Creation VSM - models of electronic components for Proteus.

Program interface VSM SDK.

As 4 already he wrote in the previous article, all models for Proteus must use VSM API. These are .HPP- files, which are located in catalog INCLUDE of installation Proteus. Now firm Labcenter isolated VSM API into the separate product, which is extended only among the developers, but it is
possible to use old (being contained in version Proteus Professional 6.3) for the newer versions Proteus. It consists of several it is file:

- vsm.hpp - basic file API, in which are contained practically all functions and interfaces.
- vdm.hpp - API diagnostic routine for the models of micro-controllers.
- vdm11.hpp, vdm51.hpp and vdmpic.hpp - API diagnostic routine for micro-controllers Motorola, Intel x51 and Microchip PIC.

In the original documentation not all are described, where as 4 I will try to describe as far as possible all possibilities. You look the concrete examples of the application of classes and functions in the articles.

Functions of creation and removal of models.

For creation or removal of the version of the model ISIS are caused the specific functions, which must be in the list of the export of our DLL-library. Are given below the titles of these functions for the different classes of models.

Active models:

extern "C" __ declspec(dllexport) IACTIVEMODEL * createactivemodel (CHAR *device, ILICENCESERVER * ils);
extern "C" __ declspec(dllexport) VOID deleteactivemodel (IACTIVEMODEL *model);

Analog (SPICE) models:

extern "C" __ declspec(dllexport) ISPICEMODEL *createspicemodel (CHAR * device, ILICENCESERVER * ils);
extern"C" __ declspec(dllexport) VOID deletespicemodel (ISPICEMODEL *);

Digital models:

extern "C" __ declspec(dllexport) IDSIMMODEL *createsdsimmodel (CHAR * device, ILICENCESERVER * ils);
extern "C" __ declspec(dllexport) VOID deletedsimmodel (IDSIMMODEL *);

Mixed (analog-digital) models:

extern "C" __ declspec(dllexport) IMIXEDMODEL *createmixedmodel (CHAR * device, ILICENCESERVER * ils);
extern "C" __ declspec(dllexport) VOID deletemixedmodel (IMIXEDMODEL*);

Server of licensing.

Before the creation of model we compulsorily must cause the server of licensing in order to make possible Proteus to verify, can user with the keys of licensing established in it create this model, i.e., authorize model. For the authorization of model we must use function **authorize**. The second is characterized by the fact that additionally it is possible to transmit date VSM SDK. function **authorize** it is possible to cause several times with different keys, importantly so that the server of licensing at least one time would regain importance of **TRUE**.
For example, we want so that our model depending on that is whether the additional key, which corresponds to model or not, it worked in the complete or reduced version:

```
if (ils->authorize(Device_Key))
{
     //to set the key of model, we work in the complete version
}
else if (ils->authorize(Common_Key))
{
     //there is no Key of model, there is pass key Proteus, we work in the reduced version
}
else
     return FALSE;
```

There is no key generally or internal error, we leave without the creation of model.
Also, with the aid of the functions of the server of licensing we can obtain additional information about the owner of this copy Proteus:

- **DWORD getid()** - the series number Proteus;
- **CHAR * getname()** - the name of owner;
- **CHAR * getcompany()** - the name of company;
- **CHAR * getexpirydate()** - the date, after which elapses the period of support.

Classes of the simulation of electrical part.

Class IDSIMMODEL.

Digital VSM- models must be inherited from the abstract class **IDSIMMODEL**.

- **INT isdigital (CHAR * pinname)**
  This function must return **1** if result of data is digital or mixed, **0** if it is analog or it is not active model and -1 if conclusion can be analog or digital, depending on context.

- **VOID setup (IINSTANCE * instance, IDSIMCKT * dsim)**
  This function initializes the model. By it transferred the interfaces of objects in type **IINSTANCE (**copy of interface) and **IDSIMCKT (**interface of diagram). Inside this function we must initialize internal variables and also preserve to the future of reference on **IINSTANCE** and **IDSIMCKT**. Also, in this function usually remain the addresses of the interfaces of conclusions (through the call of function **IINSTANCE::getdsimpin**).

- **VOID runctrl (RUNMODES mode)**
  This function changes the state of simulation. For example, when we start, we stop or interrupt the process of simulation. This is convenient in order to stop/start internal timers or to accomplish other necessary actions.

- **VOID actuate (REALTIME time, ACTIVESTATE newstate)**
  Function is caused, if set in action actuators - standard elements of control of component. That occur two it is specific - "more/fewer" and "pressed/released". This function is used with the work with the active constituents.

- **BOOL indicate (REALTIME time, ACTIVEDATA * newstate)**
  this function is caused each time with the copying of screen. If we return value of FALSE, then this function more not will be caused. It also is used with the work with the active constituents.

- **VOID simulate (ABSTIME time, DSIMMODES mode)**
  this is caused with each change of state of conclusions. Small observation. It is caused also when we change the state of its own conclusion, so that it is necessary accurately to establish, did occur any input event and what precisely.

- **VOID callback (ABSTIME time, EVENTID eventid)**
  This is caused with the offensive of the specific time, which is assigned by call **IDSIMCKT::setcallback**. This function is conveniently used for the cadence generators.

Class IINSTANCE.

The copy of this class is transferred to function **setup** with the starting of simulation. With its aid, it is possible to obtain access practically for all properties of the version of the model. We already used it for obtaining the references on the outputs of our model in the first part. I will describe in greater detail than his function.

- **CHAR * id()**
  returns the name of the concrete copy of component. For example, if we cause this function from the model, given in the first part, we will obtain reference to the line **"OD1"**. Functions of obtaining the parameters of the version of the model.

- **CHAR * value()**
  returns the value of property **value**, which there is in each component. This conveniently for example if we simulate resistor or capacitor for the task of the value of its resistance or capacity.

- **CHAR * getstrval(CHAR * name, CHAR * defval = NULL)**
  Returns line of the value of the parameter with the name indicated. If there is no this parameter, then returns value **defval**.

