
Chapter 1:01
The Basics
Principia v3.0
Reference Series
(c) Western Star
Entertainment Ltd, 2003-2006

Chapter 1 - Basics

- Demo101A – Starting a simple Principia app
- Demo101B – Basic GUI with user code
- Demo101C – Adding text and performance info
- Demo101D – Adding sound to the basic GUI
- Demo101E – Diagnostics
- Demo101F – Rendering basics
- Demo101G – Finalizing the demos GUI


PrincipiaV.3.0.0

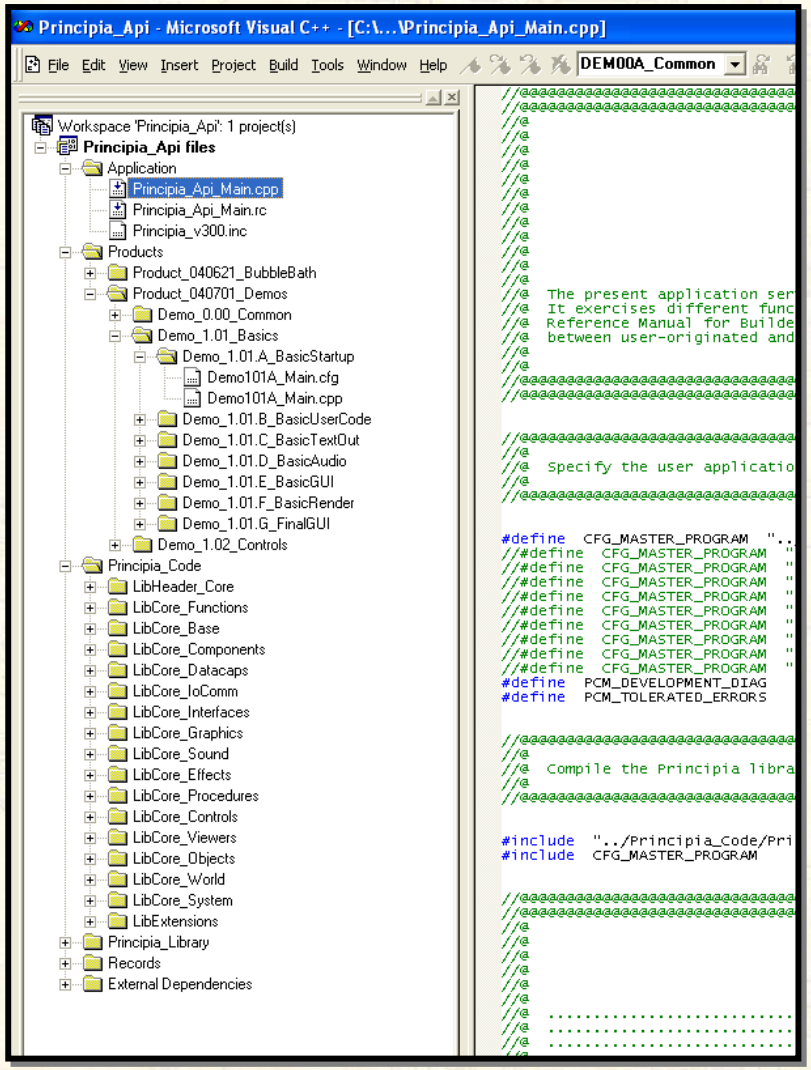
D101A – Basic Startup

- No custom user code – run from script only
- The simplest requirements possible
  - Display GUI framework
  - Display cursor
  - Exit when user presses ESC
  - Fixed 1024 x 768 x 32bpp resolution
  - No sound, bells or whistles
- Illustrates the basic structure of a Principia application and the use of the Principia API within a C++ development environment
- Particularly useful for troubleshooting basic system issues across different platforms

D101A – Basic Concepts

- **Development environment**
  - E.g. a Visual C++ project for building Principia-based applications. Ensures that all the external libraries and includes are properly compiled.
- **API file**
  - The main C++ file that is compiled to produce the executable application.
- **Main user file**
  - The main C++ program written by the user. It must be included in the API file.
- **Script file**
  - A file containing Principia commands and definitions that can be parsed and executed by the Principia interpreter.
- **Root directory**
  - An absolute path relative to which all Principia file name references are made. Typically, contains script files, media files, runtime directories...etc.

D101A – Dev Structure



- Open VC++ project Principia_Api that contains all the demos
- Project compiles Principia_Api_Main.cpp which is just a pointer to the desired demo code. This file is in the project application folder.
- Principia is included in source form. Other ways to link Principia will be explored later. Principia code is in its own folder.
- The demo source and data hierarchy resides in its own folder structure.

