB_ICONINFORMATION | MB_OK);
                return;
          }
        sprintf (txt," %s \n Ne = %d ", txt, Ne);
  ::MessageBox (0, txt, "Revision", MB_ICONINFORMATION | MB_OK);
       TBasePlot * plot = First;
    while ( plot )
   {
  if (!(plot -> Static) || (!NonStaticOnly))
      plot -> IdClassMessage ( );
      plot = (plot->next);
   }
};
//-----------------------------------------------------------------
  int TPlotsList :: Element ( char* item, int k )
{
    int Res = 0;
 if ( Ne == 0 )  return Res;
   TBasePlot * p = First;      int j = 0;
          while ( p )
    {  if ( k == ++j )
        {  sprintf (item,"%d : %s",  p->id, p->ClassName);
           Res = j;                 break;
        }
        p = p->next;
    }
 return Res;
};
//-----------------------------------------------------------------
 void TPlotsList :: ForEach ( void (*Procedure) (TBasePlot * ) )
{
```

```
 if ( Ne == 0 )  return;
   TBasePlot * plot = First;
      while ( plot )
   {                  Procedure ( plot );
      plot = (plot->next);
   }
};
////////////////////////////////////////////////////////////////
//---------------------------------------------------------------
   TAxisStruct :: TAxisStruct ()
{ int rx = 10;
              svName  [0]  = 0;
     sprintf (  svOrg,   "%g", 0.0 );
     sprintf (  svEnd,   "%g", 1.0 );
 itoa ( 10,      svN,     rx);
 itoa ( 10,      svM,     rx);
      strcpy (  svFmt,    "%.2f" );
     sprintf (  svH,      "%g", 0.1 );
              svUnits [0]  = 0;
 itoa ( 10,      svX1,    rx);
 itoa ( 10,      svY1,    rx);
 itoa ( 90,      svX2,    rx);
 itoa ( 90,      svY2,    rx);
};
//---------------------------------------------------------------
   TAxisStruct :: TAxisStruct ( axis & A )
{ int rx = 10;
      strcpy (  svName,           A.name );
     sprintf (  svOrg,   "%g",     A.org );
     sprintf (  svEnd,   "%g",     A.end );
    itoa ( A.N,  svN,     rx);
    itoa ( A.N,  svM,     rx);
      strcpy (  svFmt,    "%.2f" );
     sprintf (  svH,      "%g",    A.step );
      strcpy (  svUnits,          A.units );
 itoa ( 10,      svX1,    rx);
 itoa ( 10,      svY1,    rx);
 itoa ( 90,      svX2,    rx);
 itoa ( 90,      svY2,    rx);
};
//---------------------------------------------------------------
 void TAxisStruct :: GetData ( axis & A )
{ char * erp;
 strcpy ( A.name,               svName  );
        A.org  = (num) strtod (  svOrg,  &erp );
        A.end  = (num) strtod (  svEnd,  &erp );
        A.N    =       atoi   (  svN     );
        A.step = (num) strtod (  svH,    &erp );
 strcpy ( A.units,              svUnits );
};

//---------------------------------------------------------------
   TFrameStruct :: TFrameStruct ()
{
```

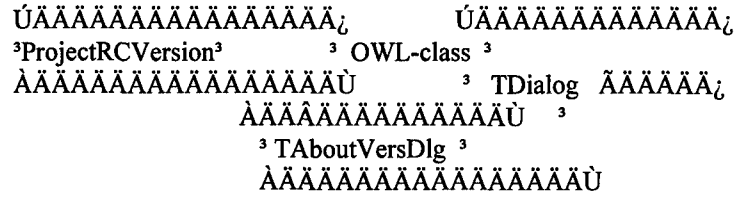
```
 TAxisStruct X ();
 TAxisStruct Y ();
          svName  [0]  = 0;
};
//---------------------------------------------------------------------
  TFrameStruct :: TFrameStruct ( axis & Hrz, axis & Vrt )
{
 TAxisStruct X ( Hrz );
 TAxisStruct Y ( Vrt );
          svName  [0]  = 0;
};
//---------------------------------------------------------------------
 void TFrameStruct :: GetData ( axis & Hrz, axis & Vrt )
{
 X.GetData ( Hrz );
 Y.GetData ( Vrt );
};
//---------------------------------------------------------------------
```

3.DIALOG CLASSES

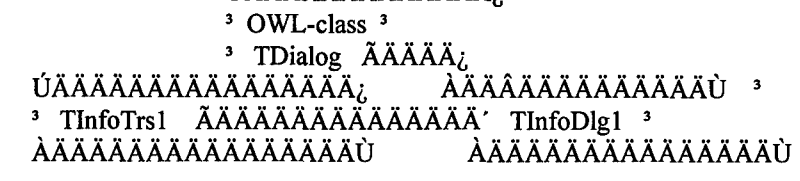
Dialog classes are necessary to create a friendly environment for the user working with the software package. In view of the fact that the project under developments meant for operation under MS WINDOWS we make use of the OWL Library, v.2 compiled by Borland.

THe report contains the child classes from the class TDialog OWL Borland Library. They were developed for input and variation of data required to initialize new objects. In addition to the input dialog windows the library contains windows for systems messages and information windows.

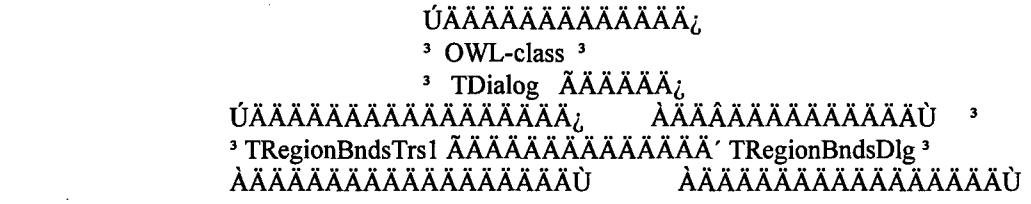
Windows informing of the version and authors of the code.



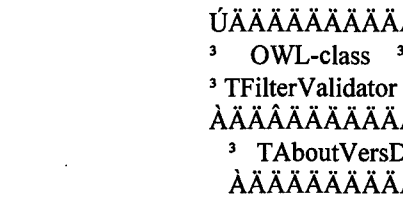
The dialog window (TInfoDig 1) and the data structure (TInfoTrs1) are to output information data on reserves of the computer, on computetional model, etc. through the window of the class TInfoDlg1.



The dialog window (TRegion Dlg) are to input data into the class TAbstr and its child classes.



A service class to check the input information in the dialog window (TRegionBndsDlg).




```
/*****************************************************************/
/*                                                               */
/*              ObjectOriented PIC for C++                       */
/*                 Version 1.00                                  */
/*   Copyright (c) 1993, 1994 The Group Compuetr Simulation      */
/*         High Current Electronics Institute                    */
/*         Contract SPC93-4035(F6170893W0612)                    */
/*                                                               */
/*                        Last write 10-20-94                    */
/*****************************************************************/
#if !defined (_PS_ABOUT_H)
#define     _PS_ABOUT_H

#include <owl\dialog.h>
#include "ps_idres.rh"

class TAboutVersDlg : public TDialog
{
public:
        TAboutVersDlg (TWindow * parent,
                TResId   resId  = iddAboutVer,
                TModule * module = 0);
  virtual ~TAboutVersDlg ();

public:
    void SetupWindow ();
};

#endif
/*** PS_ABOUT.CPP ***/
// #include <_allpch.h>
// #pragma hdrstop

#include <owl\static.h>
```

```
#include <ver.h>

#include "ps_owl.rh"
#include "ps_idres.rh"
#include "ps_about.h"

// Reading the VERSIONINFO resource.
class ProjectRCVersion
{
public:
        ProjectRCVersion (TModule *module);
  virtual ~ProjectRCVersion ();

  BOOL GetProductName    (LPSTR &prodName);
  BOOL GetProductVersion (LPSTR &prodVersion);
  BOOL GetAuthorName     (LPSTR &authorName);
  BOOL GetCopyright      (LPSTR &copyright);
  BOOL GetDebug          (LPSTR &debug);

protected:
  LPBYTE     TransBlock;
  void FAR   *FVData;

private:
  // Don't allow this object to be copied.
  ProjectRCVersion (const ProjectRCVersion &);
  ProjectRCVersion & operator =(const ProjectRCVersion &);
};

ProjectRCVersion::ProjectRCVersion (TModule *module)
{
  char   appFName[255];
  DWORD  fvHandle;
  UINT   vSize;

  FVData = 0;

  module->GetModuleFileName(appFName, sizeof(appFName));
  DWORD dwSize = GetFileVersionInfoSize(appFName, &fvHandle);
  if (dwSize) {
    FVData  = (void FAR *)new char[(UINT)dwSize];
    if (GetFileVersionInfo(appFName, fvHandle, dwSize, FVData))
      if (!VerQueryValue(FVData, "\\VarFileInfo\\Translation", (void FAR* FAR*)&TransBlock,
&vSize)) {
        delete FVData;
        FVData = 0;
      }
  }
}


ProjectRCVersion::~ProjectRCVersion ()
{
  if (FVData)
```

```
    delete FVData;
}


BOOL ProjectRCVersion::GetProductName (LPSTR &prodName)
{
  UINT   vSize;
  char   subBlockName[255];

  wsprintf ( subBlockName,
        "\\StringFileInfo\\%08lx\\%s",
             *(DWORD *)TransBlock,
             (LPSTR)"ProductName");
return FVData ?
   VerQueryValue ( FVData,
            subBlockName,
           (void FAR* FAR*) & prodName,
           &vSize )
 : FALSE;
}


