Определение текстовых ресурсов на уровне компонента Блэкбокс

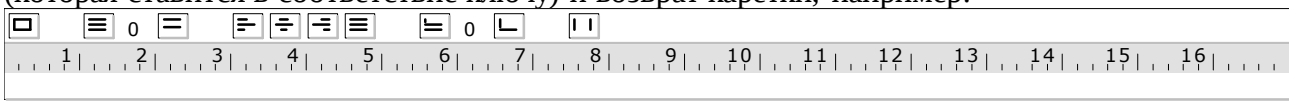
Темиргалеев Е. Э.
(05.09.2013)

Поддержка так называемых "текстовых ресурсов" обеспечивается библиотекой каркаса системы Блэкбокс для задач локализации компонентов [5, 1.4]. Напомним, что компоненты являются единицами упаковки и развёртывания, могут развёртываться независимо друг от друга, а собираться третьей стороной [7, 4.1]. «Минимальным компонентом Блэкбокс является модуль Компонентного Паскаля… *Подсистема* (*subsystem*) — коллекция взаимосвязаных (related) компонентов, размещённых в отдельных подпапках папки Блэкбокс…» [3, 1]

1. Текстовые ресурсы в системе Блэкбокс

«Текстовые ресурсы — это файлы, которые устанавливают соответствия между литерными цепочками (фактически словари для перевода — *прим. ред.*); например, цепочке "untitled" может быть поставлена в соответствие "sans titre". Это позволяет вынести сложные текстовые сообщения из программного кода и дать возможность менять эти сообщения без перекомпиляции.

… Файлы текстовых ресурсов могут быть обычными текстовыми документами Блэкбокса, которые состоят из ключевого слова STRINGS, за которым следуют строки текста, причем каждая строка включает в себя литерную цепочку (ключ), символ табуляции, другую цепочку (которая ставится в соответствие ключу) и возврат каретки, например:



```
STRINGS
untitled	sans titre
open	ouvre
close	ferme
```

» [1, 9]

Отображение цепочек (ключ -> ресурс) выполняют процедура *Dialog.MapParamString* и её упрощённый вариант *Dialog.MapString*:

```
PROCEDURE MapParamString (in, p0, p1, p2: ARRAY OF CHAR; OUT out: ARRAY OF CHAR);
PROCEDURE MapString (in: ARRAY OF CHAR; OUT out: ARRAY OF CHAR);
```

Документ с текстовыми ресурсами суть табличное определение этого отображения.

Поддерживается по одной таблице на каждую подсистему. Входная цепочка *in* вида "*#Subsystem:message*" отображается на результирующую *out* поиском по ключу *message* в таблице текстовых ресурсов подсистемы *Subsystem*. Если соответствие в таблице не задано, в качестве результата выдаётся значение ключа — "*message*". (Таблица вообще не задана — суть пуста.) [4, процедура *MapParamString*]

Механизм локализации заложен в схему определения (построения) таблиц. Основная

таблица (язык по-умолчанию) читается из текстового документа Блэкбокс в файле "Strings" из подпапки "Rsrc" папки "*Subsystem*", т. е. из файла ресурсов подсистемы *Subsystem* [3, 1]. Если система Блэкбокс настроена на конкретный язык (*Dialog.language # ""*), то при формировании таблицы текстовых ресурсов подсистемы на основную накладывается таблица локализации, читаемая из файла "Strings" подпапки "*Dialog.language*" папки "Rsrc". [4, процедура *SetLanguage*]

Например, по документам "System/Rsrc/Strings" и "System/Rsrc/ru/Strings"

STRINGS	STRINGS
SyntaxError syntax error	SyntaxError синтаксическая ошибка
SyntaxErrorIn syntax error in ^0	

при *Dialog.language* = "ru" будет построена следующая таблица "System":

"SyntaxError"	"синтаксическая ошибка"
"SyntaxErrorIn"	"syntax error in ^0"

по которой *Dialog.MapString* будет отображать:

"#System:SyntaxError"	-> "синтаксическая ошибка"
"#System:SyntaxErrorIn"	-> "syntax error in ^0"
"#System:QuoteExpected"	-> "QuoteExpected"

Такая подстановка «позволяет убрать из исходников литерные цепочки, специфичные для страны или языка, в то же время сохраняя там цепочку, используемую по умолчанию, так что программа будет работать, даже если будут утеряны файлы ресурсов» [2, процедура *MapParamString*].

2. Проблемы, порождаемые определением текстовых ресурсов на уровне подсистемы

Схема определения текстовых ресурсов на уровне подсистемы (один документ с таблицей для подсистемы) выглядит как оптимизация:

- взаимосвязанные компоненты могут использовать общие цепочки литер;
- меньшее число таблиц сокращает время поиска.

которая, однако, отрицательно сказывается на компонентности, усложняя программирование и развёртывание компонентов Блэкбокс.

В соглашениях системы Блэкбокс [3, 1] о файловой структуре, общий для всех компонентов подсистемы файл текстовых ресурсов выступает *единственной причиной* , из-за которой компонент нельзя выделить как "самодостаточный" набор файлов, копирование (удаление) которого равносильно добавлению (удалению) компонента — подобно тому, как это справедливо для (папок) подсистем [6, 3.2]: «It is only appropriate for component-oriented software that addition and removal of a component can be performed incrementally, by adding or removing a directory. All kinds of central installation or registration mechanisms which distribute the constituents of a component should be avoided, since they inevitably lead to (unnecessary) management problems.»

Означенные в цитате проблемы заключаются в необходимость править таблицу текстовых ресурсов в документе "*Subsystem*/Rsrc/Strings" при добавлении, удалении, обновлении компонентов подсистемы *Subsystem*. Очевидно, что создатели системы Блэкбокс, сами и обозначившие проблему, исходили из принципа "одна подсистема — один изготовитель". Однако, как показала практика, оный среди пользователей системы не прижился. Не редки подсистемы, просто содержащие набор по сути ничем не связанных компонентов. Например:

- «Cpc is a subsystem maintained on the Component Pascal Collection web site. It comprises individual modules that do not require a subsystem each, and that provide tools or utilities that may be of general use to Component Pascal users.» (http://www.zinnamturm.eu/downloadsAC.htm#Cpc, дата обращения: 27.08.2013);
- omc — аналогично Cpc;
- ypk, Stern — изделия вида "набор всего-всего" товарищей Кушнира П. М. и Кузьмицкого И. А.

Но даже в случае одного изготовителя, имеет место проблема правки текстовых ресурсов.

Трудность заключается в возлагаемой на человека задаче поиска строк таблицы, содержащих ключи, используемые конкретным компонентом (несколькими), возникающей при:
- переносе (переименовании) компонента из одной подсистемы в другую;
- удалении компонента;
- изменении структуры или сути текстового сообщения с сохранением неизменным ключа, могущей нарушить логику работы других компонентов, его использующих.

Оптимизационная, так сказать, тенденция, заложенная в схему, провоцирует человеческий фактор использовать "чужие" текстовые ресурсы — других компонентов подсистемы или вообще других подсистем. Она-то как раз и может привести к тому, что ресурс, внесённый в таблицу для одного компонента, задним числом может быть задействован для другого — в том числе и другим программистом. Учитывая к этому, что ключи могут вычисляться программно, отчётливо видим, что общее решение означенной в предыдущем абазце задачи автоматически (машиной) — не возможно.

3. Пути решения

Основных пути два — облегчить работу при существующей схеме (A) или изменить схему с целью устранить трудности (B). Последний подразумевает обеспечение каждого компонента независимым от других компонентов набором текстовых ресурсов. И представляется лучшим в сравнеии с (A), как более согласный с принципами «Keep all constituents of a component in one place.» [6, 3.2] и «Калашникова» [9].

Перечислим некоторые варианты решений:

0-B) Самое простое чисто организационное решение — размещать в подсистеме ровно по одному компоненту. Практически оно не представляемся приемлемым — будет слишком много папок (подсистем).

1-A) Простое решение, эксплуатирующееся для компонентов подсистемы *Cpc* из коллекции компонентов Cpc, заключается в комплектации компонентов вспомогательным для развёртывания компонентом *CpcInstall*. Его команды позволяют создать пустой документ "Cpc/Rsrc/Strings" (*CpcInstall.InitRsrc*) и дописать к нему цепочки компонента *Comp* из отдельного документа "Cpc/Xtras/*Comp*/Strings" (*CpcInstall.AppendToStrings*). Пример — содержимое архива *CpcUtf8Conv* [8]:

```
Cpc/Docu/Utf8Conv-Quick-Start
Cpc/Mod/Install
Cpc/Mod/Utf8Conv
Cpc/Xtras/Utf8Conv/Strings
```

2-A) Автором рассматривалась идея сборки документа "*Subsystem*/Rsrc/Strings" на лету из подобным (1) образом организованных — по-одному на компонент документов.

3-A) Простое организационное решение для облегчение поиска/правки ресурсов в документе "Strings" оформлением: вставка "заголовков"; использование в ключах приставок, связанных с именем компонента.

```
STRINGS
...
ert0devSpecViews/ert0devSpecViewsCmds
SV:InsertLineBreak          CB: вставить разрыв строки
SV:InsertExportMark         CB: вставить значок экспорта
...
```

4-B) Запрограммировать альтернативный *Dialog* сервис поддержки текстовых ресурсов, который обеспечит независимость ресурсов для своих клиентов. Своего рода золотая середина. С одной стороны — не требует обеспечить обратную совместимость, с другой — никак не затрагивает "старые" компоненты.

Пример решения по варианту (4) приведён далее.

4. *PrivLS* (модернизированный *Dialog*) — текстовые ресурсы для отдельных модулей

Для отдельных компонентов Блкэбокс (модулей КП) обеспечивает поддержку текстовых ресурсов аналогично *Dialog* (1):

PROCEDURE MapParamString (IN in, p0, p1, p2: ARRAY OF CHAR; OUT out: ARRAY OF CHAR);

PROCEDURE MapString (IN in: ARRAY OF CHAR; OUT out: ARRAY OF CHAR);

Формат цепочки *in*, отображаемой на ресурс *out*:

РБНФ = "#" ИдентификаторМодуля ":" Ключ.