- **DOUBLE getnumval (CHAR * name, DOUBLE defval = 0)**
  the same for the numerical parameter with the floating point.

- **BOOL getboolval (CHAR * name, BOOL defval = FALSE)**
  the same for the parameter of the type **BOOL**.

- **DWORD gethexval (CHAR * name, DWORD defval = 0)**
  the same for the hexadecimal integral parameter.

- **LONG getinitval (CHAR * name, LONG defval = 0)**
  the same for the integral parameter.

- **RELTIME getdelay (CHAR * name, RELTIME deftime = 0)**
  the same for the parameter of the type of delay time.

Functions of obtaining the special properties of copy.

- **IACTIVEMODEL * getactivemodel()**
  returns reference to the version of the active model, inside which we create our model. This function can be used for the installation of the connection between the graphic and electrical parts of model.

- **IINSTANCE * getinterfacemodel()**
  returns reference to the interface of basic copy, if our model is its part.

- **BOOL getmoddata (BYTE ** data, DWORD * size)**
  returns reference to the data, which can be preserved in the process of simulation for further use in the future. For example, we can preserve internal contained PZU. For its use we must indicate the property of model **MODDATA=xxx.sss** where of the **.xxx** - quantity of bytes, which will be preserved and **.sss** - their initial state.

Functions of obtaining the interfaces of conclusions.

- **SPICENODE getspicenode (CHAR * namelist, BOOL required)**
  returns the address of the interface of analog conclusion with the name indicated. The possible names of legs are indicated through the comma. Parameter **required** indicates, will be simulation interrupted by communication about the error, if this conclusion there does not exist.

- **IDSIMPIN * getdsimpin (CHAR * namelist, BOOL required)**
  the same for the numeric readouts.

- **IBUSPIN * getbuspin (CHAR * namestem, UINT base, UINT width, BOOL required)**
  the same for the digital tire. Parameter **base** indicates the first number of conclusion, **width** - quantity of conclusions, which correspond to tire.

- **IBUSPIN * getbuspin (CHAR * name, IDSIMPIN ** pins, UINT width)**
  This is formed from the list of the conclusions, assigned by parameter **pins** tire. A quantity of conclusions is assigned by parameter **width**. Functions of the conclusion of communications.

- **VOID log (CHAR * msg...)**
  This will bring in communication into the periodical of simulation. The parameters of this function are the same as in C-language function **printf**.

- **VOID warning (CHAR * msg...)**
  the same, only into the periodical will be brought in warning.

- **VOID error (CHAR * msg...)**
  the same, only into the periodical will be brought in error and simulation is interrupted.

- **VOID fatal (CHAR * msg...)**
  the same, only into the periodical will be brought in fatal error and simulation is interrupted immediately. This function must be caused only at the worst.

- **BOOL message (CHAR * msg...)**
  This is derived communication into the line of status ISIS.

Work functions with the floating up windows.

These functions can be used for creation and removal of the windows, in which the information of model is represented or the additional data input is accomplished.

- **IPOPUP * createpopup (CREATEPOPUPSTRUCT * cps)**
  this is created window with the parameters indicated.

- **VOID deletepopup (POPUPID id)**
  this is moved away the window indicated.

Other functions.

- **BOOL setvdmhlr (class ICPU *)**
  the installation of the monitor of diagnostic routine. It is used for the models of micro-controllers.

- **BOOL loadmemory (CHAR * filename, VOID * buffer, UINT size, UINT base=0, UINT shift=0)**
  load into the memory of the contained file.

Class IDSIMCKT.

The copy of class **IDSIMCKT** also is transferred with the call of function **setup**. It serves for the access to the nucleus of the simulator of digital part.

Functions of access to the internal variables ISIS.

Through them it is possible to learn for example the step of simulation, accuracy, etc.

- **DOUBLE sysvar (DSIMVARS var)**
  returns the value of internal variable with the name indicated.

- **ABSTIME systime()**
  returns the value of the current time of simulation.

Functions of installation callback.

- **EVENT * setcallback (ABSTIME evttime, IDSIMMODEL * model, EVENTID id)**
  this is established the call of the standard processor callback of model **model** in the absolute time **evttime** with the transfer of the code of event **id**.

- **EVENT * setcallbackex (ABSTIME evttime, IDSIMMODEL * model, CALLBACKHANDLERFN func, EVENTID id)**
  the same only is established processor callback, indicated through parameter **func**.

- **EVENT * setclockcallback (ABSTIME starttime, RELTIME period, IDSIMMODEL * model, CALLBACKHANDLERFN func, EVENTID id)**
  This is established the periodic call callback function with period **period**.

- **BOOL cancelcallback (EVENT * event, IDSIMMODEL * model)**
  this is moved away the established call callback. If there was the single (not periodic) call callback, then after its call it is moved away automatically.

Of the functions of creation it is main and conclusions.

These functions are used for creating the composite model, which consists of several models.

- **DSIMNODE newnode (CHAR * partid, CHAR * nodename)**
  this is created additional knot inside the model.

- **IDSIMPIN * newpin (IINSTANCE *, DSIMNODE node, CHAR * name, DWORD flags)**
  this is created additional conclusion inside the model Flags. **Flags** can take values of **DPF_INPUT** for the entrance, **DPF_OUTPUT** for the output and **DPF_BIDIR** for the bidirectional conclusion.

Other functions.

- **VOID suspend (IINSTANCE * instance, CHAR * msg)**
  stops the process of simulation with the output of communication to screen. It is convenient to use for the fixing.

- **VOID setbreak (ABSTIME breaktime)**
  Indicates that at the moment of time **breaktime** the process of digital simulation must interrupt for fulfilling the cycle of analog simulation. This is convenient to use in the mixed models, for example for reading of data from the entrance ATSP.

Class IDSIMPIN.

This class allows access to the numeric readouts of model.

Functions of obtaining state.