D101A - API Code

- API code is only a reference to the actual main
- API code links to the Principia library
- API code specifies the compilation modality: enable complete runtime error checking and compilation of self-diagnostic code sections.

```
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//
//@                                                                                              @//
//@  Specify the user application and compilation modality                                       @//
//@                                                                                              @//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//


#define  CFG_MASTER_PROGRAM   "../Product_040701_Demos/Demo_1.01.A_BasicStartup/Demo_101A_Main.cpp"
#define  PCM_DEVELOPMENT_DIAG   1
#define  PCM_TOLERATED_ERRORS   0


//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//
//@                                                                                              @//
//@  Compile the Principia library and the user application                                      @//
//@                                                                                              @//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//


#include  "../Principia_Code/Principia_v300.inc"
#include  CFG_MASTER_PROGRAM
```

D101A – Main Code

- **The main Demo_101A code specifies the root directory, and executes the single script therein.**

```
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@T.Valkov(c)2000-2004@@@//
//@                                                                                            @//
//@  MODULE: Application Definition                                                            @//
//@  GROUPS: User custom code                                                                  @//
//@                                                                                            @//
//@  This module provides definitions of all the components of a Principia-based application  @//
//@  .......................................................................................   @//
//@                                                                                            @//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//

#define PROD_NAME   NULL
#define PROD_VERS   NULL
#define ROOT_NAME   "..//Product_040701_Demos//Demo_1.01.A_BasicStartup//"
#define CONF_FILE   "Demo_101A_Main.cfg"
#define INST_FILE   NULL
#define CORE_DUMP   "Files_Runtime//CoreDump.txt"


//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@T.Valkov(c)2000-2004@@@//
//@                                                                                            @//
//@  MODULE: Application Main                                                                  @//
//@  GROUPS: User custom code                                                                  @//
//@                                                                                            @//
//@  This module implements the main of a Principia-based application.......................   @//
//@  .......................................................................................   @//
//@                                                                                            @//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//

BGN_PRINCIPIA_MAIN (PROD_NAME, PROD_VERS, ROOT_NAME, INST_FILE, CORE_DUMP){

   /* Execute the application main Principia script */
   Principia->ParseFile(CONF_FILE);

END_PRINCIPIA_MAIN}
```

D101A – Script, Interfaces

- **Principia scripts define application components and actions. A component is defined by its properties (#tags) and actions (#method), triggered by listening to various signals in different channels of the Principia system interface.**
- **The demo begins by incorporating some standard Principia components, such as DEF_TEX: a "property definer" for common textures. These are in separate included script files. Leave them alone unless you are an advanced Principia user.**

```
@@@@@ This is a comment line
@@@@@ Include standard name definitions, of which we will use one

##parse "..\..\Principia_Library\Principia_StandardDefinitions_Core_v300.cfg"
##parse "..\..\Principia_Library\Principia_StandardDefinitions_Procedures_v300.cfg"
##parse "../Demo_Common/Files_Scripts/Module_INTERFACES.cfg"
```

- **The demo script continues by creating the required interfaces for the application.**
  - **A system interface, defining channels for internal signaling in our applications, and features basic "clock-pausing" methods**
  - **An input interface with standard keyboard and mouse**
  - **A windowed 1024x768 32-bit graphic interface**
  - **A basic audio interface with volume control variables**
- **The interface creation is also contained in a separate script. Interface definition is too complex to be covered here. Suffice it to say that the script creates what we need in a typical application. For now, just copy this script.**

D101A – Script, Frame

- Two most basic components are surfaces and frames
- Surfaces hold graphic images. They can be created from files
- Frames display the surface content flat on screen
- Our interface is basically a frame covering the screen
- Surfaces can have many properties - such as the definer tag that specifies their memory location and usage. Here, we use a standard predefined flag for a non-transparent 2D surface in system memory.

```
@@@@@@ Definition of a frame containing the graphic interface.
@@@@@@ The DEF_S2D property is defined in the Standard Includes
@@@@@@ to denote a surface stored in plain system memory.


##define (S_INTERFACE) as <SURFACE>
   #tag Definer = (       DEF_S2D )
   #tag File    = "../Demo_Common/Files_Media/GuiL1/Image_MetalMelt.bmp"
   #tag LoadNow = (             1 )

##define (F_INTERFACE) as <FRAME_2A>
   #tag Image   = (   S_INTERFACE )
```