BOOL ProjectRCVersion::GetProductVersion (LPSTR &prodVersion)
{
  UINT   vSize;
  char   subBlockName[255];

  wsprintf ( subBlockName,
       "\\StringFileInfo\\%08lx\\%s",
                *(DWORD *)TransBlock,
                (LPSTR)"ProductVersion");
return FVData ?
  VerQueryValue ( FVData,
           subBlockName,
          (void FAR* FAR*)&prodVersion,
          &vSize )
 : FALSE;
}

BOOL ProjectRCVersion::GetAuthorName (LPSTR &authName)
{
  UINT   vSize;
  char   subBlockName[255];

  wsprintf ( subBlockName,
        "\\StringFileInfo\\%08lx\\%s",
             *(DWORD *)TransBlock,
             (LPSTR)"AuthorName");
return FVData ?
   VerQueryValue ( FVData,
            subBlockName,
           (void FAR* FAR*) & authName,
           &vSize )
 : FALSE;
```

```
}

BOOL ProjectRCVersion::GetCopyright (LPSTR &copyright)
{
  UINT    vSize;
  char    subBlockName[255];

  wsprintf ( subBlockName,
       "\\StringFileInfo\\%08lx\\%s",
             *(DWORD *)TransBlock,
             (LPSTR)"LegalCopyright");
return FVData ?
  VerQueryValue ( FVData,
           subBlockName,
          (void FAR* FAR*) & copyright,
          & vSize)
: FALSE;
}


BOOL ProjectRCVersion::GetDebug (LPSTR &debug)
{
  UINT    vSize;
  char    subBlockName[255];

  wsprintf ( subBlockName,
       "\\StringFileInfo\\%08lx\\%s",
             *(DWORD *)TransBlock,
             (LPSTR)"SpecialBuild");
return FVData ?
   VerQueryValue ( FVData,
            subBlockName,
           (void FAR* FAR*) & debug,
           & vSize)
  : FALSE;
}

//---------------------------------------------------------------------------
TAboutVersDlg :: TAboutVersDlg (TWindow *parent, TResId resId, TModule *module)
        : TDialog (parent, resId, module)
{
  // INSERT>> Your constructor code here.
}


TAboutVersDlg::~TAboutVersDlg ()
{
  Destroy ();

  // INSERT>> Your destructor code here.
}


void TAboutVersDlg::SetupWindow ()
```

```
{
   LPSTR prodName, prodVersion, authName, copyright, debug;

// Get the static text who's value is based on VERSIONINFO.
 TStatic *versionCtrl   = new TStatic (this, IDC_VERSION,   255);
 TStatic *copyrightCtrl = new TStatic (this, IDC_COPYRIGHT, 255);
 TStatic *debugCtrl     = new TStatic (this, IDC_DEBUG,     255);
 TStatic *authorCtrl    = new TStatic (this, IDC_AUTHOR,    255);

   TDialog :: SetupWindow();

// Process the VERSIONINFO.
 ProjectRCVersion psVers ( GetModule() );

// Get the product name, product version, author name and legal copyright strings.
   psVers.GetProductName    (prodName);
   psVers.GetProductVersion (prodVersion);
   psVers.GetAuthorName     (authName);
   psVers.GetCopyright      (copyright);

// IDC_VERSION is the product name and version number, the initial value of IDC_VERSION is
// the word Version (in whatever language) product name VERSION product version.
  char   buffer[255];
  char   versionName[128];
   versionCtrl -> GetText (versionName, sizeof(versionName));
   wsprintf (buffer, "%s %s %s", prodName, versionName, prodVersion);
   versionCtrl -> SetText (buffer);

   authorCtrl   -> SetText (authName);
   copyrightCtrl -> SetText (copyright);

// Only get the SpecialBuild text if the VERSIONINFO resource is there.
   if (psVers.GetDebug (debug))
      debugCtrl -> SetText (debug);
}
                          /*** PS_DLGS1.H ***/
#if !defined (_PS_DLGS1_H)
#define      _PS_DLGS1_H

// #include "ps_axis.h"

class _OWLCLASS TDialog;
class _OWLCLASS TComboBox;
class _OWLCLASS TGroupBox;
class _OWLCLASS TEdit;
class _OWLCLASS TStatic;
class _OWLCLASS TComboBoxData;

#include "ps_regn.h"

#include "ps_dlg01.rh"  // Definition of dialogs resources.
#include "ps_idres.rh"  // Definition of dialogs resources.

//----------------------------------------------------------------------------
```

```
struct TRegionBndsTrs1
{
    TRegionBndsTrs1  ();
    TRegionBndsTrs1  ( const TGeomAxis * x1, const TGeomAxis * x2 );
    TRegionBndsTrs1  ( TRegionBndsTrs1 & src );
 void DefaultFill ( );
 void GetRegionBndsData ( const TGeomAxis * x1, const TGeomAxis * x2 );
 void SetRegionBndsData (     TGeomAxis & x1,     TGeomAxis & x2 );

  BOOL   svXY;
  BOOL   svRZ;
  BOOL   svRTh;
  BOOL   svx1x2;
  char   svx1minprmpt [MXSTAT];
  char   svx1maxprmpt [MXSTAT];
  char   svx2minprmpt [MXSTAT];
  char   svx2maxprmpt [MXSTAT];
  char   svx1min [MXEDIT];
  char   svx1max [MXEDIT];
  char   svx2min [MXEDIT];
  char   svx2max [MXEDIT];
  TComboBoxData  svx1minbnd;
  TComboBoxData  svx1maxbnd;
  TComboBoxData  svx2minbnd;
  TComboBoxData  svx2maxbnd;
  char   svNx1prmpt [MXSTAT];
  char   svNx2prmpt [MXSTAT];
  char   svNx1 [MXINT];
  char   svNx2 [MXINT];
  char   svx1unitsprmpt [MXSTAT];
  char   svx2unitsprmpt [MXSTAT];
  TComboBoxData  svx1units;
  TComboBoxData  svx2units;
};
//---------------------------------------------------------------------------

class TRegionBndsDlg : public TDialog
{
public:
        TRegionBndsDlg (TWindow        * parent,
                        TRegionBndsTrs1 & ds,
                        TRectangleRgn   * rgn    = 0,
                        TResId          resId  = iddBounds1,
                        TModule         * module = 0);
  virtual ~TRegionBndsDlg ();

  virtual void SetupWindow  ();
       void CmRestore  ();
       void CmReset  ();
  virtual void CloseWindow (int retVal = 0);
  virtual void EvSysCommand (UINT id, TPoint& pt);
  virtual void EvDestroy ();
  virtual BOOL CanClose ();
       int errret;
```

```
private:
        void FillBoundsCb (TComboBox * cb, WORD mask, int sel);
    const char* BoundaryStr (TBoundaryType ind);
        void UpdatePrompts ();
        BOOL IsValid (char * errmsg);
        void ChangeGeometry (int gfw);
        void EvCBNx1nKillfocus ();
        void EvCBNx1xKillfocus ();
        void EvCBNx2nKillfocus ();
        void EvCBNx2xKillfocus ();
        void EvBNClicked1 ();
        void EvBNClicked2 ();
        void EvBNClicked3 ();
        void EvBNClicked4 ();
protected:
        TGroupBox *ivGeom;
        TStatic  *ivx1minprmpt;
        TStatic  *ivx1maxprmpt;
        TStatic  *ivx2minprmpt;
        TStatic  *ivx2maxprmpt;
        TEdit    *ivx1min;
        TEdit    *ivx1max;
        TEdit    *ivx2min;
        TEdit    *ivx2max;
        TComboBox *ivx1minbnd;
        TComboBox *ivx1maxbnd;
        TComboBox *ivx2minbnd;
        TComboBox *ivx2maxbnd;
        TStatic  *ivNx1prmpt;
        TStatic  *ivNx2prmpt;
        TEdit    *ivNx1;
        TEdit    *ivNx2;
        TStatic  *ivx1unitsprmpt;
        TStatic  *ivx2unitsprmpt;
        TComboBox *ivx1units;
        TComboBox *ivx2units;
private:
    TRegionBndsTrs1 * rbtData;
    TRegionBndsTrs1 * iniData;
        int       Geom;
    TRectangleRgn   * Region;

DECLARE_RESPONSE_TABLE ( TRegionBndsDlg );
};
//----------------------------------------------------------------------------
#define MXINF1  12
#define MXINF2  25
const int WM_DLG_CLOSED = WM_USER + 200;

struct TInfoTrs1
{
 char svi01 [MXINF1];
 char svi02 [MXINF1];
 char svi03 [MXINF1];
```

```
 char svi04 [MXINF1];
 char svi05 [MXINF1];
 char svi06 [MXINF1];
 char svi07 [MXINF1];
 char svi11 [MXINF2];
 char svi12 [MXINF2];
 char svi13 [MXINF2];
 char svi14 [MXINF2];
 char svi15 [MXINF2];
 char svi16 [MXINF2];
 char svi17 [MXINF2];
};

class TInfoDlg1 : public TDialog
{
public:
        TInfoDlg1 (TWindow   * parent,
                TInfoTrs1 & ds,
                TResId     resId  = iddInfo01);
  void  EvDestroy ();

DECLARE_RESPONSE_TABLE ( TInfoDlg1 );
};

#endif

/*** PS_DLG1.CPP ***/
#if defined (_SPRT_SERVICE_PCH)
//////////////////////////////////////
#pragma hdrfile "d:\picsim\obj\_service.csm"
 #include <owl\dialog.h>
 #include <stdlib.h>
 #include <stdio.h>
 #include <bwcc.h>
 #include <owl\combobox.h>
 #include <owl\edit.h>
 #include <owl\groupbox.h>
 #include <owl\radiobut.h>
 #include <owl\edit.h>
 #include <owl\validate.h>
 #include <owl\applicat.h>
#pragma hdrstop