The state of numeric readout can take the following values:
PHI: Power High - high level, nourishment, the strongest ;(Logic 1 power rail)
SHI: Strong High - high level ;(Logic 1 active output)
WHI: Weak High - high level ;(Logic 1 passive output)
FLT: Floating - floating output, Z-state ;(high-impedance)
WUD: Undefined - conflict, weak ;(Mid voltage from analogue source)
CON: Contention ;(Mid voltage from digital conflict)
WLO: Weak Low - low level, weak ;(Logic 0 passive output)
SLO: Strong Low - low level, strong ;(Logic 0 active output)
PLO: Power Low - low level, nourishment, the strongest ;(Logic 0 power rail)
ILO - low level, very strong;
IHI - high level, very strong;
SUD - conflict, strong.
If the simultaneously two connected outputs have a state, for example, **WLO** and **WHI**, then is formed the input state of conflict **WUD**. But if one of the outputs has a state **SLO**, and the second **WHI**, then the resulting state will be **SLO**, i.e., stronger overlaps weak.

- **STATE istate()**
  returns the current input state of conclusion.

- **STATE getstate()**
  Returns the current output state of conclusion.

- **BOOL issteady ()**
  returns **TRUE**, if the state of conclusion it did not change from the previous cycle of simulation.

- **BOOL isposedge ()**
  returns **TRUE**, if the state of conclusion it changed with "0" in "1".

- **BOOL isnegedge ()**
  returns **TRUE**, if the state of conclusion it changed s "1" in "0".

- **BOOL isedge ()**
  returns **TRUE**, if the state of conclusion it changed s "1" in "0" or with "0" in "1".

Work functions with the activity of conclusions.

There can be two versions: active low level or active high level. On silence the conclusion has active high level. This can be changed either after transferring conclusion in the list of the property of component **INVERT**, or after causing function **IDSIMPIN::invert** from function **setup**.

- **BOOL invert()**
  this is transferred conclusion to the active low level. It can be caused only out of function **IDSIMMODEL::setup**.

- **INT activity ()**
  returns the current activity of the conclusion: +1 - it is active, -1 - it is inactive, 0 - state are not determined.

- **BOOL isactive ()**
  returns **TRUE**, if the current state of conclusion is active.

- **BOOL isinactive ()**
  returns **TRUE**, if the current state of conclusion is inactive.

- **VOID setstates (STATE tstate, STATE fstate, STATE zstate)**
  clearly are assigned the values of the states of high level, low level and third state of output (with the high resistance).

- **EVENT * drivebool (ABSTIME time, BOOL flag)**
  this is set conclusion value in the active (flag==**TRUE)** or inactive (**flag** ==**FALSE**) state at the moment of time **time**.

- **EVENT * drivetristate (ABSTIME time)**
  this is converted conclusion to the third state at the moment of time **time**.

Functions of the installation of state.

- **VOID setstate (STATE state)**
  this is established the initial state of conclusion in value of **state**. It can be caused only out of function **IDSIMMODEL::setup**.

- **EVENT * setstate (ABSTIME time, RELTIME tlh, RELTIME thl, RELTIME tgq, STATE state)**
  this is established the state of conclusion in value of **state**. This will be produced at the moment of time **time**, in this case are used delays
  tlh - transit time from the low level into the high,
  thl - transit time from the high level into the low,
  tgq - delay time for the removal fragment.
  If in this period there was passage into the initial state, for example, "y"-"0"-"y", then the state of conclusion does not change. Parameter **state** also can take values of **TSTATE** for the active mode or **FSTATE** for the inactive state. After switching of conclusion of one state into another, is caused callback function.

- **EVENT * setstate (ABSTIME time, RELTIME tgq, STATE state)**
  the same, but is assigned only delay time for the removal fragment.

- **EVENT * drivestate (ABSTIME time, STATE state)**
  this is established the state of conclusion in value of state.

Other functions.

- **VOID sethandler (IDSIMMODEL * model, PINHANDLERFN phf)**
  this is establishes the processor of a change of state of conclusion. On silence is caused function **IDSIMMODEL::simulate**.

- **DSIMNODE getnode()**
  returns the address of the interface of the knot, to which is attached the conclusion.

- **VOID settiming (RELTIME tlh, RELTIME thl, RELTIME tgq)**
  this is established delay times. With the call of functions **setstate** to the indication of delay times these values are added. On silence, if is not caused function **settiming**, these times are equal to 0.

Class IBUSPIN.

This class serves for the work with the tires - the collections of conclusions. Functions **settiming**, **setstates**, **sethandler** and **drivetristate** repeat the similar functions of class **IDSIMPIN**; therefore describe them not they will be.

- **VOID drivebusvalue (ABSTIME time, DWORD value)**
  this is established the state of tire in value of **value**, i.e., each of its bits corresponds to each output of tire. I.e., 1 corresponds to active mode, 0 - inactive.

- **VOID drivebitstate (ABSTIME time, UINT bit, STATE state)**
  this is established the state of the conclusion in value of **state indicated**.

- **DWORD getbusvalue()**
  returns the current input state of tire.

- **DWORD getbusdrive()**
  returns the current output state of tire.

- **STATE getbitstate(.UINT bit)**
  returns the input value of the conclusion indicated.

Class ISPICEMODEL.

The abstract class **ISPICEMODEL** serves as basis for creating the analog models. We must inherit our model from it. Methods, which form part of class.

- **INT isanalog (CHAR * pinname)**
  this is conclusion analog. We can return the following values: **1** - conclusion is analog or mixed, **0** - the conclusion of digital or it is not used by a model, **-1** - conclusion can be analog or digital depending on context.

- **VOID setup (IINSTANCE *, ISPICECKT *)**
  the function, which ISIS is caused with the creation of the copy of class. In this case it transfers the addresses of interfaces **IINSTANCE** and **ISPICECKT**, which we can preserve for the work subsequently. The basic initialization of internal variables and the installation of the initial state of model here is produced.