D101A – Script, Cursor

- The pointer is also created out of frames.
- The frames are transparent and offset from their default upper-left corner insertion point. This way, a command to display the pointer at (X,Y) places the pointer tip there, instead of the pointer upper left corner.
- The pointer can hide or chose its appearance by activating a given frame. These actions are triggered by signals in the CH_CRS channel.

```
##define (S_CRS_NORM) as <SURFACE>
   #tag File      =
        "../Demo_Common/Files_Media/Cursors/Cursor_Norm_C.bmp"
   #tag Definer   =  (       DEF_TEX )
   #tag LoadNow   =  (             1 )

##define (S_CRS_OVER) <SURFACE>
   #tag File      =
        "../Demo_Common/Files_Media/Cursors/Cursor_Over_C.bmp"
   #tag Definer   =  (       DEF_TEX )
   #tag LoadNow   =  (             1 )

##define (F_CRS_NORM) as <FRAME_2A>
   #tag Image     =  (    S_CRS_NORM )
   #tag Offsets   =  (     -22 ,  -2 )

##define (F_CRS_OVER) as <FRAME_2A>
   #tag Image     =  (    S_CRS_OVER )
   #tag Offsets   =  (     -24 ,  -2 )

##define (API_CURSOR) as <CURSOR_2A>
   #method        =  (  EXE_HIDE   ,  CH_CRS ,  X_HIDE )
   #method        =  (  EXE_SETIMG ,  CH_CRS ,  X_NORM ,  F_CRS_NORM )
   #method        =  (  EXE_SETIMG ,  CH_CRS ,  X_OVER ,  F_CRS_OVER )
```

D101A – Script, Controls

- Another basic component is a <VARIABLE> that holds some application data. This one takes a true value if a X_TERM signal is present in the CH_SYS channel.
- Other basic components generate control signals in response to mouse location or keyboard action.
- Here, signals are generated if the user presses ESC, the mouse is in the viewport or outside of it.
- Note that channel names, signal names…etc are defined by the user.

```
##define <SYS_VARIABLE>  =  (   VSC_TERM ,   VARTYPE_LOG ,     - ,
       CH_SYS ,   X_TERM )

##define (API_KJAY) as <KJAY_2A>
   #method        =  (    ON_KEYPRS  ,    CH_SYS ,     X_TERM  ,  ESC  )

##define (API_PJAY) as <PJAY_2A>
   #method        =  (        ON_API ,      CH_CRS ,      X_NORM  )
   #method        =  (        ON_LFT ,      CH_CRS ,      X_HIDE  )
   #method        =  (        ON_RGT ,      CH_CRS ,      X_HIDE  )
   #method        =  (        ON_TOP ,      CH_CRS ,      X_HIDE  )
   #method        =  (        ON_BOT ,      CH_CRS ,      X_HIDE  )
```

D101A – Script, Scene Setting

- The basic display components are pages, books and scenes.
- A page is a collection of other components. Here, our page is the interface, the cursor, and the two signal-generating controllers.
- A book is a collection of pages that can be turned on or off. Our book here has a single page, always on.
- A scene has a display loop that shows all the scene components – here, our book.
- The scene runs the display loop until the control variable defined above turns true.

```
##define (PG_MAIN) as <PAGE_2A>
   #tag Element   =  (   F_INTERFACE ,      0 ,       0 ,      - )
   #tag Element   =  (    API_CURSOR ,      0 ,       0 ,      - )
   #tag Element   =  (      API_PJAY ,      0 ,       0 ,      - )
   #tag Element   =  (      API_KJAY ,      0 ,       0 ,      - )


##define (BK_MAIN) as <BOOK_2A>
   #tag StartOpen =  (          1 )
   #tag StateID   =  (         MAIN ,     - ,     - ,       0 ,      0
       ,                1 )
   #tag Element   =  (         MAIN ,     - ,     - ,       0 ,      0
       ,    PG_MAIN ,    1 )


##define (SCENE_MAIN) as <SCENE_2A>
   #tag VarStop           =  (     VSC_TERM )
   #tag Element           =  (     - ,    - ,      BK_MAIN )



@@@@@@@ This statement is the actual scene execution
@@@@@@@ Everything else before is definition and setup


##run (SCENE_MAIN)
```

D101A – Scripting Syntax (1)

- **Principia script commands**

##parse “Name”	Interprets a script file
##create <ID>	Creates an API interface
##define <ID>	Defines a component
##run <ID>	Runs an executable object
##save <ID>	Saves object data
##delete <ID>	Deallocates a component
##set <State>	Sets component states
##goto/###label	Script flow control
##halt	Stops parsing and exits
##iff/###els/###eif	Conditional flow control
##version	Requests Principia version

D101A – Scripting Syntax (2)