Основная таблица модуля *SubMod* читается из документа в файле ресурса, имеющего суффикс "Strings", т. е. из файла "*Sub*/Rsrc/*Mod*Strings" [3, 1]. На неё накладывается таблица локализации из файла "*Sub*/Rsrc/*Dialog.language*/*Mod*Strings", если *Dialog.language* # "".

Для "внеподсистемного" модуля *Mod* (*Sub* = "") таблица строится по текстам "Rsrc/*Mod*Strings" и "Rsrc/*Dialog.language*/*Mod*Strings", или, при отсутствии оных, по "System/Rsrc/*Mod*Strings" и "System/Rsrc/*Dialog.language*/*Mod*Strings".

Список источников

1. Блэкбокс 1.5 — базовая сборка И-21 (версия 2012-11-09). Документ "/System/Docu/ru/User-Man.odc".
   URL: http://www.inr.ac.ru/~blackbox/rsrc/blackbox15i21base.7z (дата обращения 21.06.2013)
2. Там же. Документ "/System/Docu/ru/Dialog.odc".
3. Темиргалеев Е. Э. Разработочная подсистема — упорядочивание сопутствующих процессу разработки материалов.
   URL: http://oberoncore.ru/library/temir_razrabotochnaya_podsistema-uporyadochivanie_soputstvuyushhix_processu_razrabotki_materialov
4. BlackBox Component Builder 1.6-rc6 (Win). Документ "/System/Docu/Dialog.odc".
   URL: http://oberon.ch/zip/SetupBlackBox16-rc6.exe (дата обращения: 07.06.2013)
5. Там же. Документ "/Docu/Tut-1.odc".
6. Там же. Документ "/Docu/Tut-3.odc".
7. Пфистер К. Компонентное программное обеспечение // перевод Ермакова И. Е.
   URL: http://oberoncore.ru/library/component_soft
8. CpcUtf8Conv.txt (30.01.2013). Документ "/Cpc/Mod/Install.odc".
   URL: http://www.zinnamturm.eu/downloadsAC.htm#CpcUtf8Conv (дата обращения: 27.08.2013)
9. Ткачёв Ф. В. Принцип Калашникова.
   URL: http://www.inr.ac.ru/~info21/princypKalashnikova.htm (дата обращения: 19.06.2013)