- **VOID runctrl (RUNMODES mode)**
  this function is caused before each cycle of rendering of screen. In this case the value of the current regime of the fulfillment of simulation is transferred.

- **VOID actuate (REALTIME time, ACTIVESTATE newstate)**
  for the active models with the aid of this function ISIS reports to our model about a change of state.

- **BOOL indicate (REALTIME time, ACTIVEDATA * newstate)**
  this is caused at the end of each cycle of rendering of screen. For the active models we can transfer here the new state of model. If we return value of **FALSE**, then this function more not will be caused.

- **VOID dcload (REALTIME time, SPICEMODES mode, DOUBLE * oldrhs, DOUBLE * newrhs)**
  Here we must count the matrices of currents and potentials. This function is caused for iteration of each cycle of simulation. Parameter **time** - this current time of simulation, **mode** – the regime of simulation, **oldrhs** - the vector of stresses for the previous iteration, **newrhs** - currents of branch. Since this function is caused very frequently, it must be written very optimally.

- **VOID acload (SPICEFREQ omega, DOUBLE * rhs, DOUBLE * irhs)**
  this function is caused during the analysis of alternating current. Its designation is analogous with function **dcload** with exception of the fact that they are used the vector of phasors instead of the vectors of stresses. Parameter **rhs** - the vector of real currents, **irhs** - vector of the imaginary currents. The models, which do not support this form of analysis, must cause communication about the error with the aid of the call of function **IINSTANCE::error**.

- **VOID trunc (REALTIME time, REALTIME * newtimestep)**
  it makes it possible to cause the additional iteration of calculations. We can obtain the cycle time of iteration with the aid of the call of function **ISPICECKT::sysvar(.SPICEDELTA)**. It is necessary to remember that too small a time will cause communication about the error "time step too small". We can obtain the minimum time of step with the aid of call **SPICECKT::sysvar(.SPICEDELMIN)**.

- **VOID accept (REALTIME time, DOUBLE * rhs)**
  this function is caused after all calculations, with the satisfactory value of vector RHS. it is possible to preserve for further calculations. To read the value of stress in the knot is possible with the aid of construction **voltage = rhs[.node]**. If knot this is the knot of branch, created with the aid of **ISPICECKT::newcurnode**, then this value will be the current of branch. Because of this it is possible to use voltage sources as the detectors of current. On the whole, the detector of current can be created with the aid of the voltage source, in which the value of stress is equal to zero.

Class ISPICECKT.

This interface represents the analog part of the diagram and is stored inside the nucleus of simulator SPICE3F5. It ensures access to the system variables simulator SPICE and also to the current and previous values of vectors RHS. also it makes possible for models to cause the cycles of simulation into the time indicated.

- **BOOL ismode (SPICEMODES flags)**
  returns value of **TRUE** if the current regime of simulation it corresponds to that indicated. Possible values of parameter **flags**:
  SPICETRAN - analysis of transient processes;
  SPICEAC - analysis of the direct current;
  SPICEDCOP - only search for operating point on the direct current;
  SPICETRANOP - operating point for the analysis of transient processes;
  SPICEDCTRANCURVE - operating point for the curve of the passage of direct current;
  SPICEINITFLOAT - is established for the convergence;
  SPICEINITJCT - is established only for the first iteration;
  SPICEINITSMSIG - special iteration before the analysis of alternating current;
  SPICEINITTRAN - is established for the first iteration of each cycle of simulation;
  SPICEUIC - is established if we model it must establish the values of the IC.

- **DOUBLE sysvar (SPICEVARS var)**
  returns the values of the internal variables SPICE. Possible values of parameter **var**:
  SPICETIME - current time of simulation;
  SPICEOMEGA - 2*.PI * (the instantaneous value of frequency) with the analysis of alternating current;
  SPICEDELTA - current step of simulation;
  SPICEGMIN - minimum time of access;
  SPICEDELMIN - minimum step of simulation;
  SPICEMINBREAK - minimum space between the points of stop;
  SPICESRCFACT - initial coefficient of iterations;
  SPICEFINALTIME - time of the end of simulation.

- **DOUBLE &statevar (INT s, INT n)**
  Access to the variables, preserved for each step. These variables are created with the aid of **ISPICECKT::allocvars**. These variables can be both read and changed. Each variable contains the collection of the preserved values, access to which can be obtained with the aid of **s**. by **0** it means that is produced the access to the variable on the current iteration, **1** - on the previous iteration, and so on. Models usually use this mechanism for storing the previous values of currents and voltages. Parameter **s** - the number of iteration, **n** - index of variable, which is obtained with the aid of call **ISPICECKT::allocvars**.

- **DOUBLE &.ryus (SPICENODE n)**
  it makes it possible to obtain access on read/write to the current vector RHS, which stores the values of the stresses, calculated on the current step. During the analysis of alternating current this function ensures access to the real part of the phasor of stress. Parameter **n** - the value of the knot, for which we obtain value.

- **DOUBLE &ryusold (SPICENODE n)**
  the same, only for the previous step of simulation.

- **DOUBLE &iryus (SPICENODE n)**
  ensures access to the imaginary part of the phasor with the analysis of alternating current at the current step.

- **DOUBLE &iryusold (SPICENODE n)**
  the same, only at the previous step.

- **SPICENODE getnode (CHAR * netname)**
  it makes it possible to obtain knot on the name.

- **SPICENODE newvoltnode (CHAR * partid, CHAR * nodename)**
  it is created new internal knot. With its aid it is possible to create internal resistors and/or voltage sources. Usually this function is caused during the initialization from **ISPICEMODEL::setup**. Parameter **partid** - the unique identifier of the prefix of knot. Usually is used the value, obtained with the aid of **IINSTANCE::id**. Parameter **nodename** - the unique end of the name of knot. It is necessary to use unique ends of name for each knot of model.

- **SPICENODE newcurnode (CHAR * partid, CHAR * nodename)**
  the same only is created new branch (current).