#else
//////////////////////////////////////
#pragma hdrfile "f:\picsim\obj\_allpch.csm"
 #define _ALLPCH
 #include <_allpch.h>
#pragma hdrstop

#endif
//////////////////////////////////////

#include "ps_dlg1.h"
```

```
#include "ps_str01.rh"

//***************************************************************************
class TNumValidator : public TFilterValidator
{
public:
        TNumValidator (num amin, num amax);
   virtual void Error   ();
   virtual BOOL IsValid  (const char far * str);
//  virtual UINT Transfer (char far * str, void * buf, TTransDirection drctn);
protected:
        num Amin;
        num Amax;
};
//---------------------------------------------------------------------
 TNumValidator :: TNumValidator (num amin, num amax)
          : TFilterValidator ("-+0123456789eE.")
{
 Amin = amin;   Amax = amax;
};
//---------------------------------------------------------------------
 BOOL TNumValidator :: IsValid (const char far * str)
{
  char ** erp;
  num    res  = Amin - 1.0;
 if (strlen(str)==0)  return TRUE;
 if (!TFilterValidator::IsValid (str)) return FALSE;
       res = strtod ( str, erp );
 return (erp == NULL) && (res >= Amin) && (res <= Amax);
};
//---------------------------------------------------------------------
 void TNumValidator :: Error ()
{
  char buff [56];
sprintf (buff, "%s %g to %g","Number Must Be In Diapazon", Amin, Amax);
 MessageBox (0, buff,"Bad Floating Point Number", MB_OK | MB_ICONEXCLAMATION);
};
//***************************************************************************
//---------------------------------------------------------------------
 void TRegionBndsTrs1 :: DefaultFill ()
{
num x = 0.0;
                svXY   = FALSE;
                svRZ   = FALSE;
                svRTh  = FALSE;
                svx1x2 = TRUE;
 strcpy ( svx1minprmpt, "X1 min" );
 strcpy ( svx1maxprmpt, "X1 max" );
 strcpy ( svx2minprmpt, "X2 min" );
 strcpy ( svx2maxprmpt, "X2 max" );
     sprintf ( svx1min, "%g", x );
     sprintf ( svx1max, "%g", x );
     sprintf ( svx2min, "%g", x );
     sprintf ( svx2max, "%g", x );
```

```
 strcpy ( svNx1prmpt, "Nx1" );
 strcpy ( svNx2prmpt, "Nx2" );
  svNx1 [0]      = 0;
  svNx2 [0]      = 0;
 strcpy ( svx1unitsprmpt, "X1 units" );
 strcpy ( svx2unitsprmpt, "X2 units" );

 svx1minbnd.AddStringItem ("Conducting",   ConductingBnd);
 svx1minbnd.AddStringItem ("Open",         OpenBnd);
 svx1minbnd.AddStringItem ("Axisymmetric", AxisymmetricBnd);
 svx1minbnd.AddStringItem ("Periodic",     PeriodicBnd);
 svx1minbnd.AddStringItem ("Dielectric",   DielectricBnd);
                          svx1minbnd.Select (ConductingBnd);
 svx1maxbnd.AddStringItem ("Conducting",   ConductingBnd);
 svx1maxbnd.AddStringItem ("Open",         OpenBnd);
 svx1maxbnd.AddStringItem ("Axisymmetric", AxisymmetricBnd);
 svx1maxbnd.AddStringItem ("Periodic",     PeriodicBnd);
 svx1maxbnd.AddStringItem ("Dielectric",   DielectricBnd);
                          svx1maxbnd.Select (ConductingBnd);
 svx2minbnd.AddStringItem ("Conducting",   ConductingBnd);
 svx2minbnd.AddStringItem ("Open",         OpenBnd);
 svx2minbnd.AddStringItem ("Axisymmetric", AxisymmetricBnd);
 svx2minbnd.AddStringItem ("Periodic",     PeriodicBnd);
 svx2minbnd.AddStringItem ("Dielectric",   DielectricBnd);
                          svx2minbnd.Select (ConductingBnd);
 svx2maxbnd.AddStringItem ("Conducting",   ConductingBnd);
 svx2maxbnd.AddStringItem ("Open",         OpenBnd);
 svx2maxbnd.AddStringItem ("Axisymmetric", AxisymmetricBnd);
 svx2maxbnd.AddStringItem ("Periodic",     PeriodicBnd);
 svx2maxbnd.AddStringItem ("Dielectric",   DielectricBnd);
                          svx2maxbnd.Select (ConductingBnd);
 svx1units.AddString("arbitr");
 svx1units.AddString("m ");
 svx1units.AddString("cm");
 svx1units.AddString("mm");
 svx1units.AddString("inchs");
                          svx1units.Select (2);
 svx2units.AddString("arbitr");
 svx2units.AddString("m ");
 svx2units.AddString("cm");
 svx2units.AddString("mm");
 svx2units.AddString("inchs");
 svx2units.AddString("radians");
 svx2units.AddString("degrees");
                          svx2units.Select (2);
};
//---------------------------------------------------------------------
 TRegionBndsTrs1 :: TRegionBndsTrs1 ()
{
   DefaultFill ();
};
//---------------------------------------------------------------------
 TRegionBndsTrs1 :: TRegionBndsTrs1 (TRegionBndsTrs1 & src)
{
```

```
//    memcpy ( this, *(&src), sizeof TRegionBndsTrs1 );
    DefaultFill ();
            svXY   = src.svXY;
            svRZ   = src.svRZ;
            svRTh  = src.svRTh;
            svx1x2 = src.svx1x2;
 strcpy ( svx1minprmpt, src.svx1minprmpt );
 strcpy ( svx1maxprmpt, src.svx1maxprmpt );
 strcpy ( svx2minprmpt, src.svx2minprmpt );
 strcpy ( svx2maxprmpt, src.svx2maxprmpt );
 strcpy ( svx1min, src.svx1min );
 strcpy ( svx1max, src.svx1max );
 strcpy ( svx2min, src.svx2min );
 strcpy ( svx2max, src.svx2max );
 strcpy ( svNx1prmpt, src.svNx1prmpt );
 strcpy ( svNx2prmpt, src.svNx2prmpt );
 strcpy ( svNx1, src.svNx1 );
 strcpy ( svNx2, src.svNx2 );
 strcpy ( svx1unitsprmpt, src.svx1unitsprmpt );
 strcpy ( svx2unitsprmpt, src.svx2unitsprmpt );
       svx1minbnd.Select (src.svx1minbnd.GetSelIndex());
       svx1maxbnd.Select (src.svx1maxbnd.GetSelIndex());
       svx2minbnd.Select (src.svx2minbnd.GetSelIndex());
       svx2maxbnd.Select (src.svx2maxbnd.GetSelIndex());
       svx1units.Select (src.svx1units.GetSelIndex());
       svx2units.Select (src.svx2units.GetSelIndex());
};
//---------------------------------------------------------------------
 TRegionBndsTrs1 :: TRegionBndsTrs1 (const TGeomAxis * x1,
                         const TGeomAxis * x2)
{
          GetRegionBndsData ( x1,  x2);
};
//---------------------------------------------------------------------
 void TRegionBndsTrs1 :: GetRegionBndsData (const TGeomAxis * x1,
                              const TGeomAxis * x2)
{
 int rx = 10;
          DefaultFill ();
    if ((x1)&&(x2))
 {
 wsprintf ( svx1minprmpt, "%s min", (x1->name) );
 wsprintf ( svx1maxprmpt, "%s max", (x1->name) );
 wsprintf ( svx2minprmpt, "%s min", (x2->name) );
 wsprintf ( svx2maxprmpt, "%s max", (x2->name) );
     sprintf ( svx1min, "%g", (x1->org) );
     sprintf ( svx1max, "%g", (x1->end) );
     sprintf ( svx2min, "%g", (x2->org) );
     sprintf ( svx2max, "%g", (x2->end) );
 wsprintf ( svNx1prmpt, "N %s", (x1->name) );
 wsprintf ( svNx2prmpt, "N %s", (x2->name) );
     itoa ((x1->N), svNx1, rx);
     itoa ((x2->N), svNx2, rx);
  svXY = svRZ = svRTh = svx1x2 = FALSE;
```