Приложение 1. Исходный текст *PrivLS*

```
MODULE PrivLS;

   IMPORT SYSTEM, Files, Dialog;

   CONST
      rsrcDir = "Rsrc";
      stringSuffix = "Strings";
      TAB = 09X; CR = 0DX;

   TYPE
      StringPtr = POINTER TO ARRAY [untagged] OF CHAR;
      StringTab = POINTER TO RECORD
         next: StringTab;
         name: Files.Name;
         key: POINTER TO ARRAY OF StringPtr;
         str: POINTER TO ARRAY OF StringPtr;
         data: POINTER TO ARRAY OF CHAR
      END;

      Notifier = POINTER TO RECORD (Dialog.LangNotifier) END;

   VAR
      tabList: StringTab;
      notifier: Notifier;

   PROCEDURE ^ FlushMappings*;
   PROCEDURE (n: Notifier) Notify;
   BEGIN
      FlushMappings
   END Notify;

   PROCEDURE RegisterNotifier;
   BEGIN
      NEW(notifier); Dialog.RegisterLangNotifier(notifier)
   END RegisterNotifier;

   PROCEDURE RemoveNotifier;
   BEGIN
      Dialog.RemoveLangNotifier(notifier); notifier := NIL
   END RemoveNotifier;

   PROCEDURE ReadInt (in: Files.Reader; OUT result: INTEGER);
      VAR b: BYTE; x: INTEGER;
   BEGIN
      in.ReadByte(b); x := b MOD 256;
      in.ReadByte(b); x := x + (b MOD 256) * 100H;
      in.ReadByte(b); x := x + (b MOD 256) * 10000H;
      in.ReadByte(b); x := x + b * 1000000H;
      result := x
   END ReadInt;

   PROCEDURE ReadHead (in: Files.Reader; OUT next, down, end: INTEGER);
      VAR b, t: BYTE; n: INTEGER;
   BEGIN
      in.ReadByte(b);
      REPEAT
```

```
      in.ReadByte(t);
      IF t = -14 THEN
         ReadInt(in, n)
      ELSE
         REPEAT in.ReadByte(b) UNTIL b = 0
      END
   UNTIL t # -15;
   ReadInt(in, n);
   ReadInt(in, next); next := next + in.Pos();
   ReadInt(in, down); down := down + in.Pos();
   ReadInt(in, end); end := end + in.Pos()
END ReadHead;

PROCEDURE ReadStringFile (IN tabName: Files.Name; f: Files.File; OUT tab: StringTab);
   VAR
      i, j, h, n, s, x, len, next, down, end: INTEGER;
      in, in1: Files.Reader;
      ch: CHAR; b: BYTE; p, q: StringPtr;
BEGIN
   IF (f = NIL) OR (f.Length() <= 8) THEN
      in := NIL
   ELSE
      in := f.NewReader(NIL); in1 :=  f.NewReader(NIL)
   END;
   IF ~((in # NIL) & (in1 # NIL)) THEN
      tab := NIL
   ELSE (* read text file *)
      NEW(tab); tab.name := tabName$;
      NEW(tab.data, f.Length());
      in.SetPos(8); ReadHead(in, next, down, end);  (* document view *)
      in.SetPos(down); ReadHead(in, next, down, end); (* document model *)
      in.SetPos(down); ReadHead(in, next, down, end); (* text view *)
      in.SetPos(down); ReadHead(in, next, down, end); (* text model *)
      in.ReadByte(b); in.ReadByte(b); in.ReadByte(b);  (* versions *)
      in.ReadByte(b); in.ReadByte(b); in.ReadByte(b);
      ReadInt(in, x); in1.SetPos(in.Pos() + x);  (* text offset *)
      next := down;
      n := 0; i := 0; s := 0; in.ReadByte(b);
      WHILE b # -1 DO
         IF next = in.Pos() THEN  (* skip attributes *)
            ReadHead(in, next, down, end); in.SetPos(end)
         END;
         ReadInt(in, len);
         IF len > 0 THEN (* shortchar run *)
            WHILE len > 0 DO
               in1.ReadByte(b); ch := CHR(b MOD 256);
               IF ch >= " " THEN
                  IF s = 0 THEN j := i; s := 1 END;(* start of left part *)
                  tab.data[j] := ch; INC(j)
               ELSIF (s = 1) & (ch = TAB) THEN
                  tab.data[j] := 0X; INC(j);
                  s := 2   (* start of right part *)
               ELSIF (s = 2) & (ch = CR) THEN
                  tab.data[j] := 0X; INC(j);
                  INC(n); i := j; s := 0  (* end of line *)
               ELSE
                  s := 0   (* reset *)
               END;
               DEC(len)
```

```
            END
         ELSIF len < 0 THEN    (* longchar run *)
            WHILE len < 0 DO
               in1.ReadByte(b); x := b MOD 256; in1.ReadByte(b); ch := CHR(x + 256 * (b
+ 128));
               IF s = 0 THEN j := i; s := 1 END;(* start of left part *)
               tab.data[j] := ch; INC(j);
               INC(len, 2)
            END
         ELSE (* view *)
            ReadInt(in, x); ReadInt(in, x); in1.ReadByte(b);   (* ignore *)
         END;
         IF next = in.Pos() THEN  (* skip view data *)
            ReadHead(in, next, down, end); in.SetPos(end)
         END;
         in.ReadByte(b);
      END;
      IF n > 0 THEN
         NEW(tab.key, n); NEW(tab.str, n); i := 0; j := 0;
         WHILE j < n DO
            tab.key[j] := SYSTEM.VAL(StringPtr, SYSTEM.ADR(tab.data[i]));
            WHILE tab.data[i] >= " " DO INC(i) END;
            INC(i);
            tab.str[j] := SYSTEM.VAL(StringPtr, SYSTEM.ADR(tab.data[i]));
            WHILE tab.data[i] >= " " DO INC(i) END;
            INC(i); INC(j)
         END;
         (* sort keys (shellsort) *)
         h := 1; REPEAT h := h*3 + 1 UNTIL h > n;
         REPEAT h := h DIV 3; i := h;
            WHILE i < n DO p := tab.key[i]; q := tab.str[i]; j := i;
               WHILE (j >= h) & (tab.key[j-h]^ > p^) DO
                  tab.key[j] := tab.key[j-h]; tab.str[j] := tab.str[j-h]; j := j-h
               END;
               tab.key[j] := p; tab.str[j] := q; INC(i)
            END
         UNTIL h = 1
      END
   END
END ReadStringFile;

PROCEDURE MergeTabs (master, extra: StringTab): StringTab;
   VAR tab: StringTab; nofKeys, datalength, di, mi, ei, ml, el, ti, i: INTEGER; result:
StringTab;
BEGIN
   IF (extra = NIL) OR (extra.key = NIL) THEN
      result := master
   ELSIF (master = NIL) OR (master.key = NIL) THEN
      result := extra
   ELSE (* (master # NIL) & (master.key # NIL) & (extra # NIL) & (extra.key # NIL) *)
      ml := LEN(master.key); el := LEN(extra.key);
      mi := 0; ei := 0; datalength := 0; nofKeys := 0;
      (* find out how big the resulting table will be *)
      WHILE (mi < ml) OR (ei < el) DO
         INC(nofKeys);
         IF (mi < ml) & (ei < el) & (master.key[mi]$ = extra.key[ei]$) THEN
            datalength := datalength + LEN(master.key[mi]$) + LEN(master.str[mi]$) + 2;
            INC(mi); INC(ei)
         ELSIF (ei < el) & ((mi >= ml) OR (master.key[mi]$ > extra.key[ei]$)) THEN
```

```
            datalength := datalength + LEN(extra.key[ei]$) + LEN(extra.str[ei]$) + 2;
            INC(ei)
         ELSE
            datalength := datalength + LEN(master.key[mi]$) + LEN(master.str[mi]$) + 2;
            INC(mi)
         END
      END;
      NEW(tab); tab.name := master.name;
      NEW(tab.key, nofKeys); NEW(tab.str, nofKeys); NEW(tab.data, datalength);
      mi := 0; ei := 0; di := 0; ti := 0;
      (* do the merge *)
      WHILE (mi < ml) OR (ei < el) DO
         IF  (mi < ml) & (ei < el) & (master.key[mi]$ = extra.key[ei]$)  THEN
            i := 0; tab.key[ti] := SYSTEM.VAL(StringPtr, SYSTEM.ADR(tab.data[di]));
            WHILE  master.key[mi][i] # 0X DO tab.data[di] := master.key[mi][i]; INC(di);
INC(i) END;
            tab.data[di] :=0X; INC(di); i := 0;
            tab.str[ti] := SYSTEM.VAL(StringPtr, SYSTEM.ADR(tab.data[di]));
            WHILE master.str[mi][i] # 0X DO tab.data[di] := master.str[mi][i]; INC(di);
INC(i) END;
             tab.data[di] :=0X; INC(di);
            INC(mi); INC(ei)
         ELSIF (ei < el) & ((mi >= ml) OR (master.key[mi]$ > extra.key[ei]$)) THEN
            i := 0; tab.key[ti] := SYSTEM.VAL(StringPtr, SYSTEM.ADR(tab.data[di]));
            WHILE extra.key[ei][i] # 0X DO tab.data[di] := extra.key[ei][i]; INC(di); INC(i)
END;
            tab.data[di] :=0X; INC(di); i := 0;
            tab.str[ti] := SYSTEM.VAL(StringPtr, SYSTEM.ADR(tab.data[di]));
            WHILE  extra.str[ei][i] # 0X DO tab.data[di] := extra.str[ei][i]; INC(di); INC(i)
END;
             tab.data[di] :=0X; INC(di);
            INC(ei)
         ELSE
            i := 0; tab.key[ti] := SYSTEM.VAL(StringPtr, SYSTEM.ADR(tab.data[di]));
            WHILE master.key[mi][i] # 0X DO  tab.data[di] := master.key[mi][i]; INC(di);
INC(i) END;
            tab.data[di] :=0X; INC(di); i := 0;
            tab.str[ti] := SYSTEM.VAL(StringPtr, SYSTEM.ADR(tab.data[di]));
            WHILE master.str[mi][i] # 0X DO  tab.data[di] := master.str[mi][i]; INC(di);
INC(i) END;
             tab.data[di] :=0X; INC(di);
            INC(mi)
         END;
         INC(ti)
      END;
      result := tab
   END;
   RETURN result
END MergeTabs;

PROCEDURE Split (IN ident: Files.Name; OUT sub, mod: Files.Name);
   VAR  i, j: INTEGER; ch: CHAR;
BEGIN
   i := 0; ch := ident[0];
   WHILE (ch # 0X) & (("A" <= ch) & (ch <= "Z") OR ("A" <= ch) & (ch <= "Ю")) DO
      sub[i] := ch;
      INC(i); ch := ident[i]
   END;
   WHILE (ch # 0X) & ~(("A" <= ch) & (ch <= "Z") OR ("A" <= ch) & (ch <= "Ю")) DO
```

```
      sub[i] := ch;
      INC(i); ch := ident[i]
    END;
    sub[i] := 0X;
    IF ch = 0X THEN
      mod := sub; sub := ""
    ELSE
      j := 0;
      WHILE (ch # 0X) DO
        mod[j] := ch; INC(j);
        INC(i); ch := ident[i]
      END;
      mod[j] := 0X
    END
  END Split;

  PROCEDURE LoadStringTab (IN ident: Files.Name; OUT tab: StringTab);
    VAR loc: Files.Locator; f: Files.File; sub, mod, name: Files.Name; ltab: StringTab;
  BEGIN
    Split(ident, sub, mod);
    Files.dir.GetFileName(mod + stringSuffix, Files.docType, name);
    loc := Files.dir.This(sub); loc := loc.This(rsrcDir);
    IF loc = NIL THEN
      tab := NIL
    ELSE (* loc # NIL *)
      f := Files.dir.Old(loc, name, Files.shared);
      ReadStringFile(ident, f, tab);
      IF Dialog.language # "" THEN
        loc := loc.This(Dialog.language);
        IF loc # NIL THEN
          f := Files.dir.Old(loc, name, Files.shared);
          ReadStringFile(ident, f, ltab);
          tab := MergeTabs(ltab, tab)
        END
      END
    END;
    IF (tab = NIL) & (sub = "") THEN
      loc := Files.dir.This("System"); loc := loc.This(rsrcDir);
      IF loc = NIL THEN
        (* tab := NIL *)
      ELSE (* loc # NIL *)
        f := Files.dir.Old(loc, name, Files.shared);
        ReadStringFile(ident, f, tab);
        IF Dialog.language # "" THEN
          loc := loc.This(Dialog.language);
          IF loc # NIL THEN
            f := Files.dir.Old(loc, name, Files.shared);
            ReadStringFile(ident, f, ltab);
            tab := MergeTabs(ltab, tab)
          END
        END
      END
    END
  END LoadStringTab;

  PROCEDURE MapString1 (in: ARRAY OF CHAR; OUT out: ARRAY OF CHAR; OUT result:
INTEGER);
    VAR i, j, k, len: INTEGER; ch: CHAR; mod: Files.Name; tab: StringTab;
      code: INTEGER; ident: BOOLEAN;
```

```
BEGIN
   code := 0;
   IF ~(in[0] = "#") THEN
      code := 1  (* неверный синт. *)
   ELSE
      i := 0; ch := in[1];
      ident := ("A" <= ch) & (ch <= "Z") OR ("a" <= ch) & (ch <= "z")
         OR ("A" <= ch) & (ch <= 0FFX) & (ch # 0D7X) & (ch # 0F7X)
         OR (ch = "_");
      WHILE (ch # ":") & (ch # 0X) & ident DO
         mod[i] := ch;
         INC(i); ch := in[i + 1];
         ident := ("0" <= ch) & (ch <= "9") OR
            ("A" <= ch) & (ch <= "Z") OR ("a" <= ch) & (ch <= "z")
            OR ("A" <= ch) & (ch <= 0FFX) & (ch # 0D7X) & (ch # 0F7X)
            OR (ch = "_")
      END;
      mod[i] := 0X;
      IF ch = ":" THEN
         INC(i, 2); ch := in[i]; j := 0;
         WHILE (ch # 0X) DO in[j] := ch; INC(i); INC(j); ch := in[i] END;
         in[j] := 0X
         (* in = key *)
      ELSIF ch = 0X THEN
         code := 2  (* неверный синт. *)
      ELSIF ~ident THEN
         code := 3  (* неверный синт. *)
      ELSE
         HALT(100)
      END
   END;
   IF code = 0 THEN
      tab := tabList;
      WHILE (tab # NIL) & (tab.name # mod) DO tab := tab.next END;
      IF tab = NIL THEN
         LoadStringTab(mod, tab);
         IF tab # NIL THEN tab.next := tabList; tabList := tab END
      END;
      IF tab = NIL THEN code := 4 END    (* таблица отсутствует  и не загрузилась *)
   END;
   IF code = 0 THEN
      i := 0;
      IF tab.key = NIL THEN j := 0 ELSE j := LEN(tab.key^) END;
      WHILE i < j DO  (* binary search *)
         k := (i + j) DIV 2;
         IF tab.key[k]^ < in THEN i := k + 1 ELSE j := k END
      END;
      IF (tab.key # NIL) & (j < LEN(tab.key^)) & (tab.key[j]^ = in) THEN
         k := 0; len := LEN(out)-1;
         WHILE (k < len) & (tab.str[j][k] # 0X) DO
            out[k] := tab.str[j][k]; INC(k)
         END;
         out[k] := 0X
      ELSE
         code := 5  (* нет в таблице *)
      END
   END;
   IF code = 0 THEN
      (* - *)
```

```
      ELSE
         (* out := in$ *)
         i := 0; ch := in[0]; len := LEN(out) - 1;
         WHILE (i < len) & (ch # 0X) DO
            out[i] := ch;
            INC(i); ch := in[i]
         END;
         out[i] := 0X
      END;
      result := code
   END MapString1;

   PROCEDURE Subst (in: ARRAY OF CHAR; IN p0, p1, p2: ARRAY OF CHAR; VAR out: ARRAY
OF CHAR);
      VAR len, i, j, k: INTEGER; ch, c: CHAR;
   BEGIN
      i := 0; ch := in[i]; j := 0; len := LEN(out) - 1;
      WHILE (ch # 0X) & (j < len) DO
         IF ch = "^" THEN
            INC(i); ch := in[i];
            IF ch = "0" THEN
               k := 0; c := p0[0];
               WHILE (c # 0X) & (j < len) DO out[j] := c; INC(j); INC(k); c := p0[k] END;
               INC(i); ch := in[i]
            ELSIF ch = "1" THEN
               k := 0; c := p1[0];
               WHILE (c # 0X) & (j < len) DO out[j] := c; INC(j); INC(k); c := p1[k] END;
               INC(i); ch := in[i]
            ELSIF ch = "2" THEN
               k := 0; c := p2[0];
               WHILE (c # 0X) & (j < len) DO out[j] := c; INC(j); INC(k); c := p2[k] END;
               INC(i); ch := in[i]
            ELSE out[j] := "^"; INC(j)
            END
         ELSE out[j] := ch; INC(j); INC(i); ch := in[i]
         END
      END;
      out[j] := 0X
   END Subst;

   PROCEDURE FlushMappings*;
   BEGIN
      tabList := NIL
   END FlushMappings;

   PROCEDURE MapParamString* (IN in, p0, p1, p2: ARRAY OF CHAR; OUT out: ARRAY OF
CHAR);
      VAR  res: INTEGER;
   BEGIN
      MapString1(in, out, res);
      Subst(out, p0, p1, p2, out)
   END MapParamString;

   PROCEDURE MapString* (IN in: ARRAY OF CHAR; OUT out: ARRAY OF CHAR);
      VAR  res: INTEGER;
   BEGIN
      MapString1(in, out, res)
   END MapString;
```

```
BEGIN
    RegisterNotifier
CLOSE
    RemoveNotifier
END PrivLS.
```