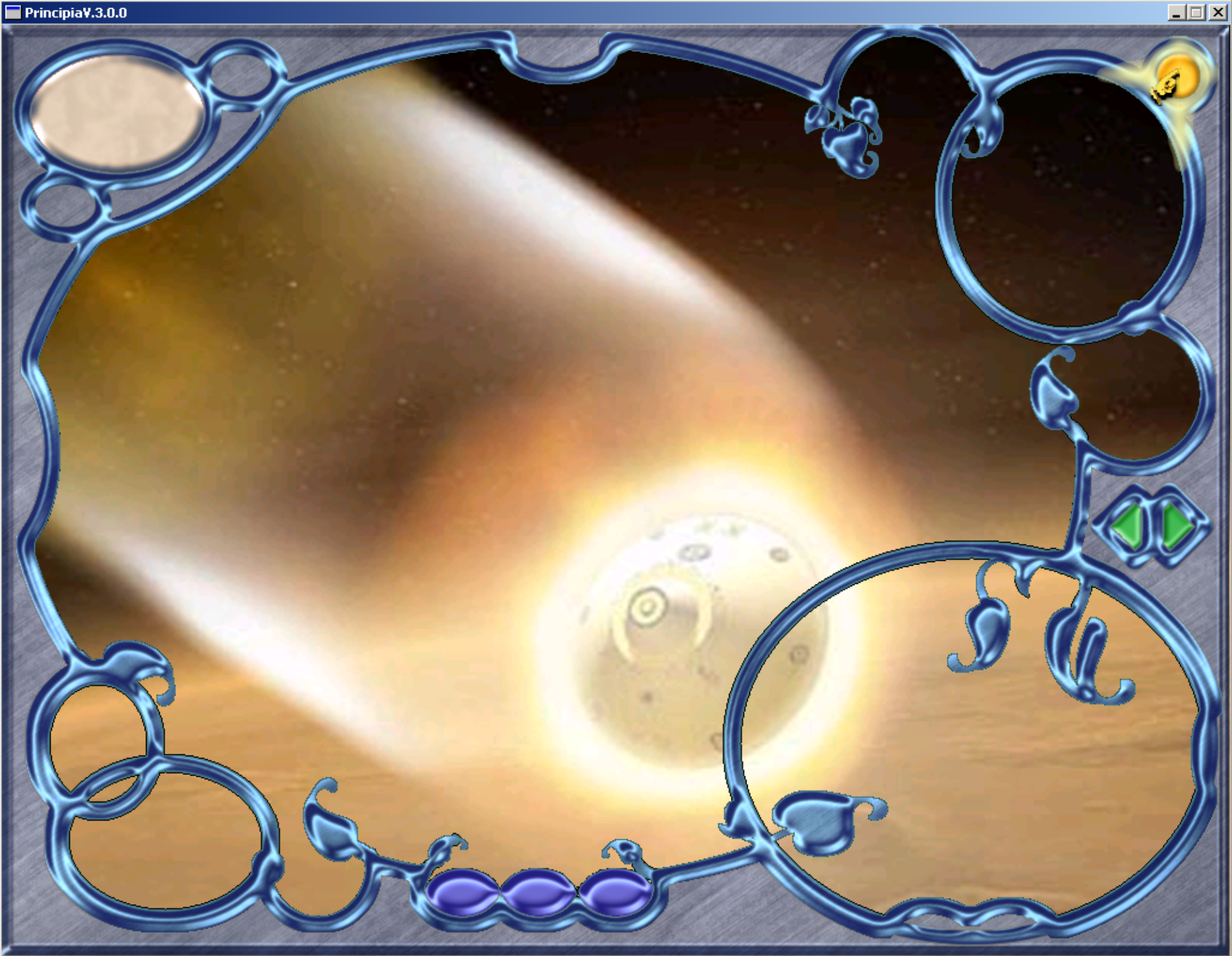
- **Principia command tags provide arguments for each command. There are two types of tags:**
  - Short tags for short, single-line commands just hold the argument data between parentheses in the same line as the command. For example ##run (CMD) with (ARGS...). In such cases, the tags do not have names. Principia interprets the context of the single line command, and automatically determines what the tags mean.
  - Long tags for commands that require complex data, such as ##define. Each long tag has the format #tag TagName = (data stream) on its own line.
  - Most component definition commands use named tags, and Principia expects you to use correct names. In development mode, unrecognized tags will be listed in the runtime diagnostic file.

D101A – Scripting Syntax (2)

- **Tag data stream syntax**
  - () delimits tag data
  - , or ; separates tag data items
  - @ ’ leading/trailing comment, skip rest of line
  - $Var inserts the value of variable Var into stream
  - “” encloses text data
  - {}… used to issue special instructions in text data
- **Text data special instructions**
  - “{ABSOLUTE}…” with file names indicates that the file name is an absolute path.

D101A – Summary

- Our application does not do much on the surface besides displaying an arguably funky interface with a moving pointer.
  - Behind the scenes, all of the complex initializations of the 3D graphic accelerators and various devices has been accomplished
- It has taken a single line of code to do this: Principia->ParseFile(CONFIG_FILE_NAME)
- Principia V3 features hundreds of components with a myriad of properties and actions.
  - Blending scripted components with some custom code enables the rapid creation of extremely powerful, commercial-grade entertainment products.
- How to do it is the subject of this document.