```
    if ((x1->id == 0)&&(x2->id == 1)) svXY   = TRUE;
    if ((x1->id == 2)&&(x2->id == 3)) svRZ   = TRUE;
    if ((x1->id == 2)&&(x2->id == 4)) svRTh  = TRUE;
    if ((x1->id == 5)&&(x2->id == 6)) svx1x2 = TRUE;
        svx1minbnd.Select (x1->bound1);
        svx1maxbnd.Select (x1->bound2);
        svx2minbnd.Select (x2->bound1);
        svx2maxbnd.Select (x2->bound2);
   if (x1->uid > 0) svx1units.Select (x1->uid);
   if (x2->uid > 0) svx2units.Select (x2->uid);
  };
};
//---------------------------------------------------------------------
 void TRegionBndsTrs1 :: SetRegionBndsData (TGeomAxis & x1, TGeomAxis & x2)
{
  string str;
  char * erp;
  double  res1, res2;
 if (svXY)  { x1.id = 0; x2.id = 1; strcpy (x1.name, "X");  strcpy (x2.name, "Y"); };
 if (svRZ)  { x1.id = 2; x2.id = 3; strcpy (x1.name, "R");  strcpy (x2.name, "Z"); };
 if (svRTh) { x1.id = 2; x2.id = 4; strcpy (x1.name, "R");  strcpy (x2.name, "Th"); };
 if (svx1x2){ x1.id = 5; x2.id = 6; strcpy (x1.name, "X1"); strcpy (x2.name, "X2"); };
  x1.org = (num) strtod (svx1min, &erp );
  x2.org = (num) strtod (svx2min, &erp );
  x1.end = (num) strtod (svx1max, &erp );
  x2.end = (num) strtod (svx2max, &erp );
  x1.N   = atoi ( svNx1 );
  x2.N   = atoi ( svNx2 );
 x1.bound1 = svx1minbnd.GetSelIndex();
 x2.bound1 = svx2minbnd.GetSelIndex();
 x1.bound2 = svx1maxbnd.GetSelIndex();
 x2.bound2 = svx2maxbnd.GetSelIndex();
  str = svx1units.GetSelection ();   strcpy (x1.units, str.c_str());
  str = svx2units.GetSelection ();   strcpy (x2.units, str.c_str());
 x1.uid = svx1units.GetSelIndex();
 x2.uid = svx2units.GetSelIndex();
};
//---------------------------------------------------------------------
//*********************************************************************
//---------------------------------------------------------------------
// int errret;

#define cms_Restore 201
#define cms_Reset   202

DEFINE_RESPONSE_TABLE1 (TRegionBndsDlg, TDialog)
  EV_BN_CLICKED   (idc_XY,        EvBNClicked1),
  EV_BN_CLICKED   (idc_RZ,        EvBNClicked2),
  EV_BN_CLICKED   (idc_RTh,       EvBNClicked3),
  EV_BN_CLICKED   (idc_x1x2,      EvBNClicked4),
  EV_CBN_KILLFOCUS (idc_x1min_bnd,  EvCBNx1nKillfocus),
  EV_CBN_KILLFOCUS (idc_x1max_bnd,  EvCBNx1xKillfocus),
  EV_CBN_KILLFOCUS (idc_x2min_bnd,  EvCBNx2nKillfocus),
  EV_CBN_KILLFOCUS (idc_x2max_bnd,  EvCBNx2xKillfocus),
```

```
  EV_WM_DESTROY,
  EV_WM_SYSCOMMAND,
  EV_COMMAND     (cms_Restore,    CmRestore),
  EV_COMMAND     (cms_Reset,      CmReset),
END_RESPONSE_TABLE;
//----------------------------------------------------------------------------
TRegionBndsDlg :: TRegionBndsDlg (TWindow        * parent,
                        TRegionBndsTrs1 & ds,
                        TRectangleRgn   * rgn,
                        TResId          resId,
                        TModule         * module)
          : TDialog (parent, resId, module), iniData (0), rbtData (0)
{
       ivGeom = new TGroupBox (this, idc_geom);
             new TRadioButton (this, idc_XY,   ivGeom);
             new TRadioButton (this, idc_RZ,   ivGeom);
             new TRadioButton (this, idc_RTh,  ivGeom);
             new TRadioButton (this, idc_x1x2, ivGeom);
   ivx1minprmpt = new TStatic (this, idc_x1min_prompt, MXSTAT);
   ivx1maxprmpt = new TStatic (this, idc_x1max_prompt, MXSTAT);
   ivx2minprmpt = new TStatic (this, idc_x2min_prompt, MXSTAT);
   ivx2maxprmpt = new TStatic (this, idc_x2max_prompt, MXSTAT);
             ivx1minprmpt -> EnableTransfer ();
             ivx1maxprmpt -> EnableTransfer ();
             ivx2minprmpt -> EnableTransfer ();
             ivx2maxprmpt -> EnableTransfer ();
     ivx1min = new TEdit (this, idc_x1min, MXEDIT);
     ivx1max = new TEdit (this, idc_x1max, MXEDIT);
     ivx2min = new TEdit (this, idc_x2min, MXEDIT);
     ivx2max = new TEdit (this, idc_x2max, MXEDIT);
   ivx1minbnd = new TComboBox (this, idc_x1min_bnd, MXCOMBO);
   ivx1maxbnd = new TComboBox (this, idc_x1max_bnd, MXCOMBO);
   ivx2minbnd = new TComboBox (this, idc_x2min_bnd, MXCOMBO);
   ivx2maxbnd = new TComboBox (this, idc_x2max_bnd, MXCOMBO);
    ivNx1prmpt = new TStatic (this, idc_Nx1_prompt, MXSTAT);
    ivNx2prmpt = new TStatic (this, idc_Nx2_prompt, MXSTAT);
             ivNx1prmpt -> EnableTransfer ();
             ivNx2prmpt -> EnableTransfer ();
   ivNx1 = new TEdit (this, idc_Nx1, MXINT);
   ivNx2 = new TEdit (this, idc_Nx2, MXINT);
ivx1unitsprmpt = new TStatic (this, idc_x1units_prompt, MXSTAT);
ivx2unitsprmpt = new TStatic (this, idc_x2units_prompt, MXSTAT);
             ivx1unitsprmpt -> EnableTransfer ();
             ivx2unitsprmpt -> EnableTransfer ();
ivx1units = new TComboBox (this, idc_x1units, MXCOMBO);
ivx2units = new TComboBox (this, idc_x2units, MXCOMBO);
//     ivx1min -> SetValidator (new TNumValidator (-1000.0, 1000));
       ivx1min -> SetValidator (new TFilterValidator ("-+.0123456789eE"));
       ivx2max -> SetValidator (new TFilterValidator ("-+.0123456789eE"));
       ivNx1   -> SetValidator (new TRangeValidator ( 3, 128));
       ivNx2   -> SetValidator (new TRangeValidator ( 3, 128));

 if ( rgn )
   {     if ((rgn->X2())->id == 2)
```

```
    rbtData = new TRegionBndsTrs1 ( rgn->X2(), rgn->X1() );
            else
    rbtData = new TRegionBndsTrs1 ( rgn->X1(), rgn->X2() );
   }
  else
    rbtData = new TRegionBndsTrs1 ( ds );
    iniData = &ds;
                        Geom = (rgn ? 1 : 4);
              if (rbtData->svXY)   Geom = 1;
              if (rbtData->svRZ)   Geom = 2;
              if (rbtData->svRTh)  Geom = 3;
              if (rbtData->svx1x2) Geom = 4;
 SetTransferBuffer ( rbtData );
                            errret = IDOK;
//   SetBkgndColor ( COLOR_APPWORKSPACE );
//   SetBkgndColor ( RGB ( 0, 127, 127) );
}
//-----------------------------------------------------------------------
TRegionBndsDlg :: ~TRegionBndsDlg ()
{
//   Destroy (errret);
 if (rbtData) delete rbtData;
         else MessageBeep (-1);
// if (iniData) delete iniData;
//>>.
//      GetApplication() -> EnableCtl3d (FALSE);
}
//-----------------------------------------------------------------------
void TRegionBndsDlg::SetupWindow ()
{
  HMENU hSysMnu;
                    TDialog::SetupWindow ();
hSysMnu = GetSystemMenu ( );
  AppendMenu (hSysMnu, MF_SEPARATOR, 0, NULL);
  AppendMenu (hSysMnu, MF_STRING, cms_Restore, "Restore &Initial");
  AppendMenu (hSysMnu, MF_STRING, cms_Reset,   "Reset &Default");
//      GetApplication() -> EnableCtl3d (TRUE);
};
//-----------------------------------------------------------------------
void TRegionBndsDlg :: UpdatePrompts ()
{
  TransferData (tdSetData);
};
//-----------------------------------------------------------------------
BOOL TRegionBndsDlg :: IsValid (char * errmsg)
{
  char * erp;
  double  res1, res2;
  long   les;
                strcpy (errmsg, "okay");
 erp = 0;       res1 = strtod ( rbtData->svx1min, &erp );
  if (*erp) { strcpy (errmsg,"Bad x1min."); return FALSE; }
  if (((Geom==2)||(Geom==3))&&(res1 < 0.0))
{ strcpy (errmsg,"Radius must be positive."); return FALSE; }
```