Приложение 2. Исходный текст *PrivTV* (модернизированный *DevDebug*) — обработчик трапов, использующий "новые" ресурсы

```
MODULE PrivTV;

   IMPORT SYSTEM,
      Kernel, Strings, Files, Fonts, Services, Ports, Stores, Converters,
      Models, Views, Controllers, Properties, Dialog, Containers,
      Windows, StdDialog, StdFolds, StdLinks,
      TextModels, TextMappers, TextControllers, TextViews, TextRulers, StdLog,
      D := DevDebug, LS := PrivLS;

   CONST
      refViewSize = 9 * Ports.point;

      heap = 1; source = 2; module = 3; (* RefView types *)
      open = 1; undo = 2; update = 3;(* RefView commands *)

      (* additional scanner types *)
      import = 100; smodule = 101; semicolon = 102; becomes  = 103; stop = 104; comEnd
= 105;

   TYPE
      Name = Kernel.Name;
      ArrayPtr = POINTER TO EXTENSIBLE RECORD
         last, t, first: INTEGER;    (* gc header *)
         len: ARRAY 16 OF INTEGER (* dynamic array length table *)
      END;
      RefView = POINTER TO RECORD (Views.View)
         type: SHORTINT;
         command: SHORTINT;
         back: RefView;
         adr: INTEGER;
         desc: Kernel.Type;
         ptr: ArrayPtr;
         name: Name
      END;

      Action = POINTER TO RECORD (Services.Action)
         text: TextModels.Model
      END;

      Cluster = POINTER TO RECORD [untagged]  (* must correspond to Kernel.Cluster *)
         size: INTEGER;
         next: Cluster
      END;


   VAR
      out: TextMappers.Formatter;
      path: ARRAY 4 OF Ports.Point;
      empty: Name;


   PROCEDURE NewRuler (): TextRulers.Ruler;
      CONST mm = Ports.mm;
      VAR r: TextRulers.Ruler;
   BEGIN
      r := TextRulers.dir.New(NIL);
```

```
      TextRulers.SetRight(r, 140 * mm);
      TextRulers.AddTab(r, 4 * mm); TextRulers.AddTab(r, 34 * mm); TextRulers.AddTab(r, 80 *
mm);
      RETURN r
   END NewRuler;

   PROCEDURE OpenViewer (t: TextModels.Model; IN titl: ARRAY OF CHAR;
ruler:TextRulers.Ruler);
      VAR v: TextViews.View; c: Containers.Controller; title: Views.Title;
   BEGIN
      LS.MapString(titl, title);
      v := TextViews.dir.New(t);
      v.SetDefaults(ruler, TextViews.dir.defAttr);
      c := v.ThisController();
      IF c # NIL THEN
         c.SetOpts(c.opts - {Containers.noFocus, Containers.noSelection} +
{Containers.noCaret})
      END;
      Views.OpenAux(v, title)
   END OpenViewer;

   PROCEDURE OpenFold (IN hidden: ARRAY OF CHAR);
      VAR fold: StdFolds.Fold; t: TextModels.Model; w: TextMappers.Formatter; s: ARRAY 256
OF CHAR;
   BEGIN
      LS.MapString(hidden, s);
      t := TextModels.CloneOf(StdLog.buf);
      w.ConnectTo(t); w.WriteString(s);
      fold := StdFolds.dir.New(StdFolds.expanded, "", t);
      out.WriteView(fold)
   END OpenFold;

   PROCEDURE CloseFold (collaps: BOOLEAN);
      VAR fold: StdFolds.Fold; m: TextModels.Model;
   BEGIN
      fold := StdFolds.dir.New(StdFolds.expanded, "", NIL);
      out.WriteView(fold);
      IF collaps THEN fold.Flip(); m := out.rider.Base(); out.SetPos(m.Length()) END
   END CloseFold;

   PROCEDURE WriteHex (n: INTEGER);
   BEGIN
      out.WriteIntForm(n, TextMappers.hexadecimal, 9, "0", TextMappers.showBase)
   END WriteHex;

   PROCEDURE WriteString (adr, len, base: INTEGER; zterm, unicode: BOOLEAN);
      CONST beg = 0; char = 1; code = 2;
      VAR ch: CHAR; sc: SHORTCHAR; val, mode: INTEGER; str: ARRAY 16 OF CHAR;
   BEGIN
      mode := beg;
      IF base = 2 THEN SYSTEM.GET(adr, ch); val := ORD(ch) ELSE SYSTEM.GET(adr, sc);
val := ORD(sc) END;
      IF zterm & (val = 0) THEN out.WriteSString('"')
      ELSE
         REPEAT
            IF (val >= ORD(" ")) & (val < 7FH) OR (val > 0A0H) & (val < 100H) OR unicode &
(val >= 100H) THEN
               IF mode # char THEN
                  IF mode = code THEN out.WriteSString(", ") END;
```

```
                  out.WriteChar(22X); mode := char
               END;
               out.WriteChar(CHR(val))
            ELSE
               IF mode = char THEN out.WriteChar(22X) END;
               IF mode # beg THEN out.WriteSString(", ") END;
               mode := code; Strings.IntToStringForm(val, Strings.hexadecimal, 1, "0", FALSE,
str);
               IF str[0] > "9" THEN out.WriteChar("0") END;
               out.WriteString(str); out.WriteChar("X")
            END;
            INC(adr, base); DEC(len);
            IF base = 2 THEN SYSTEM.GET(adr, ch); val := ORD(ch) ELSE SYSTEM.GET(adr,
sc); val := ORD(sc) END
         UNTIL (len = 0) OR zterm & (val = 0)
      END;
      IF mode = char THEN out.WriteChar(22X) END
   END WriteString;

   PROCEDURE OutString (IN s: ARRAY OF CHAR);
      VAR str: ARRAY 256 OF CHAR;
   BEGIN
      LS.MapString(s, str);
      out.WriteString(str)
   END OutString;

   PROCEDURE ShowParamMsg (IN msg, p0, p1, p2: ARRAY OF CHAR);
      VAR  s: ARRAY 1024 OF CHAR;
   BEGIN
      LS.MapParamString(msg, p0, p1, p2, s);
      Dialog.ShowMsg(s)
   END ShowParamMsg;

   PROCEDURE ShowStatus (IN msg: ARRAY OF CHAR);
      VAR  s: ARRAY 1024 OF CHAR;
   BEGIN
      LS.MapString(msg, s);
      Dialog.ShowStatus(s)
   END ShowStatus;

   (* ------------------  variable display ------------------ *)

   PROCEDURE FormOf (t: Kernel.Type): SHORTCHAR;
   BEGIN
      IF SYSTEM.VAL(INTEGER, t) DIV 256 = 0 THEN
         RETURN SHORT(CHR(SYSTEM.VAL(INTEGER, t)))
      ELSE
         RETURN SHORT(CHR(16 + t.id MOD 4))
      END
   END FormOf;

   PROCEDURE LenOf (t: Kernel.Type; ptr: ArrayPtr): INTEGER;
   BEGIN
      IF t.size # 0 THEN RETURN t.size
      ELSIF ptr # NIL THEN RETURN ptr.len[t.id DIV 16 MOD 16 - 1]
      ELSE RETURN 0
      END
   END LenOf;
```

```
   PROCEDURE SizeOf (t: Kernel.Type; ptr: ArrayPtr): INTEGER;
   BEGIN
      CASE FormOf(t) OF
      | 0BX: RETURN 0
      | 1X, 2X, 4X: RETURN 1
      | 3X, 5X: RETURN 2
      | 8X, 0AX: RETURN 8
      | 11X: RETURN t.size
      | 12X: RETURN LenOf(t, ptr) * SizeOf(t.base[0], ptr)
      ELSE RETURN 4
      END
   END SizeOf;

   PROCEDURE WriteName (t: Kernel.Type; ptr: ArrayPtr);
      VAR name: Kernel.Name; f: SHORTCHAR;
   BEGIN
      f := FormOf(t);
      CASE f OF
      | 0X: OutString("#PrivTV:Unknown")
      | 1X: out.WriteSString("BOOLEAN")
      | 2X: out.WriteSString("SHORTCHAR")
      | 3X: out.WriteSString("CHAR")
      | 4X: out.WriteSString("BYTE")
      | 5X: out.WriteSString("SHORTINT")
      | 6X: out.WriteSString("INTEGER")
      | 7X: out.WriteSString("SHORTREAL")
      | 8X: out.WriteSString("REAL")
      | 9X: out.WriteSString("SET")
      | 0AX: out.WriteSString("LONGINT")
      | 0BX: out.WriteSString("ANYREC")
      | 0CX: out.WriteSString("ANYPTR")
      | 0DX: out.WriteSString("POINTER")
      | 0EX: out.WriteSString("PROCEDURE")
      | 0FX: out.WriteSString("STRING")
      | 10X..13X:
         IF (t.mod # NIL) & (t.id DIV 256 # 0) & (t.mod.refcnt >= 0) THEN
            Kernel.GetTypeName(t, name);
            IF name = "!" THEN
               IF f = 11X THEN out.WriteSString("RECORD")
               ELSIF f = 12X THEN out.WriteSString("ARRAY")
               ELSE OutString("#PrivTV:Unknown")
               END
            ELSE
               out.WriteSString(t.mod.name); out.WriteChar("."); out.WriteSString(name)
            END
         ELSIF f = 11X THEN
            IF t.mod # NIL THEN out.WriteSString(t.mod.name); out.WriteChar(".") END;
            out.WriteSString("RECORD");
         ELSIF f = 12X THEN
            out.WriteSString("ARRAY "); out.WriteInt(LenOf(t, ptr)); t := t.base[0];
            WHILE (FormOf(t) = 12X) & ((t.id DIV 256 = 0) OR (t.mod = NIL) OR (t.mod.refcnt
< 0)) DO
               out.WriteSString(", "); out.WriteInt(LenOf(t, ptr)); t := t.base[0]
            END;
            out.WriteSString(" OF "); WriteName(t, ptr)
         ELSIF f = 13X THEN
            out.WriteSString("POINTER")
         ELSE
            out.WriteSString("PROCEDURE")
```

```
      END
    | 20X: out.WriteSString("COM.IUnknown")
    | 21X: out.WriteSString("COM.GUID")
    | 22X: out.WriteSString("COM.RESULT")
    ELSE OutString("#PrivTV:UnknownFormat"); out.WriteInt(ORD(f))
    END
  END WriteName;

  PROCEDURE WriteGuid (a: INTEGER);

    PROCEDURE Hex (a: INTEGER);
      VAR x: SHORTCHAR;
    BEGIN
      SYSTEM.GET(a, x);
      out.WriteIntForm(ORD(x), TextMappers.hexadecimal, 2, "0", FALSE)
    END Hex;

  BEGIN
    out.WriteChar("{");