PrincipiaV.3.0.0

D101B – Basic User GUI

- Add sockets for custom user code in scene
- Introduce some new GUI features
  - Transparent interface window
  - Background image visible thru window
  - Simple button to exit application
  - Button to change pointer and glow if mouse over
  - Cursor to cast shadow
- Illustrates the basic methods for blending custom user code with Principia components
- Illustrates basic concepts in image transparency and memory storage

D101B – Main Code

- The API main C++ file has not changed except to point to the user main Demo_101B.cpp file
- That file enables the user to define three custom procedures and bind them to the start, display loop and termination of the main scene. These procedures do nothing for the moment.

```
/* Code changes relative to Demo01A */
void User_Scene_Init (void* Varg /*Pointer to calling parent object */)
BGN_ACTOR_CODE{
END_ACTOR_CODE}

void User_Scene_Main (void* VArg /*Pointer to calling parent object */)
BGN_ACTOR_CODE{
END_ACTOR_CODE}

void User_Scene_Term (void* VArg /*Pointer to calling parent object */)
BGN_ACTOR_CODE{
END_ACTOR_CODE}


/* Code changes relative to Demo01A */
BGN_PRINCIPIA_MAIN (PROD_NAME, PROD_VERS, ROOT_NAME, INST_FILE, CORE_DUMP){

   /* Bind the user custom code to particular application objects */
   Principia->BindUserProcInit ( "SCENE_MAIN" , User_Scene_Init );
   Principia->BindUserProcMain ( "SCENE_MAIN" , User_Scene_Main );
   Principia->BindUserProcTerm ( "SCENE_MAIN" , User_Scene_Term );

   /* Execute the application main Principia script */
   Principia->ParseFile(CONF_FILE);

END_PRINCIPIA_MAIN}
```

D101B – Script (1)

- The interface definer is changed to request a transparent surface in managed video memory
- This transparency uses the default color key 0xFF000000 (black).
- To create the illusion of a shadow, the pointer transparency utilizes an alpha map provided in a separate file.
- If a surface is created with an alpha map, color keys are ignored.

```
@@@@@ Transparent interface frame is based on a simple color key

##define (S_INTERFACE) as <SURFACE>
   #tag Definer   =  (       DEF_TEX )
   #tag File      =  "../Demo_Common/Files_Media/GuiL1/Image_MetalMelt.bmp"
   #tag LoadNow   =  (             1 )


@@@@@ Cursor frames use a shifted alpha transparency map of the
@@@@@ object image to create the illusion of a shadow.

##define (S_CRS_NORM) as <SURFACE>
   #tag File      =  "../Demo_Common/Files_Media/Cursors/Cursor-Norm.bmp"
   #tag Alpha     =  "../Demo_Common/Files_Media/Cursors/Cursor-Norm.xmp"
   #tag Definer   =  (       DEF_TEX )
   #tag LoadNow   =  (             1 )

##define (S_CRS_OVER) <SURFACE>
   #tag File      =  "../Demo_Common/Files_Media/Cursors/Cursor-Over.bmp"
   #tag Alpha     =  "../Demo_Common/Files_Media/Cursors/Cursor-Over.xmp"
   #tag Definer   =  (       DEF_TEX )
   #tag LoadNow   =  (             1 )
```

 +  = Hand outline with shadow

D101B – Script (2)

- The button has a single state and two actions. When not active, it displays nothing – its base image is already on the interface.
- The check region for the pointer-button interaction can be a complex polygon, specified relative to the button insertion point.
- When moused over, the button generates a pointer change signal and displays a frame simulating glow.
- When clicked, the button sends the same termination signal as the ESC key.
- The glow effect is achieved using an alpha map.

```
@@@@ This button frame is activated when the button is moused over
@@@@ It implements a glow effect using an alpha map and bright
@@@@ background on the object image.

##define (S_BTN_TERM) as <SURFACE>
   #tag File      =  "../Demo_Common/Files_Media/GuiL1/Btn_Term_Over_C.bmp"
   #tag Alpha     =  "../Demo_Common/Files_Media/GuiL1/Btn_Term_Over_X.bmp"
   #tag Definer   =  (       DEF_TEX )
   #tag LoadNow   =  (             1 )

##define (F_BTN_TERM) as <FRAME_2A>
   #tag Image     =  (    S_BTN_TERM )


@@@@ The button features two signal actions and a single state.
@@@@ The button check region is specified as a polygon in pixel
@@@@ coordinates relative to the button insertion point.

##define (BTN_TERM) as <BUTTON_2C>
   #tag State     =  (    NORMAL ,           - ,     NONE ,     NONE ,
       - )
   #method        =  (   ON_OVER ,      CH_CRS ,   X_OVER ,   NORMAL ,
       F_BTN_TERM ,       NONE ,        - )
   #method        =  (   ON_LCLK ,      CH_SYS ,   X_TERM ,   NORMAL ,
       F_BTN_TERM ,       NONE ,        - )
   #tag Region    =  (  60 ,  30 )
   #tag Region    =  (  80 ,  10 )
   #tag Region    =  ( 100 ,  30 )
   #tag Region    =  (  80 ,  50 )
```

D101B – Script (3)