```
 erp = 0;        res2 = strtod ( rbtData->svx1max, &erp );
  if (*erp) { strcpy (errmsg,"Bad x1max."); return FALSE; }
  if (res1 >= res2) { strcpy (errmsg,"x1min >= x1max."); return FALSE; }
 erp = 0;        res1 = strtod ( rbtData->svx2min, &erp );
  if (*erp) { strcpy (errmsg,"Bad x2min."); return FALSE; }
 erp = 0;        res2 = strtod ( rbtData->svx2max, &erp );
  if (*erp) { strcpy (errmsg,"Bad x2max."); return FALSE; }
  if (res1 >= res2) { strcpy (errmsg,"x2min >= x2max."); return FALSE; }
  if (((Geom==2)||(Geom==3))&&(rbtData->svx1minbnd.GetSelIndex()==(int)PeriodicBnd))
{ strcpy (errmsg,"Missmatch boundary (Periodic) for radii."); return FALSE; }
  if ((Geom==1)&&(rbtData->svx1minbnd.GetSelIndex()==(int)AxisymmetricBnd))
{ strcpy (errmsg,"Missmatch boundary for Xmin."); return FALSE; }
  if (rbtData->svx1maxbnd.GetSelIndex()==(int)AxisymmetricBnd)
{ strcpy (errmsg,"Missmatch outer boundary (Axisymmetric)."); return FALSE; }
  if (rbtData->svx2minbnd.GetSelIndex()==(int)AxisymmetricBnd)
{ strcpy (errmsg,"Missmatch boundary (not Radial)."); return FALSE; }
  if (rbtData->svx2maxbnd.GetSelIndex()==(int)AxisymmetricBnd)
{ strcpy (errmsg,"Missmatch boundary (Axisymmetric not Radial)."); return FALSE; }
  if (((rbtData->svx1minbnd.GetSelIndex()==(int)PeriodicBnd)&&
      (rbtData->svx1maxbnd.GetSelIndex()!=(int)PeriodicBnd))||
     ((rbtData->svx1minbnd.GetSelIndex()!=(int)PeriodicBnd)&&
      (rbtData->svx1maxbnd.GetSelIndex()==(int)PeriodicBnd)))
{ strcpy (errmsg,"Missmatch Periodic boundaries for X1."); return FALSE; }
  if (((rbtData->svx2minbnd.GetSelIndex()==(int)PeriodicBnd)&&
      (rbtData->svx2maxbnd.GetSelIndex()!=(int)PeriodicBnd))||
     ((rbtData->svx2minbnd.GetSelIndex()!=(int)PeriodicBnd)&&
      (rbtData->svx2maxbnd.GetSelIndex()==(int)PeriodicBnd)))
{ strcpy (errmsg,"Missmatch Periodic boundaries for X2."); return FALSE; }
  if (((rbtData->svx2units.GetSelIndex()==5)||
      (rbtData->svx2units.GetSelIndex()==6))&&
      (Geom != 3))
{ strcpy (errmsg,"Missmatch Units for X2 (not Angle)."); return FALSE; }
 erp = 0;        les = strtol ( rbtData->svNx1, &erp, 10 );
  if (*erp) { strcpy (errmsg,"Bad Nx1."); return FALSE; }
  if ((les < 3)||(les > 512))
    { strcpy (errmsg,"Nx1 must be in range 3 - 512."); return FALSE; }
 erp = 0;        les = strtol ( rbtData->svNx2, &erp, 10 );
  if (*erp) { strcpy (errmsg,"Bad Nx2."); return FALSE; }
  if ((les < 3)||(les > 512))
    { strcpy (errmsg,"Nx2 must be in range 3 - 512."); return FALSE; }
 return TRUE;
};
//---------------------------------------------------------------------------
BOOL TRegionBndsDlg :: CanClose ()
{
 char ErrorMessage [80];
   TransferData (tdGetData);              errret = IDOK;
 if (!IsValid (ErrorMessage))
   {  strcat (ErrorMessage, "\n Return for Edit ?");
                    MessageBeep ( -1 );
          SetDefaultId (idc_x2min);
       if (BWCCMessageBox (HWindow, ErrorMessage,"Errors",
            MB_ICONEXCLAMATION | MB_YESNO) == IDYES)
                 { errret = IDOK;    return FALSE; }
```

```
                else { errret = IDCANCEL; return TRUE;  }
   };
 if ( ::MessageBox (0,"Transfer Data ?","PICSIM",
              MB_ICONQUESTION | MB_YESNO ) == IDYES)
    {
     Transfer (iniData, tdGetData);
     errret = IDOK;
     return TRUE;
    }
 else return FALSE;
};
//---------------------------------------------------------------------
void TRegionBndsDlg :: CloseWindow (int retVal)
{
 if (errret != IDOK)   TDialog::CloseWindow (errret);
        else        TDialog::CloseWindow (retVal);
//>>.
}
//---------------------------------------------------------------------
void TRegionBndsDlg :: EvDestroy ()
{
//>>.
   TDialog::EvDestroy ();
}
//---------------------------------------------------------------------
void TRegionBndsDlg::EvSysCommand (UINT id, TPoint& pt)
{
                switch (id)
   { case cms_Reset:    CmReset ();     break;
     case cms_Restore:  CmRestore ();   break;
          default:  TDialog::EvSysCommand (id, pt);
   }
};
//---------------------------------------------------------------------
void TRegionBndsDlg::CmReset ()      // default
{
  delete rbtData;
       rbtData = new TRegionBndsTrs1 ();
 SetTransferBuffer ( rbtData );
  TransferData (tdSetData);
               MessageBeep (-1);
};
//---------------------------------------------------------------------
void TRegionBndsDlg::CmRestore ()     // initial
{
  delete rbtData;
       rbtData = new TRegionBndsTrs1 (*iniData);
 SetTransferBuffer ( rbtData );
  TransferData (tdSetData);
 if (Transfer (rbtData, tdSetData) < 10) MessageBeep (-1);
};
//---------------------------------------------------------------------
void TRegionBndsDlg :: EvCBNx1nKillfocus ()
{
```

```
  int ind = ivx1minbnd -> GetSelIndex ();
  if (ind == PeriodicBnd) ivx1maxbnd -> SetSelIndex (PeriodicBnd);
};
//-----------------------------------
void TRegionBndsDlg :: EvCBNx1xKillfocus ()
{
  int ind = ivx1maxbnd -> GetSelIndex ();
  if (ind == PeriodicBnd) ivx1minbnd -> SetSelIndex (PeriodicBnd);
};
//-----------------------------------
void TRegionBndsDlg :: EvCBNx2nKillfocus ()
{
  int ind = ivx2minbnd -> GetSelIndex ();
  if (ind == PeriodicBnd) ivx2maxbnd -> SetSelIndex (PeriodicBnd);
};
//-----------------------------------
void TRegionBndsDlg :: EvCBNx2xKillfocus ()
{
  int ind = ivx2maxbnd -> GetSelIndex ();
  if (ind == PeriodicBnd) ivx2minbnd -> SetSelIndex (PeriodicBnd);

};
//-----------------------------------
void TRegionBndsDlg :: EvBNClicked1 ()
{
 if (Geom != 1)
  {
 if (BWCCMessageBox (0,"Set Cartesian Coordinate System.\n Are You Sure?",
              "Change Geometry",
            MB_ICONQUESTION | MB_YESNO ) == IDYES)
          Geom = 1;
 ChangeGeometry ( Geom );
  }
};
//-----------------------------------
void TRegionBndsDlg :: EvBNClicked2 ()
{
 if (Geom != 2)
   {
 if (::MessageBox (0,"Set Cylindric Coordinate System.\n Are You Sure?",
              "Change Geometry",
            MB_ICONQUESTION | MB_YESNO ) == IDYES)
          Geom = 2;
 ChangeGeometry ( Geom );
   }
};
//-----------------------------------
void TRegionBndsDlg :: EvBNClicked3 ()
{
 if (Geom != 3)
    {
 if (MessageBox ("Set Polar Coordinate System.\n Are You Sure?",
              "Change Geometry",
            MB_ICONQUESTION | MB_YESNO ) == IDYES)
```

```
        Geom = 3;
ChangeGeometry ( Geom );
   }
};
//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
void TRegionBndsDlg :: EvBNClicked4 ()
{
if (Geom != 4)
   {
if (::MessageBox (0,"Set General Orthogonal Coordinate System.\n Are You Sure?",
              "Change Geometry",
           MB_ICONQUESTION | MB_YESNO ) == IDYES)
        Geom = 4;
ChangeGeometry ( Geom );
    }
};
//-----------------------------------------------------------------------
void TRegionBndsDlg :: ChangeGeometry (int gfw)
{
   TransferData (tdGetData);
      switch (gfw)
 {
 case 1:
   {  strcpy ( rbtData->svx1minprmpt, "X min" );
      strcpy ( rbtData->svx1maxprmpt, "X max" );
      strcpy ( rbtData->svNx1prmpt,   "Nx" );
      strcpy ( rbtData->svx1unitsprmpt, "X units" );
          break;
   }
 case 2:
 case 3:
   {  strcpy ( rbtData->svx1minprmpt, "R min" );
      strcpy ( rbtData->svx1maxprmpt, "R max" );
      strcpy ( rbtData->svNx1prmpt,   "Nr" );
      strcpy ( rbtData->svx1unitsprmpt, "R units" );
          break;
   }
 case 4:
   {  strcpy ( rbtData->svx1minprmpt, "X1 min" );
      strcpy ( rbtData->svx1maxprmpt, "X1 max" );
      strcpy ( rbtData->svNx1prmpt,   "Nx1" );
      strcpy ( rbtData->svx1unitsprmpt, "X1 units" );
          break;
   }
 default:    break;
 }
      switch (gfw)
 {
 case 1:
   {  strcpy ( rbtData->svx2minprmpt, "Y min" );
      strcpy ( rbtData->svx2maxprmpt, "Y max" );
      strcpy ( rbtData->svNx2prmpt,   "Ny" );
      strcpy ( rbtData->svx2unitsprmpt, "Y units" );
          break;
```