- **VOID delnode (SPICENODE node)**
  it makes it possible to remove the knot indicated.

- **DOUBLE * allocsmp (SPICENODE node1, SPICENODE node2)**
  it is created element inside the matrix for the simulation of the source of current or resistor. This function usually is caused from **ISPICEMODEL::setup** and returns the value, which must be preserved for the use in functions **dcload** and/or **acload**. Parameters **node1** and **node2** are the indices of line and column inside the matrices, which indicate the positions of the created element.

- **BOOL setbreak (REALTIME time)**
  it is caused the cycle of simulation into the time indicated. Its work is analogous **ISPICEMODEL::trunc**.

- **VOID suspend (IINSTANCE * instance, CHAR * msg)**
  it stops the process of simulation with the output of communication to screen. It is convenient to use for the fixing.

- **INT allocvars (INT n)**
  it is separated the given quantity of stored variables. To them it is possible to obtain access with the aid of **ISPICECKT:: statevar**.

- **VOID integrate (DOUBLE * geq, DOUBLE * ceq, DOUBLE cap, INT statevars)**
  it is caused the built-in algorithm of numerical integration. The parameter they are named such means, as if the capacitor (its internal idea - this current source) is simulated. Function works with two storable variables, which must be isolated with the aid of call **ISPICECKT::allocvars** and then there are initialized. The first stored variable contains the value of charge on the capacitor, the second - current, flowing through it. The algorithm of integration uses value of cap and these two stored variables in order to calculate the value of conductivity and current source for the idea of capacitor in the diagram. The same variables must be to transfer function **ISPICECKT::truncerr** so that SPICE would select the correct step of simulation. In order to correspond to the assigned accuracy. For the additional information be turned to the management on SPICE3F5. Parameter **geq** - indicator to the variable, in which there will be preserved calculated value of conductivity, **ceq** - indicator to the variable, in which the value of current source will be preserved, **cap** - value of capacitance of capacitor, **statevars** - initial index of the stored variables. The correct realization of numerical simulation in the program (for the simulation of capacitor or inductance coil) this is a very complex question, and to you is necessary very well to know itself SPICE in order to use this mechanism.)

- **VOID truncerr (INT statevars, DOUBLE * timestep)**
  it is caused the algorithm of the calculation of reduction built-in in SPICE. This function must be caused from function **ISPICEMODEL::trunc**.

Classes of graphic simulation.

Class IACTIVEMODEL.

This is the abstract class, from which must be inherited the models of active models. Active models make it possible to interact with the user and to sketch in the diagram. Active model must realize the following functions.

- **VOID initialize (ICOMPONENT * cpt)**
  the initialization of model. As the parameter is transferred the indicator to interface **ICOMPONENT**.

- **ISPICEMODEL * getspicemodel (CHAR * primitive)**
  ISIS causes this function when it appears the need of creating the analog model. The name of model is transferred as the parameter. For example, if we assigned in the script the parameter **PRIMITIVE=ANALOG, ammeter** that will be transmitted line **"AMMETER"**.

- **IDSIMMODEL * getdsimmodel (CHAR * primitive)**
  this function will be caused when ISIS it creates the digital model, which corresponds to active model. If in the script the parameter is assigned **PRIMITIVE=DIGITAL, display** that this function will be caused with the parameter **"DISPLAY"**.

- **VOID plot (ACTIVESTATE state)**
  this function is caused when it appears the need for the copying of component. It is differed from function **IACTIVEMODEL::animate** in terms of the fact that it is necessary to completely draw again component. Also, as the parameter the current state is transferred by it. The minimum realization of this function is the following:

```
VOID MYMODEL::plot (ACTIVESTATE state)
{
     component->drawstate (state);
}
```

- **VOID animate (INT element, ACTIVEDATA * newstate)**
  It is caused for the analysis of an event, generated inside the electric analogue (from classes **ISPICEMODEL** or **IDSIMMODEL**) for the appropriate active model. Active events are created by nucleus PROSPICE as a result of the recovered values of functions **ISPICEMODEL::indicate** and **IDSIMMODEL::indicate**, which are caused with each copying of screen. This ensures the general mechanism, with the aid of which the electric analogue can interact with the graphic. Frequently this mechanism is used if you plan to use primitives **RTVPROBE**, **RTIPROBE** or **RTDPROBE** in order to take the simple measurements, which you then use in the graphic model. Parameter element is the element of graphic model, for which is generated the event. This special feature makes it possible to use several primitives **RTVPROBE**, who are located in the model in order to conduct measurements by one graphic component. Parameter **event** is indicator in the data, transmitted with the generation of event.

- **BOOL actuate (WORD key, INT x, INT y, DWORD flags)**
  this function is caused with the pushing of the knobs of keyboard and with motion or pushing of the knobs of mouse. It is used in the models of actuators. This function is caused only if mouse pointer it is located above the component. Also, if function returns value of **TRUE**, this means that it seizes entire subsequent flow of the events of keyboard and mouse until it returns value **FALSE**. Parameter **key** - the code of the virtual key Windows, **x** and **y** - the coordinate of mouse pointer relative to **ORIGIN**, **flags** - pressed buttons of the mouse: **1** - leftist, **2** - right.

Class ICOMPONENT.

This interface makes it possible to work with the graphic model in the active constituents. It is possible to use it in order to sketch in the diagram or for interaction with the user. Indicator to the interface is transferred in the form of the parameter with the call of function **IACTIVEMODEL::initialize**.

Functions of access to the properties of component.

- **CHAR * getprop (CHAR * name)**
  to obtain the value of line property.

- **CHAR * getproptext (VOID)**
  to obtain the value of property TEXT.

- **VOID addprop (CHAR * propname, CHAR * item, WORD hflags)**
  to add new property to the component or to change value and/or flags of that existing. Parameter **propname** - the name of property, **item** - its value, **hflags** can take the values: **SHOW_ALL** - to reflect all parts of the property, **HIDE_ALL** - not to reflect all parts of the property, **HIDE_KEYWORD** - not to reflect the keywords, **HIDE_VALUE** - not to reflect value.