- The background image is simply a transparent frame displayed behind the interface.
- The enumeration order determines display order.
- Nesting: The button is nested within a page, nested within a book, nested within a scene. This is a key Principia concept enabling powerful and flexible scene construction.
- Anchors: The position of a nested object is specified relative to the parent anchor using the upper-left-corner pixel-units display paradigm
- The topmost parent is usually a scene object, anchored to the screen upper left corner.

```
##define (PG_MAIN) as <PAGE_2A>
   #tag Anchor    =  (              - ,      - )
   #tag Region    =  (              - ,      - ,      - ,      - )
   #tag Element   =  (     F_BACK_01 ,      0 ,      0 ,      - )
   #tag Element   =  (   F_INTERFACE ,      0 ,      0 ,      - )
   #tag Element   =  (       BTN_TERM ,    900 ,      9 ,      - )
   #tag Element   =  (    API_CURSOR ,      0 ,      0 ,      - )
   #tag Element   =  (      API_PJAY ,      0 ,      0 ,      - )
   #tag Element   =  (      API_KJAY ,      0 ,      0 ,      - )
```

D101B – Summary

- Significant new functionality has been added with only few new definition lines of script.
- A key Principia programming model introduced
  - Binding of custom user code to Principia objects enables to use all the predefined capabilities of the myriad objects in a customized user application.
- Two key new capabilities introduced
  - Alpha maps enable easy definition of translucency, shadows, halos and many other complex effects.
  - Nesting of components in books, pages and scenes enables to create and manage complex scenes.
- These are few of the thousands of features that drive the power and flexibility of Principia V3.


PrincipiaV.3.0.0
FPS: 62.5
VPF: 612
SPF: 5
ADAPTER: Intel(R) 82915G Express Chipset Family
OS: Windows 2000
GS: DirectX 9.0
FREE MEMORY: 757413 kb
FREE DISK: 142912 mb

D101C – Basic Text Output

- Display a corner diagnostic panel
  - Showing application FPS: frames per second
  - Showing application VPF: vertices per frame
  - Showing application SPF: state changes per frame
- Display background info in a separate frame
  - Operating and graphic system
  - Free disk space on volume holding demos
  - Free memory
  - Adapter information
- Introduce some more basic concepts:
  - Fonts
  - Principia commands
  - Textout component for text display
- Introduce key custom code basics
  - Accessing Principia objects
  - Integrating custom code with Principia object execution
  - Executing graphic operations from within custom code

D101C – Script (1)

- Numeric variables are defined in the script to hold the desired values
- Textouts are key components for text output. Three textouts are defined and connected to the variables
- A separate page holding only the three textouts is defined. This is all needed to display the values of the three variables on the screen.

```
##define <SYS_VARIABLE>   =  (    VI_FPS ,   VARTYPE_FLT ,   9.0 )
##define <SYS_VARIABLE>   =  (    VI_VPF ,   VARTYPE_INT ,     7 )
##define <SYS_VARIABLE>   =  (    VI_SPF ,   VARTYPE_INT ,     8 )


##define (TX_FPS) as <TEXTOUT_2A>
   #tag Plate     =  (          - ,    - ,    - )
   #tag Value     =  (    VI_FPS ,    - ,    - ,  "FPS:%6.1f" , FN_INFO14 )

##define (TX_VPF) as <TEXTOUT_2A>
   #tag Plate     =  (          - ,    - ,    - )
   #tag Value     =  (    VI_VPF ,    - ,    - ,  "VPF:%6d" , FN_INFO14 )

##define (TX_SPF) as <TEXTOUT_2A>
   #tag Plate     =  (          - ,    - ,    - )
   #tag Value     =  (    VI_SPF ,    - ,    - ,  "SPF:%6d" , FN_INFO14 )

##define (PG_INFO) as <PAGE_2A>
   #tag Anchor    =  (              - ,      - )
   #tag Region    =  (              - ,      - ,       - ,      - )
   #tag Element   =  (        TX_FPS ,      0 ,       0 ,      - )
   #tag Element   =  (        TX_VPF ,      0 ,      18 ,      - )
   #tag Element   =  (        TX_SPF ,      0 ,      36 ,      - )
```

D101C – User Code (1)

- Where the variable values are going to come from?
- The user code declares references to the three variables.
- When the scene begins, the references are connected to the objects declared in the script.
- When the scene runs, the variables (now connected) are assigned with the three performance metrics from the public graphic interface object.
- This is one of the key techniques for integrating user code with Principia objects and frameworks.

```
DECLARE_GLOBL   SX_Variable*   aVarFPS   = NULL;
DECLARE_GLOBL   SX_Variable*   aVarVPF   = NULL;
DECLARE_GLOBL   SX_Variable*   aVarSPF   = NULL;


void User_Scene_Init (void* VArg)
BGN_ACTOR_CODE{

   /* Obtain pointers to the Principia objects defined in the script */
   aVarFPS = (SX_Variable*)  Principia->GetReferenceTo("VI_FPS");
   aVarVPF = (SX_Variable*)  Principia->GetReferenceTo("VI_VPF");
   aVarSPF = (SX_Variable*)  Principia->GetReferenceTo("VI_SPF");
   aFreeFr = (CX_Frame2A*)   Principia->GetReferenceTo("F_LEAF_01");
   aFreeTx = (CX_Textout2A*) Principia->GetReferenceTo("TX_FREE");

END_ACTOR_CODE}


void User_Scene_Main (void* VArg)
BGN_ACTOR_CODE{

   /* Update the display variables with the appropriate internal metrics */
   aVarFPS->Assign(Api_PerfMonitor->PerfRepFPS);
   aVarVPF->Assign(Api_PerfMonitor->PerfRepVPF);
   aVarSPF->Assign(Api_PerfMonitor->PerfRepSPF);

END_ACTOR_CODE}
```