```
    }
  case 2:
    {  strcpy ( rbtData->svx2minprmpt, "Z min" );
       strcpy ( rbtData->svx2maxprmpt, "Z max" );
       strcpy ( rbtData->svNx2prmpt,   "Nz" );
       strcpy ( rbtData->svx2unitsprmpt, "Z units" );
           break;
    }
  case 3:
    {  strcpy ( rbtData->svx2minprmpt, "Th min" );
       strcpy ( rbtData->svx2maxprmpt, "Th max" );
       strcpy ( rbtData->svNx2prmpt,   "Nth" );
       strcpy ( rbtData->svx2unitsprmpt, "Theta" );
//           rbtData->svx2units.Clear ();
//           rbtData->svx2units.AddString ("radians");
//           rbtData->svx2units.AddString ("degrees");
//           rbtData->svx2units.AddString ("arbitrary");
      rbtData->svx2units.Select (5);
//       ivx2units->DeleteString (3);
//       ivx2units->DeleteString (4);
//       ivx2units->ClearList ();
//       ivx2units->AddString ("radians");
//       ivx2units->AddString ("degrees");
//       ivx2units->SetSelIndex (0);
           break;
    }
  case 4:
    {  strcpy ( rbtData->svx2minprmpt, "X2 min" );
       strcpy ( rbtData->svx2maxprmpt, "X2 max" );
       strcpy ( rbtData->svNx2prmpt,   "Nx2" );
       strcpy ( rbtData->svx2unitsprmpt, "X2 units" );
           break;
    }
  default:    break;
  }
    TransferData (tdSetData);
};
//----------------------------------------------------------------------
 const char* TRegionBndsDlg :: BoundaryStr (TBoundaryType ind)
{
 return string (*GetModule(), ids_boundary00+ind).c_str ();
//   return "No Bounds";
}
//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 void  TRegionBndsDlg :: FillBoundsCb (TComboBox * cb, WORD mask, int sel)
{
 TBoundaryType i;
    for ( i = 0; i < NBoundaryTypes; i++)
  if ( mask && i ) cb -> InsertString ( BoundaryStr (i), i );
              cb -> SetSelIndex  ( sel );
};
//*************************************************************************
DEFINE_RESPONSE_TABLE1 (TInfoDlg1,TDialog)
  EV_WM_DESTROY,
```

```
END_RESPONSE_TABLE;

//---------------------------------------------------------------------------
TInfoDlg1 :: TInfoDlg1 (TWindow  * parent,
                TInfoTrs1 & ds,
                TResId    resId )
          : TDialog (parent, resId, 0)
{
    int MX1 = sizeof (ds.svi01);       int MX2 = sizeof (ds.svi11);
 new TEdit (this, idc_info01, MX1);  new TEdit (this, idc_info02, MX1);
 new TEdit (this, idc_info03, MX1);  new TEdit (this, idc_info04, MX1);
 new TEdit (this, idc_info05, MX1);  new TEdit (this, idc_info06, MX1);
 new TEdit (this, idc_info07, MX1);  new TEdit (this, idc_info11, MX2);
 new TEdit (this, idc_info12, MX2);  new TEdit (this, idc_info13, MX2);
 new TEdit (this, idc_info14, MX2);  new TEdit (this, idc_info15, MX2);
 new TEdit (this, idc_info16, MX2);  new TEdit (this, idc_info17, MX2);

      SetTransferBuffer ( &ds );
// GetApplication() -> EnableCtl3d (TRUE);
}
//---------------------------------------------------------------------
void TInfoDlg1::EvDestroy ()
{
 Parent -> PostMessage ( WM_DLG_CLOSED, WM_DLG_CLOSED );
 TDialog::EvDestroy();
};
```

4.COMPUTATIONAL CLASSES

Due to the fact that during the project implementation on various numerical algorythms are used to compute the same physical values and also to reach the main target of the project-REUSABILITY-we attempted to separate computational classes from base classes. Development of the class hierarchy with inheritance is feasible only foe families of certain numerical algoryrhms having similar structure and identical sets of input and output parameters. In this case the base class is used for late binding and replacement of the numerical algorythms on the run. The library contains only one computational class which is a prototype of the approach described.

TPoisson

The service class providing links between the computational module and the object -oriented base classes is to be made up by the classes-descendants TAbstrProcess.

TAbstrProcess
TDetermPrcs
TMomentPrcs
TStaticPrcs - TElectrostaticPrcs
TDynamicPrcs
THistoricPrcs

However, simple and laconic algorythms can be fully encapsulated into respective classes of our main structure, e. g. the FFT algorythm can be implemented both in a member-function of the class TGridVal: Spectrum ( ), and in a member-function of the class THarmonic Process::Spectrum (TPhysVal*).

```
/*********************************************************************/
/*                                                                   */
/*              ObjectOriented PIC for C++                           */
/*                  Version 1.00                                     */
/*    Copyright (c) 1993, 1994 The Group Compuetr Simulation         */
/*          High Current Electronics Institute                       */
/*          Contract SPC93-4035(F6170893W0612)                       */
/*                                                                   */
/*                        Last write 10-20-94                        */
/*********************************************************************/
#if !defined (_PS_MODEL_H)
#define     _PS_MODEL_H

#include "ps_phval.h"
#include "ps_procs.h"

/******** TAbstrModel ----------------------------------*/

class TAbstrModel
{
public:
           int idModel;      //
        IDSTRUCT *  ModelInfo;   //
       static int   ModelsCount; //
           int   ModelStatus; //
 TGeomFrameWork *GFW;
 TAbstrRegion   *Region;
 TAbstrProcess  *Processes;
 TAbstrPhysVal  *PhysVals;
      int    phvCount;
      int    prcCount;
public:
       TAbstrModel () { ModelStatus = 0; phvCount = 0; prcCount = 0; };
 virtual ~TAbstrModel () {};

 virtual void Prepare ()              {};
 virtual void SetupModel ()           {};
 virtual void Initialization ()       {};

 virtual int  DefRegion    (TAbstrRegion* Rgn);
 virtual int  AddSubRegion (TAbstrRegion* Rgn) { return 0; };
 virtual int  DefMesh    (TAbstrMesh* Mesh) = 0;
 virtual int  AssignGeom (TGeomFrameWork* Geom) = 0;

 virtual int  DefProcess     (TAbstrProcess* Prcs);
 virtual int  AddProcessToList (TAbstrProcess* Prcs);
 virtual int  DefPhysVal     (TAbstrPhysVal* PhVal);
 virtual int  AddPhysVal     (TAbstrPhysVal* PhVal);
```

```
  virtual void Run () = 0;    // Pure virtual, must be defined in Ancesors
  virtual void Stop() = 0;    // Pure virtual, must be defined in Ancesors
        int Status ()  { return ModelStatus; };
};

/******** TSimple2DModel ----------------------------------*/

class TSimple2DModel : public TAbstrModel
{
public:
       TAbstrMesh * mesh;
public:
    TSimple2DModel ();

  virtual int DefRegion   (TAbstrRegion* Rgn);
  virtual int AddSubRegion (TAbstrRegion* Rgn);
  virtual int DefMesh   (TAbstrMesh* Mesh);
  virtual int AssignGeom (TGeomFrameWork* Geom);

  virtual int DefProcess      (TAbstrProcess* Prcs);
  virtual int AddProcessToList (TAbstrProcess* Prcs);

  virtual void Run ();
  virtual void Stop();
};

/******** TElectrostaticModel ----------------------------------*/

class TElectrostaticModel : public TSimple2DModel
{
public:
    TEulerVal    * charge;
    TEulerVal    * potential;
    TEulerVal    * Ex1;
    TEulerVal    * Ex2;
    TBeamVal     * ebeam;
    THistoricVal  * time;
public:
       TElectrostaticModel ();

  virtual void Prepare ();
  virtual void SetupModel ();
  virtual void Initialization ();

  virtual int DefRegion   (TAbstrRegion* Rgn);
  virtual int AddSubRegion (TAbstrRegion* Rgn);

  virtual int DefProcess      (TAbstrProcess* Prcs);
  virtual int AddProcessToList (TAbstrProcess* Prcs);

  virtual void Run ();
  virtual void Stop();
};
```

```
/******** TPiClassicModel ----------------------------------*/

class TPiClassicModel : public TAbstrModel
{
public:
        TPiClassicModel () : TAbstrModel () {};
};

/******** TMixModel ----------------------------------*/

class TMixModel   : public TPiClassicModel
//      ,virtual public TAbstrRegion
{
public:
      TMixModel() : TPiClassicModel () {};
//              , TAbstrRegion () {};
 virtual void Run () {};
 virtual void Stop() {};
};
/*--------------------------------------*/
#endif
/*** PS_MODEL.CPP ***/
#include "ps_model.h"

/************* TAbstrModel ----------------------------------*/

  int TAbstrModel :: DefRegion (TAbstrRegion* Rgn)
{
   Region = Rgn; return (Region? 1 : 0);
}

  int TAbstrModel :: DefProcess (TAbstrProcess* Prcs)
{
   Processes = Prcs; return (Processes ? prcCount=1 : prcCount=0);
}

  int TAbstrModel :: AddProcessToList (TAbstrProcess* Prcs)
{
  if (prcCount == 0)  return DefProcess ( Prcs );
   else
    {    TAbstrProcess *Current = Processes; int cnt = 0;
   while ( Current->Next )
       { Current = Current->Next;  cnt++;
       }
   Current->Next = Prcs;     int added = ++cnt - prcCount;
            return ( added );
    }
}

  int TAbstrModel :: DefPhysVal (TAbstrPhysVal* PhVal)
{
   PhysVals = PhVal; return (PhysVals ? phvCount=1 : phvCount=0);
}
```

```
 int TAbstrModel :: AddPhysVal (TAbstrPhysVal* PhVal)
{
 if (phvCount == 0)  return DefPhysVal ( PhVal );
  else
   {    TAbstrPhysVal *Current = PhysVals; int cnt = 0;
  while ( Current->Next )
      { Current = Current->Next;  cnt++;
      }
  Current->Next = PhVal;     int added = ++cnt - phvCount;
           return ( added );
   }
}

/************ TSimple2DModel ---------------------------------*/

 TSimple2DModel :: TSimple2DModel ()
           : TAbstrModel ()
{
}

 int TSimple2DModel :: DefRegion (TAbstrRegion* Rgn)
{
 if (Rgn)
  { if ((Rgn->Type < 20) && (Rgn->Type > 0))
       return (TAbstrModel::DefRegion (Rgn));
    else  return 0;
  }  else  return 0;
}

 int TSimple2DModel :: AddSubRegion (TAbstrRegion* Rgn)
{
  return 0;
}

 int TSimple2DModel :: DefProcess (TAbstrProcess* Prcs)
{
  return  TAbstrModel :: DefProcess ( Prcs );
}

 int TSimple2DModel :: DefMesh (TAbstrMesh* Mesh)
{
  mesh = Mesh; return 0;
}

 int TSimple2DModel :: AssignGeom (TGeomFrameWork* Geom)
{
  GFW = Geom; return 0;
}

 int TSimple2DModel :: AddProcessToList (TAbstrProcess* Prcs)
{
 return  TAbstrModel :: AddProcessToList ( Prcs );
}
```