- **VOID delprop (CHAR * propname)**
  to remove the properties of component.

- **VOID setproptext(CHAR * text)**
  to establish the new value of property TEXT.

Work with the current state of component.

- **ACTIVESTATE getstate (INT element, ACTIVEDATA * data)**
  to obtain the current state of the element of active constituent indicated and the value of its data.

- **BOOL setstate (ACTIVESTATE state)**
  it is established the current state of active constituent.

Control of drawing.

- **VOID setdrawscale (INT ppi)**
  to establish the scale of drawing at points on in.

- **HDC begincache (BOX &area)**
  it begins caching of rendering in the section of screen indicated. Also, this function returns the standard descriptor of context HDC, which makes it possible to use the standard functions Windows for the drawing in this section.

- **HDC begincache (INT symbol)**
  the same, but only for the symbol with the number indicated.

- **VOID endcache()**
  Concludes caching and it causes it reflects the something has been cached section of screen.

- **VOID repaint (BOOL erase)**
  it is caused the copying of component. Parameter **erase** indicates, will be first produced erasure.

Work functions with the vector drawing.

- **HGFXSTYLE creategfxstyle (CHAR * name=NULL)**
  it creates and makes that flowing new graphic style. Graphic style includes the width of feather, the style of painting and color. Usually model creates the collection of graphic styles during the initialization, preserving the values in the internal variables returned, and subsequently are used they.

- **VOID selectgfxstyle (HGFXSTYLE style)**
  it is made itself graphic the style indicated that flowing.

- **VOID setpenwidth (INT w)**
  it is established the width of feather in the current style indicated.

- **VOID setpencolour (COLOUR c)**
  it is established the color of feather in the current style indicated.

- **VOID setbrushcolour (COLOUR c)**
  it is established the color of brush in the current style indicated.

- **VOID drawline (INT x1, INT y1, INT x2, INT y2)**
  sketches line in the coordinates indicated.

- **VOID drawbox (INT x1, INT y1, INT x2, INT y2)**
  sketches rectangle in the coordinates indicated.

- **VOID drawbox (BOX &bkh)**
  the same, only coordinates are assigned through the structure.

- **VOID drawcircle (INT x, INT y, INT radius)**
  sketches circle with the center in the coordinates with a diameter **of radius indicated**.

- **VOID drawbezier (POINT *, INT numpoints=4)**
  Sketches curved arc.

- **VOID drawpolyline (POINT *, INT numpoints)**
  sketches broken line.

- **VOID drawpolygon (POINT *, INT numpoints)**
  the same only locks end and beginning of line, i.e., sketches polygon.

- **VOID drawsymbol(INT symbol)**
  it is caused render of the symbol indicated.

- **VOID drawsymbol(INT x, INT y, INT rot, INT mir, INT symbol)**
  the same in the coordinates indicated, in this case turning is counterclockwise indicated by parameter **rot**, and parameter **mir** indicates the reflection of the symbol: **0** - without the reflection, **MIR_X** - reflection relative to axis X, **MIR_Y** - reflection relative to Y axis in this order: reflection, turning, the installation of coordinates.

- **VOID drawstate (ACTIVESTATE state)**
  symbol in the state indicated.

- **BOOL getsymbolarea (INT symbol, BOX * area)**
  there are obtained the coordinates of the symbol indicated.

- **BOOL getmarker (CHAR * name, POINT * pos=NULL, INT * rot=NULL, INT * mir=NULL)**
  find coordinates turning and reflection of marker with the name indicated.

Functions of the conclusion of text.

- **HTEXTSTYLE createtextstyle (CHAR * name=NULL)**
  it creates the new style of text with the name indicated and makes with its with that flowing.

- **VOID selecttextstyle (HTEXTSTYLE style)**
  it is selected by that flowing the style of text indicated.

- **VOID settextfont (CHAR * name)**
  it is established type in the current style of text.

- **VOID settextsize (INT h)**
  it is established the size of type in the current style of text.

- **VOID setbold (BOOL f)**
  indicates, will be type fatty in the current style of text.

- **VOID setitalic (BOOL f)**
  indicates, will be type inclined in the current style of text.

- **VOID setunderline (BOOL f)**
  indicates, will be type that emphasized in the current style of text.

- **VOID settextcolour (COLOUR c)**
  it is established the color of type in the current style of text.

- **VOID drawtext (INT x, INT y, INT rot, INT jflags, CHAR * text...)**
  reflects text in the coordinates with turning **rot indicated**. Parameter **jflags** indicates the leveling off of the text:
  TXJ_LEFT - text is equalized on the left edge;
  TXJ_RIGHT - text is equalized on the right edge;
  TXJ_CENTRE - text is equalized on to center;
  TXJ_BOTTOM - text is located below point;
  TXJ_TOP - text is located above point;
  TXJ_MIDDLE - text is located through to center points.
  Parameters **text** and further the same as in function **printf**.

Functions of the support of the floating up windows.

- **IPOPUP * createpopup (CREATEPOPUPSTRUCT * cps)**
  it is created the floating up window with the parameters indicated.

- **VOID deletepopup (POPUPID id)**
  it is moved away surfacing the window indicated.

Other functions.

- **VOID settimestep (DOUBLE time)**
  it is established the time of one step of simulation.

- **VOID error (CHAR * msg...)**
  it derives communication about the error into the periodical and stops simulation.

Floating up windows.

For some models appears the need of creating the floating up windows, in which the information about the state of model is represented or control of it is accomplished. In ISIS there are several versions of the standard floating up windows: **IDEBUGPOPUP** - check-out window, into which it is possible to derive lines. In this case is achieved scrolling of information as in the standard periodical of simulation.
ISTATUSPOPUP - window of state, in which it is possible to derive the formatted text.
IMEMORYPOPUP - window of the dump of memory, into which usually is derived contained PZU or OZU of micro-controller.
IUSERPOPUP - window controlled by user with the aid of functions Windows.
IWATCHPOPUP - window, in which are reflected the assigned variables with the address indicated.
ISOURCEPOPUP - window, in which is located the source text of the program of micro-controller.
IVARPOPUP - window, in which are reflected the variables of micro-controller.