D101C – Script (2)

- The purpose of the new user code is to only assign the correct value to the displayed data
- To do the actual display, all we need to do is add our new page with textouts to the scene display stream by incorporating it in the main book at the desired insertion coordinates.
- But wait ... how did the system know what font and text size to use?

```
##define (BK_MAIN) as <BOOK_2A>
   #tag StartOpen =  (       1 )
   #tag StateID   =  (    MAIN ,   - ,   - ,    0 ,   0 ,        1 )
   #tag Element   =  (    MAIN ,   - ,   - ,    0 ,   0 ,  PG_MAIN ,    1 )
   #tag Element   =  (    MAIN ,   - ,   - ,   52 ,  48 ,  PG_INFO ,    1 )
```

D101C – Script (3)

- Principia features a fast industrial grade text engine that does not rely on Windows fonts
- Fonts are defined via glyph maps that enable fonts to be packaged with the product and achieve many special effects.
- The Principia text engine renders text by rasterizing portions of the glyph map through the GPU.
- Two fonts are defined in the demo and their glyph maps stored as textures in video memory

```
@@@@ Typical production font definition refers requires three files
@@@@ 1) Glyph image map
@@@@ 2) Glyph alpha map
@@@@ 3) Glyph box geometry .xfd data file
@@@@ Plus surface definer for the glyph data, alignment flags …etc.


##define (FN_INFO14) as <FONT_3A>
   #tag ImgFile   =  "..\Demo_Common\Files_Media\Fonts\Font_INFO14_C.bmp"
   #tag AlfFile   =  "..\Demo_Common\Files_Media\Fonts\Font_INFO14_X.bmp"
   #tag XfdFile   =  "..\Demo_Common\Files_Media\Fonts\Font_INFO14_C.xfd"
   #tag Template  =  (       DEF_TEX )
   #tag AlignH    =  (    TXTINS_LFT )
   #tag AlignV    =  (    TXTINS_TOP )

##define (FN_INFO12) as <FONT_3A>
   #tag ImgFile   =  "..\Demo_Common\Files_Media\Fonts\Font_INFO12_C.bmp"
   #tag AlfFile   =  "..\Demo_Common\Files_Media\Fonts\Font_INFO12_X.bmp"
   #tag XfdFile   =  "..\Demo_Common\Files_Media\Fonts\Font_INFO12_C.xfd"
   #tag Template  =  (       DEF_TEX )
   #tag AlignH    =  (    TXTINS_LFT )
   #tag AlignV    =  (    TXTINS_TOP )
```

D101C – Script (4)

- Where do glyph maps come from? What if I just need a simple Arial font?
- D101C shows how to use Principia to automatically generate the glyph maps needed from a system font
- The glyph map of any font can be generated by using the SAVE command
- If not needed further, any object can be removed from memory using the DELETE command

```
@@@@ This section of the configuration script shows how to use Principia
@@@@ to generate glyph maps based on installed fonts. It needs to be run
@@@@ only once to generate the glyph maps used in the demos, including
@@@@ the .xfd files. The glyph maps can be used as generated or edited to
@@@@ create the desired special effects such as glows …etc.


##define (FN_INFO14) as <FONT_3A>
   #tag Foundry   =  "Lucida Console"
   #tag Template  =  (       DEF_TEX )
   #tag Height    =  (            14 )
   #tag Weight    =  (           800 )
   #tag TextClr   =  (      CL_DKBLU )
   #tag BackClr   =  (             - )
   #tag AlignH    =  (    TXTINS_LFT )
   #tag AlignV    =  (    TXTINS_MID )
   #tag ExpandW   =  (             5 )
   #tag ExpandH   =  (             5 )

##save (FN_INFO14) in ("..\Demo_Common\Files_Media\Fonts\Font_INFO14_C.bmp")
       alpha in ("..\Demo_Common\Files_Media\Fonts\Font_INFO14_X.bmp")


##define (FN_INFO12) as <FONT_3A>
   #tag Foundry   =  "Verdana"
   #tag Template  =  (       DEF_TEX )
   #tag Height    =  (            14 )
   #tag Weight    =  (           800 )
   #tag TextClr   =  (      CL_DKRED )
   #tag BackClr   =  (             - )
   #tag AlignH    =  (    TXTINS_LFT )
   #tag AlignV    =  (    TXTINS_MID )
   #tag ExpandW   =  (             5 )
   #tag ExpandH   =  (             5 )

##save (FN_INFO12) in ("..\Demo_Common\Files_Media\Fonts\Font_INFO12_C.bmp")
       alpha in ("..\Demo_Common\Files_Media\Fonts\Font_INFO12_X.bmp")


##delete (FN_INFO14)
##delete (FN_INFO12)
```