```
  void TSimple2DModel :: Run ()
{
 TAbstrProcess *Each = Processes; int cnt = 0;
   while ( Each )
        {
         Each -> Run ();
         Each = Each->Next;  cnt++;
        }
}

  void TSimple2DModel :: Stop ()
{
}

/************* TElectrostaticModel ---------------------------------*/

  TElectrostaticModel :: TElectrostaticModel ()
                : TSimple2DModel ()
{
}

  int TElectrostaticModel :: DefRegion (TAbstrRegion* Rgn)
{
   Region = Rgn; return 0;
}

  int TElectrostaticModel :: AddSubRegion (TAbstrRegion* Rgn)
{
   Region = Rgn; return 0;
}

  int TElectrostaticModel :: DefProcess (TAbstrProcess* Prcs)
{
   Processes = Prcs; return 0;
}

  int TElectrostaticModel :: AddProcessToList (TAbstrProcess* Prcs)
{
   Processes = Prcs; return 0;
}

  void TElectrostaticModel :: Prepare ()
{
}

  void TElectrostaticModel :: SetupModel ()
{
}

  void TElectrostaticModel :: Initialization ()
{
}
```

```
  void TElectrostaticModel :: Run ()
{
}

  void TElectrostaticModel :: Stop ()
{
}

/******** TPiClassicModel ----------------------------------*/
/*** PS_POISS.CPP ***/
#include "ps_poiss.h"
#include "math.h"

/******** TPoisson ----------------------------------*/

  TPoisson :: TPoisson ( TGridVal* Source, TGridVal* Result )
{
 Mesh = Source->Mesh;
   N1 = Mesh->N1;
   N2 = Mesh->N2;
   Ms = Mesh->Ms;    N = N2;
   h1 = Mesh->h1;
   h2 = Mesh->h2;
      asph = h1/h2;  asph2 = asph*asph;
 array a (N1);   // scheme coeffs at u[i-1,j]
 array b (N1);   // scheme coeffs at u[i+1,j]
 array ei(N1);   // scheme sub-coeffs at u[i,j]
 array q (N1);   // scheme coeffs at RHS
 array c (N2);   // scheme coeffs at u[i,j-1]
 array d (N2);   // scheme coeffs at u[i,j+1]
 array ej(N2);   // scheme sub-coeffs at u[i,j]
                          // e = ei+ej
 matrix Aq ( Source->data, N1, N2 );      // Charge Density f[i,j]
 matrix Ap ( Result->data, N1, N2 );      // Potential     p[i,j]

 array v (N1);        // work array
 array cosin2 (N2);    // scheme cosins
}
/*-------------------------------------------------------------------*/
 void  TPoisson :: Prepare ( )
{
 int i, j;
 point x, h (h1,h2);
   for (i=0; i<=N1; i++)
   for (j=0; i<=N2; j++)
   {                  x = (*Mesh)(i,j);
 (Mesh->GFW)->LaplaceShablon ( x, h,
                       a [i], ei [i], b [i],
                       c [j], ej [j], d [j],   q [i] );
   }
     q *= eps0;
 TBoundaryType bt1 = (Mesh->X1)->bound1;
 TBoundaryType bt2 = (Mesh->X1)->bound2;
                       BoundaryConditions (bt1, bt2);
```

```
   for (j=1; j<=Ms-1; j++)
  {                    int k = 1 << (j-1);
   for (i=1; i<=k; i++) cosin2 [k+i-1] = 2.0*cos(0.5*(2*i-1)*M_PI/k);
  }
     Ap = Aq = 0.0;

      i1 = 7; i2 = 8;
     for (j=1;  j<=3;  j++) Aq [j] = 1.0;
}
/*-------------------------------------------------------------------*/
 void TPoisson :: BoundaryConditions (TBoundaryType sbc1, TBoundaryType sbc2)
{
          switch (sbc1)
   {
 case ConductingBnd   : c [0]  = 1.0;  b [0] = 0.0;   break;
 case AxisymmetricBnd : c [0] -= a [0];              break;
 default :            c [0] -= a [0];            break;
   }
          switch (sbc2)
   {
 case ConductingBnd : c [N1] = 1.0;   a [N1] = 0.0;    break;
 case OpenBnd       : c [N1] = 1.0;   a [N1] = 1.0;    break;
 default          : c [N1] = 1.0;   a [N1] = 0.0;    break;
   }
}
/*-------------------------------------------------------------------*/
  TPoisson ::~TPoisson ( )
{
}
/*-------------------------------------------------------------------*/
 void  TPoisson :: GetSource ( TGridVal * Src )
{
   Ap = 0.0;
   Aq.assign (Src->data);
}
/*-------------------------------------------------------------------*/
 void  TPoisson :: TestSource ()
{
 for (int i=0; i < N1; i++)
 for (int j=0; j < N2; j++)
  {
    Ap[i][j] = 0.0;
     if ((i <= i2) && (i >= i1))
       {
      if (j > 0) Aq[i][j] = Aq[i][j-1];
          else  Aq[i][0] = Aq[i][N2-1];
       }
  }
}
/*-------------------------------------------------------------------*/
 void TPoisson :: TridiagSolver ( array& vp, num dc, num ds )
{
  array x (N1);
  array y (N1);
```

```
   num  d;
                    vp *= ds;
x [1] =   b [0] / ( ei [0] - dc );
y [1] =   vp[0] / ( ei [0] - dc );
     for (int i=1; i < N1; i++)
     {
          d = 1.0 / (ei [i] - dc - a [i] * x [i]);
               x [i+1] = d * b [i];
               y [i+1] = d * (vp [i] + a [i] * y [i]);
     }
vp [N1] = (vp [N1] + a [N1] * y [N1]) / (ei [N1] - dc - a [N1] * x [N1]);

for (i=N1-1; i >= 0; i--)  vp [i] = x [i+1] * vp [i+1] + y [i+1];
}
/*-------------------------------------------------------------------*/
void TPoisson :: AdditiveFactorization ( array & vs, int lev )
{
   array w1 (N1);   w1 = 0.0;    //   ( n0 );
   array w2 (vs);                //   ( vec );
   num  signt = -1.0;
   int  j, msl;
   num  dc, de, ds;
             msl = 1 << lev;
  for ( j=1; j <= msl; j++)
   {
     de = 0.5*( dc = cosin2 [j+msl-1] );      -signt;
     ds = signt * sqrt(1.0 - de*de) / msl;
                 TridiagSolver ( w2, dc, ds );
      w1 += w2;
   };
      vs = w1;
}
/*-------------------------------------------------------------------*/
void TPoisson :: TopFactorization ( array & vs )
{      int j;
  for (j=1; j <= (Ms-2); j++)  AdditiveFactorization ( vs, j );
  for (j=1; j <= (Ms-2); j++)  AdditiveFactorization ( vs, j );
                TridiagSolver ( vs,  2.0 );
                TridiagSolver ( vs, -2.0 );
}
/*-------------------------------------------------------------------*/
void  TPoisson :: PeriodicSolver   ( )
{
 array vw (N1);
 int   level, j, h;
 int   Nm = 1 << (Ms-1);
// - - - - - - - - - - - - - - - - - - - - - - - - - Up To -
 for ( level=1; level < Ms; level++)
 {
      h = 1 << (level-1);
// . . . . . . . . . . . . . . . . . . . . . . . . boundary
 vw = Aq [0] + Aq [h] + Aq [N-h];
                AdditiveFactorization ( vw, level );
    Ap [0] += vw;
```

```
    Aq [0] = 2.0*Ap [0] + Aq [h] + Aq [N-h];
//. . . . . . . . . . . . . . . . . . . . . . . . . regular
  for ( j=2*h; j <= N-2*h; j += 2*h )
   {
    vw = Aq [j] + Ap [j-h] + Ap [j+h];
                AdditiveFactorization ( vw, level );
     Ap [j] += vw;
     Aq [j] = 2.0*Ap [j] + Aq [j-h] + Aq [j+h];
   }
  };      // level
//- - - - - - - - - - - - - - - - - - - - - - - - - Top Level -
     vw = Aq [0] + 2.0*Ap [Nm];
                TopFactorization ( vw );
   Ap [Nm] += vw;
   Ap [N] = Ap [0] += vw;
//- - - - - - - - - - - - - - - - - - - - - - - - - Down To -
  for ( level=Ms-1; level > 0; level--)
  {                              h = 1 << (level-1);
     for ( j=h; j <= N-h; j += 2*h )
    {
      vw = Aq [j] + Ap [j-h] + Ap [j+h];
                AdditiveFactorization ( vw, level );
      Ap [j] += vw;
    };          // j
  };            // level
};
/*-------------------------------------------------------------------*/
 void  TPoisson :: DirichletSolver   ( )
{
 array vw (N1);
 int level, j, h;

    for ( level=1; level < Ms; level++)
  {                              h = 1 << (level-1);
       for ( j = 2*h;  j <= N-2*h;  j += 2*h )
     {
          vw = Aq [j] + Ap [j-h] + Ap [j+h];
                  AdditiveFactorization ( vw, level );
         Ap [j] += vw;
         Aq [j] = 2.0*Ap [j] + Aq [j-h] + Aq [j+h];
     }
  };
//..................................... level down to
    for ( level = Ms; level >= 1; level--)
  {                              h = 1 << (level-1);
       for ( j = h;  j <= N-h;  j += 2*h )
     {
        vw = Aq [j] + Ap [j-h] + Ap [j+h];
                  AdditiveFactorization ( vw, level );
       Ap [j] += vw;
     };          // j
  };            // level
};
/*-------------------------------------------------------------------*/
```