Creation of the floating up window.

The floating up window can be created with the aid of functions **ICOMPONENT::createpopup** or **IINSTANCE::createpopup** depending on whether our model of active does appear or not. In both cases are used structure **CREATEPOPUPSTRUCT**, in which are contained the following fields:
POPUPID id - the unique identifier of the window of model;
POPUPTYPES type - the type of the created window. It can be the following:
PWT_USER - window of the type **IUSERPOPUP**;
PWT_DEBUG - window of the type **IDEBUGPOPUP**;
PWT_STATUS - window of the type **ISTATUSPOPUP**;

PWT_MEMORY - window of the type **IMEMORYPOPUP**;
PWT_WATCH - window of the type **IWATCHPOPUP**;
PWT_SOURCE - window of the type **ISOURCEPOPUP**;
PWT_VAR - window of type **IVARPOPU**P.

CHAR * caption - the title of window;
INT width, height - width and the height of window;
DWORD flags - the attributes of window. They can be following:
PWF_VISIBLE - window seen;
PWF_SIZEABLE - the dimensions of window can be changed;
PWF_LOCKPOSITION - the position of window is not memorized;
PWF_HIDEONANIMATE - window is placed on screen only during the pause;
PWF_AUTOREFRESH - window draws again each time with the copying of screen;
PWF_WANTKEYBOARD - when window is active, it will obtain symbols from the keyboard.
Usually at the end of simulation ISIS it destroys all created windows. But if need arose, window can be destroyed, after causing functions **ICOMPONENT::deletepopup** or **IINSTANCE::deletepopup**.

Class IUSERPOPUP.

This is the user window, in which it is possible to use functions Windows for creating the elements of control and mapping of information.

- **CHAR * getprop (CHAR * key)**
  returns the value of the preserved parameter of window. Model can preserve the parameters of the windows, which then can be read with the following simulation.

- **VOID setprop (CHAR * key, CHAR * value)**
  it is established the value of the parameter of window.

- **VOID setmsghlr (IMSGHLR * handler)**
  Establishes the processor of communications for the window.

- **LRESULT callwindowproc (MESSAGE msg, WPARAM warg, LPARAM larg)**
  Transfers communication Windows API for the working in ISIS. Usually this is done at the end of the processor for all unprocessed communications.

Class IDEBUGPOPUP.

This class is used for the conclusion of check-out information.

- **VOID print (CHAR * msg...)**
  the conclusion of line into the window. The parameters are the same as in function **printf**.

- **VOID dump (const byte * ptr, UINT nbytes, UINT base=0)**
  it is derived the dump of the quantity of bytes of memory from buffer **ptr indicated**. Parameter **base** indicates the constant, which is added to the counter of the numbers of those concluded it is byte.

Class ISTATUSPOPUP.

Class **ISTATUSPOPUP** is the window of status, into which it is possible to derive the formatted text.

- **VOID setarea (UINT columns, UINT rows, UINT border, BOOL copy)**
  it is created rectangle with the quantity of columns and lines indicated and with a width of boundary **of border**.

- **VOID setcursorto (UINT pixelx, UINT pixely)**
  it is established cursor in the coordinates indicated.

- **UINT getcharwidth (VOID)**
  returns the width of symbols.

- **UINT getcharheight (VOID)**
  returns the height of symbols.

- **VOID print (CHAR * message...)**
  it is derived text in the coordinates indicated. The parameters of function are the same as in function **printf**.

- **VOID print (INT col, INT row, COLOUR textcolour, CHAR * msg...)**
  the same, only additionally is indicated position in the symbols, where will be derived text, and its color.

- **VOID setctabstops (const int ts[ ], INT n)**
  There are established the positions of tabulations for the conclusion of text.

- **VOID setptabstops (const int ts[ ], INT n)**
  the same. It is necessary to a little experiment. In order to understand difference between them, for thus far I do not know.

- **VOID clear (COLOUR bkcolour=NOCOLOUR)**
  erases contents of window and paints by its color indicated.

- **BOOL setredraw (BOOL on_off, BOOL redraw_now)**
  indicates, the copying of window with each copying of screen will be produced. Also it is possible to cause immediate drawing. The old state of automatic copying returns.

- **VOID repaint (VOID)**
  it is caused the immediate copying of window.

Class IMEMORYPOPUP.

In the window, created the use of this class, it is possible to reflect the dumps of memory. For example, it is possible to derive dump OZU or PZU in the model of micro-controller. This window draws again each time with the copying of window, so that it is better to cause with flag **PWF_HIDEONANIMATE**.

- **VOID setmemory (ADDRESS baseaddr, BYTE * data, UINT nbytes)**
  indicates that will be reflected data from buffer **data**, by quantity **nbytes**. In this case to the initial address, which is placed on screen, will be added **baseaddr**.

- **VOID repaint (VOID)**
  it is caused the forced copying of window.

Class ISOURCEPOPUP.

This is the window, in which is reflected the source text of the program of micro-controller. Also, it is possible to derive the disassembled text of piercing. This interface is not described in the documentation, so that I write here that the fact that I think about its functions, and subsequently, when I create a test example and try it everything, I will describe in greater detail.

- **BOOL addsdifile (CHAR * file)**
  Load the file, in which are described the addresses of commands and point of stop.

- **BOOL setpcaddr (ADDRESS addr)**
  it is established the address of the entrance into the program.

- **BOOL isbreakpoint (ADDRESS addr)**
  it checks, is there with the address indicated the point of stop.

- **BOOL iscurrentline (ADDRESS addr)**
  it checks, does belong the address indicated to the current line of program.