    Hex(a + 3); Hex(a + 2); Hex(a + 1); Hex(a);
    out.WriteChar("-");
    Hex(a + 5); Hex(a + 4);
    out.WriteChar("-");
    Hex(a + 7); Hex(a + 6);
    out.WriteChar("-");
    Hex(a + 8);
    Hex(a + 9);
    out.WriteChar("-");
    Hex(a + 10);
    Hex(a + 11);
    Hex(a + 12);
    Hex(a + 13);
    Hex(a + 14);
    Hex(a + 15);
    out.WriteChar("}")
  END WriteGuid;

  PROCEDURE^ ShowVar (
    ad, ind: INTEGER; f, c: SHORTCHAR; desc: Kernel.Type; ptr: ArrayPtr; back: RefView;
VAR name, sel: Name);

  PROCEDURE^ NewRefView (type, command: SHORTINT; adr: INTEGER; back: RefView;
                         desc: Kernel.Type; ptr: ArrayPtr; name: Name): RefView;

  PROCEDURE^ InsertRefView (type, command: SHORTINT; adr: INTEGER; back: RefView;
                         desc: Kernel.Type; ptr: ArrayPtr; name: Name);

  PROCEDURE ShowRecord (a, ind: INTEGER; desc: Kernel.Type; back: RefView; VAR sel:
Name);
    VAR dir: Kernel.Directory; obj: Kernel.Object; name: Kernel.Name; i, j, n: INTEGER;
base: Kernel.Type;
  BEGIN
    WriteName(desc, NIL); out.WriteTab;
    IF desc.mod.refcnt >= 0 THEN
      OpenFold("#PrivTV:Fields");
      n := desc.id DIV 16 MOD 16; j := 0;
      WHILE j <= n DO
        base := desc.base[j];
        IF base # NIL THEN
```

```
            dir := base.fields; i := 0;
            WHILE i < dir.num DO
               obj := SYSTEM.VAL(Kernel.Object, SYSTEM.ADR(dir.obj[i]));
               Kernel.GetObjName(base.mod, obj, name);
               ShowVar(a + obj.offs, ind, FormOf(obj.struct), 1X, obj.struct, NIL, back,
name, sel);
               INC(i)
            END
         END;
         INC(j)
      END;
      out.WriteSString("   "); CloseFold((ind > 1) OR (sel # ""))
   ELSE
      OutString("#PrivTV:Unloaded")
   END
END ShowRecord;

PROCEDURE ShowArray (a, ind: INTEGER; desc: Kernel.Type; ptr: ArrayPtr; back: RefView;
VAR sel: Name);
   VAR f: SHORTCHAR; i, n, m, size, len: INTEGER; name: Kernel.Name; eltyp, t:
Kernel.Type;
      vi: SHORTINT; vs: BYTE; str: Dialog.String; high: BOOLEAN;
BEGIN
   WriteName(desc, ptr); out.WriteTab;
   len := LenOf(desc, ptr); eltyp := desc.base[0]; f := FormOf(eltyp); size := SizeOf(eltyp,
ptr);
   IF (f = 2X) OR (f = 3X) THEN  (* string *)
      n := 0; m := len; high := FALSE;
      IF f = 2X THEN
         REPEAT SYSTEM.GET(a + n, vs); INC(n) UNTIL (n = 32) OR (n = len) OR (vs = 0);
         REPEAT DEC(m); SYSTEM.GET(a + m, vs) UNTIL (m = 0) OR (vs # 0)
      ELSE
         REPEAT
            SYSTEM.GET(a + n * 2, vi); INC(n);
            IF vi DIV 256 # 0 THEN high := TRUE END
         UNTIL (n = len) OR (vi = 0);
         n := MIN(n, 32);
         REPEAT DEC(m); SYSTEM.GET(a + m * 2, vi) UNTIL (m = 0) OR (vi # 0)
      END;
      WriteString(a, n, size, TRUE, TRUE);
      INC(m, 2);
      IF m > len THEN m := len END;
      IF high OR (m > n) THEN
         out.WriteSString("   "); OpenFold("...");
         out.WriteLn;
         IF high & (n = 32) THEN
            WriteString(a, m, size, TRUE, TRUE);
            out.WriteLn; out.WriteLn
         END;
         WriteString(a, m, size, FALSE, FALSE);
         IF m < len THEN out.WriteSString(", ..., 0X") END;
         out.WriteSString("   "); CloseFold(TRUE)
      END
   ELSE
      t := eltyp;
      WHILE FormOf(t) = 12X DO t := t.base[0] END;
      IF FormOf(t) # 0X THEN
         OpenFold("#PrivTV:Elements");
         i := 0;
```

```
            WHILE i < len DO
               Strings.IntToString(i, str);
               name := "[" + SHORT(str$) + "]";
               ShowVar(a, ind, f, 1X, eltyp, ptr, back, name, sel);
               INC(i); INC(a, size)
            END;
            out.WriteSString("   "); CloseFold(TRUE)
         END
      END
   END ShowArray;

   PROCEDURE ShowProcVar (a: INTEGER);
      VAR vli, n, ref: INTEGER; m: Kernel.Module; name: Kernel.Name;
   BEGIN
      SYSTEM.GET(a, vli);
      Kernel.SearchProcVar(vli, m, vli);
      IF m = NIL THEN
         IF vli = 0 THEN out.WriteSString("NIL")
         ELSE WriteHex(vli)
         END
      ELSE
         IF m.refcnt >= 0 THEN
            out.WriteSString(m.name); ref := m.refs;
            REPEAT Kernel.GetRefProc(ref, n, name) UNTIL (n = 0) OR (vli < n);
            IF vli < n THEN out.WriteChar("."); out.WriteSString(name) END
         ELSE
            OutString("#PrivTV:ProcInUnloadedMod");
            out.WriteSString(m.name); out.WriteSString(" !!!")
         END
      END
   END ShowProcVar;

   PROCEDURE ShowPointer (a: INTEGER; f: SHORTCHAR; desc: Kernel.Type; back: RefView;
VAR sel: Name);
      VAR adr, x: INTEGER; ptr: ArrayPtr; c: Cluster; btyp: Kernel.Type;
   BEGIN
      SYSTEM.GET(a, adr);
      IF f = 13X THEN btyp := desc.base[0] ELSE btyp := NIL END;
      IF adr = 0 THEN out.WriteSString("NIL")
      ELSIF f = 20X THEN
         out.WriteChar("["); WriteHex(adr); out.WriteChar("]");
         out.WriteChar(" "); c := SYSTEM.VAL(Cluster, Kernel.Root());
         WHILE (c # NIL) & ((adr < SYSTEM.VAL(INTEGER, c)) OR (adr >=
SYSTEM.VAL(INTEGER, c) + c.size)) DO c := c.next END;
         IF c # NIL THEN
            ptr := SYSTEM.VAL(ArrayPtr, adr);
            InsertRefView(heap, open, adr, back, btyp, ptr, sel)
         END
      ELSE
         IF (f = 13X) OR (f = 0CX) THEN x := adr - 4 ELSE x := adr END;
         IF ((adr < -4) OR (adr >= 65536)) & Kernel.IsReadable(x, adr + 16) THEN
            out.WriteChar("["); WriteHex(adr); out.WriteChar("]");
            IF (f = 13X) OR (f = 0CX) THEN
               out.WriteChar(" "); c := SYSTEM.VAL(Cluster, Kernel.Root());
               WHILE (c # NIL) & ((adr < SYSTEM.VAL(INTEGER, c)) OR (adr >=
SYSTEM.VAL(INTEGER, c) + c.size)) DO
                  c := c.next
               END;
               IF c # NIL THEN
```

```
                ptr := SYSTEM.VAL(ArrayPtr, adr);
                IF (f = 13X) & (FormOf(btyp) = 12X) THEN  (* array *)
                   adr := SYSTEM.ADR(ptr.len[btyp.id DIV 16 MOD 16])
                END;
                InsertRefView(heap, open, adr, back, btyp, ptr, sel)
             ELSE OutString("#PrivTV:IllegalPointer");
             END
          END
       ELSE OutString("#PrivTV:IllegalAddress"); WriteHex(adr)
       END
    END
  END ShowPointer;

  PROCEDURE ShowSelector (ref: RefView);
    VAR b: RefView; n: SHORTINT; a, a0: TextModels.Attributes;
  BEGIN
    b := ref.back; n := 1;
    IF b # NIL THEN
       WHILE (b.name = ref.name) & (b.back # NIL) DO INC(n); b := b.back END;
       ShowSelector(b);
       IF n > 1 THEN out.WriteChar("(") END;
       out.WriteChar(".")
    END;
    out.WriteSString(ref.name);
    IF ref.type = heap THEN out.WriteChar("^") END;
    IF n > 1 THEN
       out.WriteChar(")");
       a0 := out.rider.attr; a := TextModels.NewOffset(a0, 2 * Ports.point);
       out.rider.SetAttr(a);
       out.WriteInt(n); out.rider.SetAttr(a0)
    END;
  END ShowSelector;

  PROCEDURE ShowVar (
    ad, ind: INTEGER; f, c: SHORTCHAR; desc: Kernel.Type; ptr: ArrayPtr; back: RefView;
VAR name, sel: Name
  );
    VAR i, j, vli, a: INTEGER; tsel: Name; a0: TextModels.Attributes;
       vc: SHORTCHAR; vsi: BYTE; vi: SHORTINT; vr: SHORTREAL; vlr: REAL; vs: SET;
  BEGIN
    out.WriteLn; out.WriteTab; i := 0;
    WHILE i < ind DO out.WriteSString("  "); INC(i) END;
    a := ad; i := 0; j := 0;
    IF sel # "" THEN
       WHILE sel[i] # 0X DO tsel[i] := sel[i]; INC(i) END;
       IF (tsel[i-1] # ":") & (name[0] # "[") THEN tsel[i] := "."; INC(i) END
    END;
    WHILE name[j] # 0X DO tsel[i] := name[j]; INC(i); INC(j) END;
    tsel[i] := 0X;
    a0 := out.rider.attr;
    IF c = 3X THEN  (* varpar *)
       SYSTEM.GET(ad, a);
       out.rider.SetAttr(TextModels.NewStyle(a0, {Fonts.italic}))
    END;
    IF name[0] # "[" THEN out.WriteChar(".") END;
    out.WriteSString(name);
    out.rider.SetAttr(a0); out.WriteTab;
    IF (c = 3X) & (a >= 0) & (a < 65536) THEN
       out.WriteTab; out.WriteSString("NIL VARPAR");
```

```
      ELSIF f = 11X THEN
         Kernel.GetTypeName(desc, name);
         IF (c = 3X) & (name[0] # "!") THEN SYSTEM.GET(ad + 4, desc) END;(* dynamic type
*)
         ShowRecord(a, ind + 1, desc, back, tsel)
      ELSIF (c = 3X) & (f = 0BX) THEN(* VAR anyrecord *)
         SYSTEM.GET(ad + 4, desc);
         ShowRecord(a, ind + 1, desc, back, tsel)
      ELSIF f = 12X THEN
         IF (desc.size = 0) & (ptr = NIL) THEN SYSTEM.GET(ad, a) END; (* dyn array val par