```
void  TPoisson :: NeumannSolver  ( )
{
 array vw (N1);
 int   level, j, h;
//- - - - - - - - - - - - - - - - - - - - - - - - - - - - Up To -
 for ( level=1; level < Ms; level++)
 {
      h = 1 << (level-1);
//. . . . . . . . . . . . . . . . . . . . . . . . . boundaries
 vw = Aq [0] + 2.0*Aq [h];                    //  0
                AdditiveFactorization ( vw, level );
       Ap [0] += vw;
       Aq [0] = 2.0*Ap [0] + 2.0*Aq [h];
 vw = Aq [N] + 2.0*Aq [N-h];                  //  N
                AdditiveFactorization ( vw, level );
   Ap [N] += vw;
   Aq [N] = 2.0*Ap [N] + 2.0*Aq [N-h];
//. . . . . . . . . . . . . . . . . . . . . . . . . . regular
 for ( j=2*h; j <= N-2*h; j += 2*h )
  {
   vw = Aq [j] + Ap [j-h] + Ap [j+h];
                AdditiveFactorization ( vw, level );
    Ap [j] += vw;
    Aq [j] = 2.0*Ap [j] + Aq [j-h] + Aq [j+h];
  }
 };     // level
//- - - - - - - - - - - - - - - - - - - - - - - - - - - - Top Level -
    vw = Aq [N] + 2.0*Ap [0];
                TopFactorization ( vw );
    Ap [N] = Ap [0] += vw;
//- - - - - - - - - - - - - - - - - - - - - - - - - - - - Down To -
  for ( level=Ms; level > 0; level--)
 {                              h = 1 << (level-1);
    for ( j=h; j <= N-h; j += 2*h )
   {
     vw = Aq [j] + Ap [j-h] + Ap [j+h];
                AdditiveFactorization ( vw, level );
     Ap [j] += vw;
   };       // j
 };         // level
};
/******** TPoisson-----------------------------------*/
/*** PS_PROCS.H ***/
#if !defined (_PS_PROCS_H)
#define     _PS_PROCS_H

#include "ps_regn.h"
#include "ps_mesh.h"
#include "ps_phval.h"

class TPoisson;

//-------------------------------------------------------------
enum TAlgorithm
```

```
 {
    naDirect1 = 1,
    naDirect2,
    naSOR,
    naADE,
 };

/**** TAbstrProcess ----------------------------------*/
class TAbstrProcess
{
protected:
       int idPrcs;
 static int   PrcsCount;
       int   PrcsStatus;
   IDSTRUCT *PrcsInfo;
public:
  TAbstrProcess * Next;
            TAbstrProcess (): PrcsStatus(0) { ++PrcsCount; Next = 0; };
      virtual ~TAbstrProcess () {};

 virtual void Prepare () = 0;    // Pure virtual, must be defined in Ancesors
 virtual void Run ()    = 0;    // Pure virtual, must be defined in Ancesors
 virtual void Stop()    = 0;    // Pure virtual, must be defined in Ancesors
       int  Status ()   { return PrcsStatus; };
 static int  GetCount () { return PrcsCount;  };
};

/**** TDetermPrcs ----------------------------------*/

class TDetermPrcs : public TAbstrProcess
{
public:
            TDetermPrcs () : TAbstrProcess() {};
      virtual void Run () {};
      virtual void Stop() {};
};

/**** TMomentPrcs ----------------------------------*/

class TMomentPrcs : public TAbstrProcess
{
public:
            TMomentPrcs () : TAbstrProcess() { };
      virtual void Run () { };
      virtual void Stop() { };
};

/**** TStaticPrcs ----------------------------------*/

class TStaticPrcs : public TAbstrProcess
{
public:
            TStaticPrcs () : TAbstrProcess() {};
      virtual void Run () {};
```

```
    virtual void Stop() {};
};

/**** TDynamicPrcs ----------------------------------*/

class TDynamicPrcs : public TAbstrProcess
{
public:
        TDynamicPrcs () : TAbstrProcess() {};
    virtual void Run () {};
    virtual void Stop() {};
};

/**** THistoricPrcs ----------------------------------*/

class THistoricPrcs : public TAbstrProcess
{
public:
        THistoricPrcs () : TAbstrProcess() {};
    virtual void Run () {};
    virtual void Stop() {};
};

/**** TElectrostatics ----------------------------------*/

class TElectrostaticPrcs : public TStaticPrcs
{
    TApplicator    Shablon0;
    TPrcsApplicator Shablon;
public:
     TElectrostaticPrcs ( TGridVal* charge, TGridVal* scpot, TAlgorithm Algorithm );
     ~TElectrostaticPrcs ( );
   virtual void Run  ();
   virtual void Stop ();

    static void SOR (int i, int j)  // probe Successes Over Relaxation
                    { p [i][j] = p [i][j] + f [i][j]; };
        void ADE (int i, int j)  // probe Altern.Directions Explicit
                    { p [i][j] = p [i][j] + f [i][j]; };
        void Iteration ()   { Mesh->LeapFrogNodes (*this, Shablon); };
        void StaticIter()   { Mesh->LeapFrogNodes ( SOR ); };
private:
     TPoisson    * Poisson;
     TRegularMesh * Mesh;
   static matrix   p;
   static matrix   f;
};
//---------------------------------------------------------------------
class TCoulombFieldsPrcs : public TStaticPrcs
{
    TApplicator Shablon;
public:
     TCoulombFieldsPrcs ( TGridVal* scpot, TGridVal* ex1, TGridVal* ex2 );
     ~TCoulombFieldsPrcs ( );
```

```
    virtual void Run  ();
    virtual void Stop ();
        void GradX1 (int i, int j)
                { Ex1 [i][j] = -(p [i+1][j] - p [i-1][j]) * xh1; };
        void GradX2 (int i, int j)
                { Ex2 [i][j] = -(p [i][j+1] - p [i][j-1]) * xh2; };
private:
     TRegularMesh * Mesh;
        matrix   p;
        matrix   Ex1;
        matrix   Ex2;
         num   xh1, xh2;
};
/*-----------------------------------------------------------------*/
#endif
/*** PS_PROCS.CPP ***/
#include "ps_procs.h"
#include "ps_poiss.h"

#include "math.h"

/******** TAbstrProcess ----------------------------------*/
    int TAbstrProcess :: PrcsCount = 0;

/******** TElectrostatics ----------------------------------*/
  matrix TElectrostaticPrcs :: p;
  matrix TElectrostaticPrcs :: f;
//-------------------------------------------------------------
  TElectrostaticPrcs :: TElectrostaticPrcs ( TGridVal  * charge,
                                TGridVal  * scpot,
                                TAlgorithm  Algorithm )
                : TStaticPrcs (),
                      Shablon ( 0 ),
                      Poisson ( 0 )
{
   matrix f ( charge->data,     // Charge Density f [i,j]
          charge->Mesh->N1,
          charge->Mesh->N2 );
   matrix p ( scpot->data,      // Scalar Potential p [i,j]
          scpot->Mesh->N1,
          scpot->Mesh->N2 );
        switch ( Algorithm )
   {
  case naDirect1 :
  case naDirect2 :
       Poisson = new TPoisson ( charge, scpot );
       Poisson -> Prepare ();
                break;
  case naSOR :
 Shablon0 = reinterpret_cast <TApplicator>(&TElectrostaticPrcs::SOR);
//Shablon = reinterpret_cast <TPrcsApplicator>(&TElectrostaticPrcs::SOR);
                break;
  case naADE :
Shablon = reinterpret_cast <TPrcsApplicator> (&TElectrostaticPrcs::ADE);
```

```
                break;
    }
}
//-------------------------------------------------------------
  TElectrostaticPrcs ::~TElectrostaticPrcs ( )
{
 if (Poisson) { delete Poisson; Poisson = 0; }
}
//-------------------------------------------------------------
 void TElectrostaticPrcs :: Run ( )
{
 if (Poisson) Poisson -> PeriodicSolver ();
}
//-------------------------------------------------------------
 void TElectrostaticPrcs :: Stop ( )
{
}

/******** TCoulombFieldsPrcs ----------------------------------*/
  TCoulombFieldsPrcs :: TCoulombFieldsPrcs ( TGridVal  * scpot,
                                TGridVal  * ex1,
                                TGridVal  * ex2 )
                : TStaticPrcs (),
                      Shablon ( 0 )
{
 matrix p ( scpot->data,      // Scalar Potential p [i,j]
        scpot->Mesh->N1,
        scpot->Mesh->N2 );
 matrix Ex1 ( ex1->data,      //
          ex1->Mesh->N1,
          ex1->Mesh->N2 );
 matrix Ex2 ( ex2->data,      //
          ex2->Mesh->N1,
          ex2->Mesh->N2 );
}
//-------------------------------------------------------------
  TCoulombFieldsPrcs ::~TCoulombFieldsPrcs ( )
{
}
//-------------------------------------------------------------
 void TCoulombFieldsPrcs :: Run ( )
{
}
//-------------------------------------------------------------
 void TCoulombFieldsPrcs :: Stop ( )
{
}

//------------------------------------------------------ end.
```