- **BOOL findfirstbpt (ADDRESS * addr)**
  the search for the first point of stop. The result of search returns. it found or not, and also is established the address of this point in the variable **addr**.

- **BOOL findnextbpt (ADDRESS * addr)**
  the same, only search for the following point of stop.

- **BOOL addsrcfile (CHAR * file, BOOL lowlevel)**
  Load file with the source texts.

- **VOID addcodeline (INT srclinenum, ADDRESS address)**
  it is established a correspondence between the line of program and the address.

- **VOID addcodelabel (CHAR * label, ADDRESS address)**
  it is established marker with the address indicated.

- **VOID update ()**
  it is produced the renovation of internal buffers.

- **BOOL getsteptoaddr (ADDRESS * addr)**
  returns in the variable **addr** the address of program, which corresponds to the following step. Also, if oCurred error, returns **FALSE**.

- **VOID setinsertpos (INT fileid, INT linenum, BOOL newblock)**
  it is not understandable.

- **VOID insertline (ADDRESS addr, CHAR * opcodes, CHAR * srctext)**
  it is added with the address indicated the line of the source text and also the commands, which correspond to it.

- **BOOL findfirstsrcline (ADDRESS * addr)**
  the search for the address of the first line with the source text.

- **BOOL findnextsrcline (ADDRESS * addr)**
  the search for the following line with the source text.

- **CHAR * findlabel (ADDRESS addr)**
  the search for the marker, located with the address indicated.

Class IVARPOPUP.

This class is intended for the conclusion into the window of the variables of micro-controller. It also is not described in the documentation.

- **VOID setcpu (ICPU * cpu)**
  it is established the context of the micro-controller, to which the window corresponds.

- **VOID additem (VARITEM * vip)**
  it is added into the window the assigned variable. Structure **VARITEM** contains the following fields:
  CHAR name [WATCHITEM_NAME_SIZE] - the name of variable;
  DWORD loader, seg - is not understandable.
  ADDRESS address - the address of variable.
  DATATYPES type - the type of data. It can take the following values: **DT_VOID**, **DT_STRING**, **DT_TEXT**, **DT_BYTE**, **DT_WORD**, **DT_DWORD**, **DT_QWORD**, **DT_IEEE_FLOAT**, **DT_IEEE_DOUBLE**, **DT_HTEC_FLOAT**, **DT_MCHP_FLOAT**, **DT_BIGENDIAN**.
  DISPFORMATS format - the size of mapping the variable: **DF_VOID**, **DF_BINARY**, **DF_OCTAL**, **DF_HEXADECIMAL**, **DF_SIGNED**, **DF_UNSIGNED**, **DF_FLOAT**, **DF_TEXT**.
  DWORD size - the size of variable.
  ADDRESS scope_begin - is not understandable.
  ADDRESS scope_end - is not also understandable.
  Simulation of micro-controllers.
  Here I will describe that the fact that I understood from it was file titles, since in the documentation it is said not word about the simulation of micro-controllers. Later I will try to make several test models of micro-controllers and will describe ever in more detail.

Class ICPU.

This abstract class serves as basis for creating its own models of micro-controllers. We must overlap the following functions.

- **LRESULT vdmhlr (VDM_COMMAND * cmd, BYTE * data)**
  The processor of command.

- **VOID loaddata (INT format, INT seg, ADDRESS address, BYTE * data, INT numbytes)**
  function must load transferred to it data into the segment indicated.

- **VOID disassemble (ADDRESS address, INT numbytes)**
  function must unassembled the quantity of bytes with the address indicated.

- **BOOL getvardata (VARITEM * vip, VARDATA * vdp)**
  function must return the data by the variable indicated. Fields of structure **VARDATA**:
  CHAR addr[WATCHITEM_ADDR_SIZE] - address into some to incomprehensible form.
  DATATYPES type - the type of variable.
  BYTE * memory - storage unit, in which is contained the variable.
  DWORD memsize - the size of this block.
  DWORD offset - displacement by variable relative to the beginning of block.

Functions of the load of program.

These functions serve for the load of contained PZU or piercing of micro-controller into the memory. These functions find in the exports of library loader.dll.

- **BOOL load_auto (CHAR * file, IINSTANCE *, ISOURCEPOPUP *, IVARPOPUP *, ICPU *)**
  The load of program from the file, with the automatic determination of size. The following functions load program from the file with the indication of size.

- **BOOL load_bin (CHAR * file, IINSTANCE *, ISOURCEPOPUP *, IVARPOPUP *, ICPU *);**

- **BOOL load_hex (CHAR * file, IINSTANCE *, ISOURCEPOPUP *, IVARPOPUP *, ICPU *);**

- **BOOL load_s19 (CHAR * file, IINSTANCE *, ISOURCEPOPUP *, IVARPOPUP *, ICPU *);**

- **BOOL load_omf51 (CHAR * file, IINSTANCE *, ISOURCEPOPUP *, IVARPOPUP *, ICPU *);**

- **BOOL load_ubrof (CHAR * file, IINSTANCE *, ISOURCEPOPUP *, IVARPOPUP *, ICPU *);**

- **BOOL load_cod (CHAR * file, IINSTANCE *, ISOURCEPOPUP *, IVARPOPUP *, ICPU *);**

- **BOOL load_basic (CHAR * file, IINSTANCE *, ISOURCEPOPUP *, IVARPOPUP *, ICPU *);**

- **BOOL load_coff (CHAR * file, IINSTANCE *, ISOURCEPOPUP *, IVARPOPUP *, ICPU *).**

NOTE:

1. Type of REALTIME is **DOUBLE**
2. Type of ABSTIME is **LONGLONG = int64**
3. **V** (voltage) is often referred to as the **RHS** vector. (Sitting as it does on the ***R**ight **H**and **S**ide* of the above equation).
4. Digital transient analysis is performed using a technique known as ***Event Driven Simulation***. This is different from the analogue transient analysis used by **SPICE** in that processing only occurs when some element of the circuit changes state.