*)
         IF ptr = NIL THEN ptr := SYSTEM.VAL(ArrayPtr, ad - 8) END;
         ShowArray(a, ind + 1, desc, ptr, back, tsel)
      ELSE
         IF desc = NIL THEN desc := SYSTEM.VAL(Kernel.Type, ORD(f)) END;
         WriteName(desc, NIL); out.WriteTab;
         CASE f OF
         | 0X: (* SYSTEM.GET(a, vli); WriteHex(vli) *)
         | 1X: SYSTEM.GET(a, vc);
            IF vc = 0X THEN out.WriteSString("FALSE")
            ELSIF vc = 1X THEN out.WriteSString("TRUE")
            ELSE OutString("#PrivTV:Undefined"); out.WriteInt(ORD(vc))
            END
         | 2X: WriteString(a, 1, 1, FALSE, FALSE)
         | 3X: WriteString(a, 1, 2, FALSE, TRUE);
               SYSTEM.GET(a, vi);
               IF vi DIV 256 # 0 THEN out.WriteString("  "); WriteString(a, 1, 2, FALSE, FALSE)
END
         | 4X: SYSTEM.GET(a, vsi); out.WriteInt(vsi)
         | 5X: SYSTEM.GET(a, vi); out.WriteInt(vi)
         | 6X: SYSTEM.GET(a, vli); out.WriteInt(vli)
         | 7X: SYSTEM.GET(a, vli);
               IF BITS(vli) * {23..30} = {23..30} THEN
                  IF BITS(vli) = {23..30} THEN out.WriteString("inf")
                  ELSIF BITS(vli) = {23..31} THEN out.WriteString("-inf")
                  ELSE out.WriteString("nan  ("); WriteHex(vli); out.WriteString(")")
                  END
               ELSE
                  SYSTEM.GET(a, vr); out.WriteReal(vr)
               END
         | 8X: IF Kernel.littleEndian THEN SYSTEM.GET(a, vli); SYSTEM.GET(a + 4, i)
               ELSE SYSTEM.GET(a + 4, vli); SYSTEM.GET(a, i)
               END;
               IF BITS(i) * {20..30} = {20..30} THEN
                  IF (BITS(i) = {20..30}) & (vli = 0) THEN out.WriteString("inf")
                  ELSIF (BITS(i) = {20..31}) & (vli = 0) THEN out.WriteString("-inf")
                  ELSE out.WriteString("nan  ("); out.WriteIntForm(i, TextMappers.hexadecimal,
8, "0", TextMappers.hideBase);
                     WriteHex(vli); out.WriteString(")")
                  END
               ELSE
                  SYSTEM.GET(a, vlr); out.WriteReal(vlr)
               END
         | 9X: SYSTEM.GET(a, vs); out.WriteSet(vs)
         | 0AX: IF Kernel.littleEndian THEN SYSTEM.GET(a, vli); SYSTEM.GET(a + 4, i)
               ELSE SYSTEM.GET(a + 4, vli); SYSTEM.GET(a, i)
               END;
               IF (vli >= 0) & (i = 0) OR (vli < 0) & (i = -1) THEN out.WriteInt(vli)
               ELSE out.WriteIntForm(i, TextMappers.hexadecimal, 8, "0",
```

```
TextMappers.hideBase); WriteHex(vli)
                END
          | 0CX, 0DX, 13X, 20X: ShowPointer(a, f, desc, back, tsel)
          | 0EX, 10X: ShowProcVar(a)
          | 0FX: WriteString(a, 256, 1, TRUE, FALSE)
          | 21X: WriteGuid(a)
          | 22X: SYSTEM.GET(a, vli); WriteHex(vli)
          ELSE
          END
       END
    END ShowVar;

    PROCEDURE ShowGlobals (mod: Kernel.Module);
       VAR ref, x: INTEGER; m, f: SHORTCHAR; name: ARRAY 256 OF CHAR; mname:
Kernel.Name;
          d: Kernel.Type; v: RefView; a0: TextModels.Attributes;
    BEGIN
       IF mod # NIL THEN
          out.WriteSString(mod.name);
          out.WriteTab; out.WriteTab; out.WriteTab;
          a0 := out.rider.attr;
          out.rider.SetAttr(TextModels.NewStyle(out.rider.attr, {Fonts.underline}));
          out.rider.SetAttr(TextModels.NewColor(out.rider.attr, Ports.blue));
          name := "PrivTV.UpdateGlobals('" + mod.name + "')";
          out.WriteView(StdLinks.dir.NewLink(name));
          OutString("#PrivTV:Update");
          out.WriteView(StdLinks.dir.NewLink(""));
          out.rider.SetAttr(a0); out.WriteLn;
          ref := mod.refs; Kernel.GetRefProc(ref, x, mname);  (* get body *)
          IF x # 0 THEN
             v := NewRefView (module, open, 0, NIL, NIL, NIL, mod.name);
             Kernel.GetRefVar(ref, m, f, d, x, mname);
             WHILE m = 1X DO
                ShowVar(mod.data + x, 0, f, m, d, NIL, v, mname, empty);
                Kernel.GetRefVar(ref, m, f, d, x, mname)
             END
          END;
          out.WriteLn
       END
    END ShowGlobals;

    PROCEDURE UpdateGlobals* (name: ARRAY OF CHAR);
       VAR t, t0: TextModels.Model; script: Stores.Operation; opName: Stores.OpName; mod:
Kernel.Module; n: Kernel.Name;
    BEGIN
       n := SHORT(name$); mod := Kernel.ThisLoadedMod(n);
       IF mod # NIL THEN
          t0 := TextViews.FocusText();
          LS.MapString("#PrivTV:Change", opName);
          Models.BeginScript(t0, opName, script);
          t := TextModels.CloneOf(t0);
          out.ConnectTo(t);
          ShowGlobals(mod);
          (*Stores.InitDomain(t, t0.domain);*) Stores.Join(t, t0);    (* not efficient to init
domain before writing *)
          t0.Delete(0, t0.Length()); t0.Insert(0, t, 0, t.Length());
          Models.EndScript(t0, script);
          out.ConnectTo(NIL)
       END
```

```
   END UpdateGlobals;

   PROCEDURE ShowObject (adr: INTEGER);
      VAR eltyp: Kernel.Type; ptr: ArrayPtr; desc: ARRAY 64 OF INTEGER; i, n, lev, elsize:
INTEGER;
   BEGIN
      SYSTEM.GET(adr - 4, eltyp);
      IF ODD(SYSTEM.VAL(INTEGER, eltyp) DIV 2) THEN
         DEC(SYSTEM.VAL(INTEGER, eltyp), 2);
         ptr := SYSTEM.VAL(ArrayPtr, adr);
         elsize := eltyp.size;
         IF (eltyp.mod.name = "Kernel") & (eltyp.fields.num = 1) THEN
            eltyp := eltyp.fields.obj[0].struct
         END;
         n := (ptr.last - ptr.first) DIV elsize + 1;
         lev := (ptr.first - adr - 12) DIV 4; i := 0;
         WHILE lev > 0 DO  (* dynamic levels *)
            DEC(lev);
            desc[i] := ptr.len[lev]; n := n DIV desc[i]; INC(i); (* size *)
            desc[i] := 0; INC(i);   (* module *)
            desc[i] := 2; INC(i);   (* id *)
            desc[i] := SYSTEM.ADR(desc[i+1]); INC(i)   (* desc *)
         END;
         IF n > 1 THEN    (* static level *)
            desc[i] := n; INC(i);   (* size *)
            desc[i] := 0; INC(i);   (* module *)
            desc[i] := 2; INC(i);   (* id *)
         ELSE DEC(i)
         END;
         desc[i] := SYSTEM.VAL(INTEGER, eltyp); (* desc *)
         ShowArray(ptr.first, 1, SYSTEM.VAL(Kernel.Type, SYSTEM.ADR(desc)), ptr, NIL,
empty);
         out.WriteLn
      ELSE ShowRecord(adr, 1, eltyp, NIL, empty)
      END;
   END ShowObject;

   PROCEDURE ShowPtrDeref (ref: RefView);
      VAR b: RefView;
   BEGIN
      ShowSelector(ref); b := ref.back;
      IF b # NIL THEN
         out.WriteChar(" ");
         InsertRefView(b.type, undo, b.adr, b.back, b.desc, b.ptr, b.name)
      END;
      out.WriteLn; out.WriteLn;
      out.WriteChar("["); WriteHex(ref.adr); out.WriteChar("]"); out.WriteTab;
      IF ref.desc = NIL THEN
         ShowObject(ref.adr)
      ELSIF FormOf(ref.desc) = 12X THEN
         ShowArray(ref.adr, 1, ref.desc, ref.ptr, ref, empty)
      ELSE
         ShowRecord(ref.adr, 1, Kernel.TypeOf(SYSTEM.VAL(ANYPTR, ref.ptr)), ref, empty)
      END;
      out.WriteLn
   END ShowPtrDeref;

   PROCEDURE Scan (VAR s: TextMappers.Scanner);
   BEGIN
```

```
      s.Scan;
      IF s.type = TextMappers.string THEN
         IF s.string = "IMPORT" THEN s.type := import
         ELSIF s.string = "MODULE" THEN s.type := smodule
         ELSIF s.string = "THEN" THEN s.type := stop
         ELSIF s.string = "OF" THEN s.type := stop
         ELSIF s.string = "DO" THEN s.type := stop
         ELSIF s.string = "END" THEN s.type := stop
         ELSIF s.string = "ELSE" THEN s.type := stop
         ELSIF s.string = "ELSIF" THEN s.type := stop
         ELSIF s.string = "UNTIL" THEN s.type := stop
         ELSIF s.string = "TO" THEN s.type := stop
         ELSIF s.string = "BY" THEN s.type := stop
         END
      ELSIF s.type = TextMappers.char THEN
         IF s.char = ";" THEN s.type := semicolon
         ELSIF s.char = "|" THEN s.type := stop
         ELSIF s.char = ":" THEN
            IF s.rider.char = "=" THEN s.rider.Read; s.type := becomes END
         ELSIF s.char = "(" THEN
            IF s.rider.char = "*" THEN
               s.rider.Read;
               REPEAT Scan(s) UNTIL (s.type = TextMappers.eot) OR (s.type = comEnd);
               Scan(s)
            END
         ELSIF s.char = "*" THEN
            IF s.rider.char = ")" THEN s.rider.Read; s.type := comEnd END
         END
      END
   END Scan;

   PROCEDURE ShowSourcePos (name: Name; adr: INTEGER);
      VAR loc: Files.Locator; fname: Files.Name; v: Views.View; m: Models.Model; conv:
Converters.Converter;
         c: Containers.Controller; beg, end: INTEGER; s: TextMappers.Scanner; w:
Windows.Window;
         n: ARRAY 256 OF CHAR;
   BEGIN
      (* search source by name heuristic *)
      n := name$; StdDialog.GetSubLoc(n, "Mod", loc, fname);
      v := Views.OldView(loc, fname); m := NIL;
      IF v # NIL THEN
         Views.Open(v, loc, fname, NIL);
         m := v.ThisModel();
         IF ~(m IS TextModels.Model) THEN m := NIL END
      END;
      IF m = NIL THEN
         (* search in open windows *)
         w := Windows.dir.First();
         WHILE (w # NIL) & (m = NIL) DO
            v := w.doc.ThisView();
            m := v.ThisModel();
            IF m # NIL THEN
               WITH m: TextModels.Model DO
                  s.ConnectTo(m); s.SetPos(0);
                  REPEAT
                     REPEAT s.Scan UNTIL s.rider.eot OR (s.type = TextMappers.string) &
(s.string = "MODULE");
                     s.Scan;
```

```
                  UNTIL s.rider.eot OR (s.type = TextMappers.string) & (s.string = name);
                  IF ~s.rider.eot THEN Windows.dir.Select(w, Windows.eager)
                  ELSE m := NIL
                  END
               ELSE m := NIL
               END
            END;
            w := Windows.dir.Next(w)
         END
      END;
      IF m = NIL THEN
         (* ask user for source file *)
         conv := NIL; v := Views.Old(Views.ask, loc, fname, conv);
         IF v # NIL THEN
            Views.Open(v, loc, fname, conv);
            m := v.ThisModel();
            IF ~(m IS TextModels.Model) THEN m := NIL END
         END
      END;
      IF m # NIL THEN
         (* mark error position in text *)
         WITH m: TextModels.Model DO
            beg := D.SourcePos(Kernel.ThisMod(n), adr);
            IF beg >= 0 THEN
               IF beg > m.Length() THEN beg := m.Length() - 10 END;
               s.ConnectTo(m); s.SetPos(beg);
               Scan(s); beg := s.start; end := beg + 3;
               IF s.type = stop THEN end := s.Pos() - 1
               ELSE
                  WHILE (s.type # TextMappers.eot) & (s.type # stop) & (s.type # semicolon)
DO
                     end := s.Pos() - 1; Scan(s)
                  END
               END;
               c := v(TextViews.View).ThisController();
               v(TextViews.View).ShowRange(beg, end, TextViews.any);
               c(TextControllers.Controller).SetSelection(beg, end)
            END
         END
      ELSE ShowParamMsg("#PrivTV:SourcefileNotFound", n, "", "")
      END
   END ShowSourcePos;

   (* ------------------  RefView ------------------ *)

   PROCEDURE (v: RefView) Internalize (VAR rd: Stores.Reader);
      VAR thisVersion: INTEGER;
   BEGIN
      v.Internalize^(rd); IF rd.cancelled THEN RETURN END;
      rd.ReadVersion(0, 0, thisVersion); IF rd.cancelled THEN RETURN END;
      v.command := open;
      rd.ReadSInt(v.type);
      IF v.type = source THEN
         rd.ReadInt(v.adr);
         rd.ReadSString(v.name)
      ELSIF v.type = module THEN
         rd.ReadSString(v.name)
      ELSE
         v.type := 0
```

```
      END
   END Internalize;

   PROCEDURE (v: RefView) Externalize (VAR wr: Stores.Writer);
      VAR t: SHORTINT;
   BEGIN
      v.Externalize^(wr);
      wr.WriteVersion(0);
      t := v.type;
      IF v.command # open THEN t := 0 END;
      wr.WriteSInt(t);
      IF t = source THEN
         wr.WriteInt(v.adr);
         wr.WriteSString(v.name)
      ELSIF t = module THEN
         wr.WriteSString(v.name)
      END
   END Externalize;

   PROCEDURE (v: RefView) CopyFromSimpleView (source: Views.View);
   BEGIN
      (* v.CopyFrom^(source); *)
      WITH source: RefView DO
         v.type := source.type; v.command := source.command; v.adr := source.adr;
v.back := source.back;
         v.desc := source.desc; v.ptr := source.ptr; v.name := source.name$;
      END
   END CopyFromSimpleView;

   PROCEDURE (v: RefView) Restore (f: Views.Frame; l, t, r, b: INTEGER);
   BEGIN
      f.DrawPath(path, 4, Ports.fill, Ports.green, Ports.closedPoly)
   END Restore;

   PROCEDURE (v: RefView) GetBackground (VAR color: Ports.Color);
   BEGIN
      color := Ports.background
   END GetBackground;

   PROCEDURE (v: RefView) HandleCtrlMsg (f: Views.Frame; VAR msg: Controllers.Message;
VAR focus: Views.View);
      VAR t, t0: TextModels.Model; m: Models.Model; x, y: INTEGER;
         isDown, new: BOOLEAN; mo: SET; script: Stores.Operation; opName:
Stores.OpName;
   BEGIN
      WITH msg: Controllers.TrackMsg DO
         IF v.type > 0 THEN
            REPEAT
               f.MarkRect(0, 0, refViewSize, refViewSize, Ports.fill, Ports.hilite, Ports.show);
               IF v.command = undo THEN ShowStatus("#PrivTV:ShowPrecedingObject")
               ELSIF v.command = update THEN ShowStatus("#PrivTV:UpdateWindow")
               ELSIF v.type = module THEN ShowStatus("#PrivTV:ShowGlobalVariables")
               ELSIF v.type = source THEN ShowStatus("#PrivTV:ShowSourcePosition")
               ELSIF v.type = heap THEN ShowStatus("#PrivTV:ShowReferencedObject")
               END;
               REPEAT
                  f.Input(x, y, mo, isDown)
               UNTIL (x < 0) OR (x > refViewSize) OR (y < 0) OR (y > refViewSize) OR
~isDown;
```

```
                f.MarkRect(0, 0, refViewSize, refViewSize, Ports.fill, Ports.hilite, Ports.hide);
                ShowStatus("");
                WHILE isDown & ((x < 0) OR (x > refViewSize) OR (y < 0) OR (y >
refViewSize)) DO
                   f.Input(x, y, mo, isDown)
                END
             UNTIL ~isDown;
             IF (x >= 0) & (x <= refViewSize) & (y >= 0) & (y <= refViewSize) THEN
                IF v.type = source THEN ShowSourcePos(v.name, v.adr)
                ELSE
                   m := v.context.ThisModel();
                   new := (v.command = open) & (v.back = NIL)
                      OR (Controllers.modify IN msg.modifiers) & (v.command # update)
                      OR ~(m IS TextModels.Model) ;
                   IF new THEN
                      t := TextModels.CloneOf(StdLog.buf); t0 := NIL
                   ELSE
                      t0 := m(TextModels.Model); t := TextModels.CloneOf(t0);
                   END;
                   out.ConnectTo(t);
                   IF v.type = heap THEN  ShowPtrDeref(v)
                   ELSIF v.type = module THEN ShowGlobals(Kernel.ThisLoadedMod(v.name))
                   END;
                   out.ConnectTo(NIL);
                   IF new THEN
                      OpenViewer(t, "#PrivTV:Variables", NewRuler())
                   ELSE
                      LS.MapString("#PrivTV:Change", opName);
                      Models.BeginScript(t0, opName, script);
                      t0.Delete(0, t0.Length()); t0.Insert(0, t, 0, t.Length());
                      Models.EndScript(t0, script)
                   END
                END
             END
          END
       | msg: Controllers.PollCursorMsg DO
          msg.cursor := Ports.refCursor
       ELSE
       END
    END HandleCtrlMsg;

    PROCEDURE (v: RefView) HandlePropMsg (VAR msg: Properties.Message);
    BEGIN
       WITH msg: Properties.Preference DO
          WITH msg: Properties.ResizePref DO msg.fixed := TRUE
          | msg: Properties.SizePref DO msg.w := refViewSize; msg.h := refViewSize
          | msg: Properties.FocusPref DO msg.hotFocus := TRUE
          ELSE
          END
       ELSE
       END
    END HandlePropMsg;

    PROCEDURE NewRefView (type, command: SHORTINT; adr: INTEGER; back: RefView;
                                    desc: Kernel.Type; ptr: ArrayPtr; name: Name): RefView;
       VAR v: RefView;
    BEGIN
       NEW(v); v.type := type; v.command := command; v.adr := adr; v.back := back;
       v.desc := desc; v.ptr := ptr; v.name := name$;
```

```
      RETURN v
   END NewRefView;

   PROCEDURE InsertRefView (type, command: SHORTINT; adr: INTEGER; back: RefView;
                                    desc: Kernel.Type; ptr: ArrayPtr; name: Name);
      VAR v: RefView; a0: TextModels.Attributes;
   BEGIN
      v := NewRefView(type, command, adr, back, desc, ptr, name);
      a0 := out.rider.attr;
      out.rider.SetAttr(TextModels.NewOffset(a0, Ports.point));
      out.WriteView(v);
      out.rider.SetAttr(a0)
   END InsertRefView;

   (* ---------------------------------------- *)

   PROCEDURE ShowStack;
      VAR ref, end, i, j, x, a, b, c: INTEGER; m, f: SHORTCHAR; mod: Kernel.Module; name,
sel: Kernel.Name;
         d: Kernel.Type;
   BEGIN
      a := Kernel.pc; b := Kernel.fp; c := 100;
      REPEAT
         mod := Kernel.modList;
         WHILE (mod # NIL) & ((a < mod.code) OR (a >= mod.code + mod.csize)) DO mod :=
mod.next END;
         IF mod # NIL THEN
            DEC(a, mod.code);
            IF mod.refcnt >= 0 THEN
               InsertRefView(module, open, 0, NIL, NIL, NIL, mod.name);
               out.WriteChar(" "); out.WriteSString(mod.name); ref := mod.refs;
               REPEAT Kernel.GetRefProc(ref, end, name) UNTIL (end = 0) OR (a < end);
               IF a < end THEN
                  out.WriteChar("."); out.WriteSString(name);
                  sel := mod.name$; i := 0;
                  WHILE sel[i] # 0X DO INC(i) END;
                  sel[i] := "."; INC(i); j := 0;
                  WHILE name[j] # 0X DO sel[i] := name[j]; INC(i); INC(j) END;
                  sel[i] := ":"; sel[i+1] := 0X;
                  out.WriteSString("   ["); WriteHex(a);
                  out.WriteSString("] ");
                  i := D.SourcePos(mod, 0);
                  IF i >= 0 THEN
                     InsertRefView(source, open, a, NIL, NIL, NIL, mod.name);
                  END;
                  IF name # "$$" THEN
                     Kernel.GetRefVar(ref, m, f, d, x, name);
                     WHILE m # 0X DO
                        ShowVar(b + x, 0, f, m, d, NIL, NIL, name, sel);
                        Kernel.GetRefVar(ref, m, f, d, x, name);
                     END
                  END;
                  out.WriteLn
               ELSE out.WriteSString(".???"); out.WriteLn
               END
            ELSE
               out.WriteChar("("); out.WriteSString(mod.name);
               out.WriteSString(")   (pc="); WriteHex(a);
               out.WriteSString(",  fp="); WriteHex(b); out.WriteChar(")");
```

```
            out.WriteLn
          END
        ELSE
          out.WriteSString("<system>   (pc="); WriteHex(a);
          out.WriteSString(",  fp="); WriteHex(b); out.WriteChar(")");
          out.WriteLn
        END;
        IF (b >= Kernel.fp) & (b < Kernel.stack) THEN
          SYSTEM.GET(b+4, a); (* stacked pc *)
          SYSTEM.GET(b, b); (* dynamic link *)
          DEC(a); DEC(c)
        ELSE c := 0
        END
      UNTIL c = 0
    END ShowStack;

    PROCEDURE (a: Action) Do;    (* delayed trap window open *)
    BEGIN
      Kernel.SetTrapGuard(TRUE);
      OpenViewer(a.text, "#PrivTV:Trap", NewRuler());
      Kernel.SetTrapGuard(FALSE);
    END Do;

    PROCEDURE GetTrapMsg(OUT msg: ARRAY OF CHAR);
      VAR ref, end, a: INTEGER; mod: Kernel.Module; name: Kernel.Name; head, tail, errstr:
ARRAY 32 OF CHAR;
        key: ARRAY 128 OF CHAR;
    BEGIN
      a := Kernel.pc; mod := Kernel.modList;
      WHILE (mod # NIL) & ((a < mod.code) OR (a >= mod.code + mod.csize)) DO mod :=
mod.next END;
      IF mod # NIL THEN
        DEC(a, mod.code); ref := mod.refs;
        REPEAT Kernel.GetRefProc(ref, end, name) UNTIL (end = 0) OR (a < end);
        IF a < end THEN
          Strings.IntToString(Kernel.err, errstr);
          key := name + "." + errstr;
          LS.MapString("#" + mod.name$ + ":" + key, msg);
          IF key = msg THEN
            Kernel.SplitName (mod.name$, head, tail);
            IF head = "" THEN head := "System" END;
            key := tail + "." + name + "." + errstr;
            Dialog.MapString("#" + head + ":" + key, msg);
            IF key = msg THEN msg := "" END
          END
        END
      END
    END GetTrapMsg;

    PROCEDURE Trap;
      VAR a0: TextModels.Attributes; action: Action;
        msg: ARRAY 512 OF CHAR;
    BEGIN
      out.ConnectTo(TextModels.CloneOf(StdLog.buf));
      a0 := out.rider.attr;
      out.rider.SetAttr(TextModels.NewWeight(a0, Fonts.bold));
      IF Kernel.err = 129 THEN out.WriteSString("invalid WITH")
      ELSIF Kernel.err = 130 THEN out.WriteSString("invalid CASE")
      ELSIF Kernel.err = 131 THEN out.WriteSString("function without RETURN")
```

```
	ELSIF Kernel.err = 132 THEN out.WriteSString("type guard")
	ELSIF Kernel.err = 133 THEN out.WriteSString("implied type guard")
	ELSIF Kernel.err = 134 THEN out.WriteSString("value out of range")
	ELSIF Kernel.err = 135 THEN out.WriteSString("index out of range")
	ELSIF Kernel.err = 136 THEN out.WriteSString("string too long")
	ELSIF Kernel.err = 137 THEN out.WriteSString("stack overflow")
	ELSIF Kernel.err = 138 THEN out.WriteSString("integer overflow")
	ELSIF Kernel.err = 139 THEN out.WriteSString("division by zero")
	ELSIF Kernel.err = 140 THEN out.WriteSString("infinite real result")
	ELSIF Kernel.err = 141 THEN out.WriteSString("real underflow")
	ELSIF Kernel.err = 142 THEN out.WriteSString("real overflow")
	ELSIF Kernel.err = 143 THEN
	   out.WriteSString("undefined real result  (");
	   out.WriteIntForm(Kernel.val MOD 10000H, TextMappers.hexadecimal, 4, "0",
TextMappers.hideBase); out.WriteSString(", ");
	   out.WriteIntForm(Kernel.val DIV 10000H, TextMappers.hexadecimal, 3, "0",
TextMappers.hideBase); out.WriteChar(")")
	ELSIF Kernel.err = 144 THEN out.WriteSString("not a number")
	ELSIF Kernel.err = 200 THEN out.WriteSString("keyboard interrupt")
	ELSIF Kernel.err = 201 THEN
	   out.WriteSString("NIL dereference")
	ELSIF Kernel.err = 202 THEN
	   out.WriteSString("illegal instruction: ");
	   out.WriteIntForm(Kernel.val, TextMappers.hexadecimal, 5, "0",
TextMappers.showBase)
	ELSIF Kernel.err = 203 THEN
	   IF (Kernel.val >= -4) & (Kernel.val < 65536) THEN out.WriteSString("NIL dereference
(read)")
	   ELSE out.WriteSString("illegal memory read (ad = "); WriteHex(Kernel.val);
out.WriteChar(")")
	   END
	ELSIF Kernel.err = 204 THEN
	   IF (Kernel.val >= -4) & (Kernel.val < 65536) THEN out.WriteSString("NIL dereference
(write)")
	   ELSE out.WriteSString("illegal memory write (ad = "); WriteHex(Kernel.val);
out.WriteChar(")")
	   END
	ELSIF Kernel.err = 205 THEN
	   IF (Kernel.val >= -4) & (Kernel.val < 65536) THEN out.WriteSString("NIL procedure
call")
	   ELSE out.WriteSString("illegal execution (ad = "); WriteHex(Kernel.val);
out.WriteChar(")")
	   END
	ELSIF Kernel.err = 257 THEN out.WriteSString("out of memory")
	ELSIF Kernel.err = 10001H THEN out.WriteSString("bus error")
	ELSIF Kernel.err = 10002H THEN out.WriteSString("address error")
	ELSIF Kernel.err = 10007H THEN out.WriteSString("fpu error")
	ELSIF Kernel.err < 0 THEN
	   out.WriteSString("Exception "); out.WriteIntForm(-Kernel.err,
TextMappers.hexadecimal, 3, "0", TextMappers.showBase)
	ELSE
	   out.WriteSString("TRAP "); out.WriteInt(Kernel.err);
	   IF Kernel.err = 126 THEN out.WriteSString("  (not yet implemented)")
	   ELSIF Kernel.err = 125 THEN out.WriteSString("  (call of obsolete procedure)")
	   ELSIF Kernel.err >= 100 THEN out.WriteSString("  (invariant violated)")
	   ELSIF Kernel.err >= 60 THEN out.WriteSString("  (postcondition violated)")
	   ELSIF Kernel.err >= 20 THEN out.WriteSString("  (precondition violated)")
	   END
	END;
```

```
        GetTrapMsg(msg);
        IF msg # "" THEN out.WriteLn; out.WriteString(msg) END;

        out.WriteLn; out.rider.SetAttr(a0);
        out.WriteLn; ShowStack;
        NEW(action); action.text := out.rider.Base();
        Services.DoLater(action, Services.now);
        out.ConnectTo(NIL)
    END Trap;

BEGIN
    Kernel.InstallTrapViewer(Trap);
    empty := "";
    path[0].x := refViewSize DIV 2; path[0].y := 0;
    path[1].x := refViewSize; path[1].y := refViewSize DIV 2;
    path[2].x := refViewSize DIV 2; path[2].y := refViewSize;
    path[3].x := 0; path[3].y := refViewSize DIV 2
END PrivTV.
```

Приложение 3. Текстовые ресурсы *PrivTV* (файл "Priv/Rsrc/TVStrings")

STRINGS

Change	Change
Unknown	unknown
SourcefileNotFound	sourcefile for ^0 not found
Unloaded	^0 unloaded
ProcInUnloadedMod	procedure in unloaded module
IllegalAddress	illegal address:
IllegalPointer	illegal pointer !!!
Undefined	undefined:
UnknownFormat	unknown format:
Variables	Variables
Trap	Trap
Update	Update
Fields	fields
Elements	elements
ShowPrecedingObject	show preceding object
ShowGlobalVariables	show global variables
ShowSourcePosition	show source position
ShowReferencedObject	show referenced object
UpdateWindow	update window