D101C – Script (5)

- This took care of the FPS display. How about the background info text box? It can be done in the same way as above using variables and static textouts...
- But we will use instead a different approach where the information is generated directly from within the custom user code.
- Here, we define an empty textout and page to be used as templates in the user code.

```
##define (S_LEAF_01) as <SURFACE>
   #tag File      =
        "../Demo_1.01.C_BasicTextOut/Files_Media/Image_MatteBack-C.bmp"
   #tag Alpha     =
        "../Demo_1.01.C_BasicTextOut/Files_Media/Image_MatteBack-X.bmp"
   #tag Definer   =  (        DEF_TEX )
   #tag LoadNow   =  (             1 )

##define (F_LEAF_01) as <FRAME_2A>
   #tag Image     =  (     S_LEAF_01 )

##define (TX_FREE) as <TEXTOUT_2A>
   #tag Plate     =  (          - ,     - ,     - )
   #tag Value     =  (          - ,     - ,     - ,    - ,   FN_INFO12 )

##define (PG_LEAF_01) as <PAGE_2A>
   #tag Anchor    =  (             - ,      - )
   #tag Region    =  (             - ,      - ,       - ,      - )
```

D101C – User Code (2)

- Pages and few other control objects are ideal for incorporating custom code in the rendering stream.
- We need to bind the template page to a user procedure which will perform the actual desired task.
- We also need to include the template page in the right place of the scene rendering stream.

```
// In user C++ code, bind the empty page object to a user procedure that
// Will actually perform the operations desired, namely the display of
// various background information in a semi-transparent frame overlay.

BGN_PRINCIPIA_MAIN (PROD_NAME, ROOT_NAME, INST_FILE, CORE_DUMP){

   /* Bind the user custom code to particular application objects */
   Principia->BindUserProcInit ( "SCENE_MAIN" , User_Scene_Init );
   Principia->BindUserProcMain ( "SCENE_MAIN" , User_Scene_Main );
   Principia->BindUserProcTerm ( "SCENE_MAIN" , User_Scene_Term );
   Principia->BindUserProcMain ( "PG_LEAF_01" , User_Page_FText );

   /* Execute the application main Principia script */
   Principia->ParseFile(CONF_FILE);

END_PRINCIPIA_MAIN}



@@@@ In the script, insert a reference to this page at the appropriate
@@@@ place where it should appear in the scene. Why do we do that? It
@@@@ is easier to control where in the scene rendering sequence will
@@@@ this page appear. We can force the display of the PG_LEAF_01 page
@@@@ completely from within the user code (in fact without even
@@@@ having to define it in the script) but it takes more work and is
@@@@ not required in most circumstances. We will revisit this in
@@@@ latter, more advanced demos though…


##define (PG_MAIN) as <PAGE_2A>
   #tag Anchor    =  (              - ,      - )
   #tag Region    =  (              - ,      - ,       - ,      - )
   #tag Element   =  (     F_BACK_01 ,      0 ,       0 ,      - )
   #tag Element   =  (   F_INTERFACE ,      0 ,       0 ,      - )
   #tag Element   =  (      BTN_TERM ,    900 ,       9 ,      - )
   #tag Element   =  (    PG_LEAF_01 ,    140 ,     180 ,      - )
   #tag Element   =  (    API_CURSOR ,      0 ,       0 ,      - )
   #tag Element   =  (      API_PJAY ,      0 ,       0 ,      - )
   #tag Element   =  (      API_KJAY ,      0 ,       0 ,      - )
```

D101C – User Code (3)

- The last thing left to do is to code the custom user procedure which will render the frame with background info. The code may look complex but it is actually straightforward
- We locate the parent page anchor position with *VArg
- We print the frame using the standard Principia Update() display method
- We generate the text we wish to print, the position where to print it relative to the anchor, and use the Textout2A->FreePrint() method to display it

```
void User_Page_FText (void* VArg)
BGN_ACTOR_CODE{

   /* Get the insertion point of the free text page as origin for this
       display */
   fXIns  = ((CX_Page2A*) VArg)->AnchorAbsX;
   fYIns  = ((CX_Page2A*) VArg)->AnchorAbsY;
   fYDel  = 20;

   /* Display the free frame and set the insertion point for text */
   aFreeFr->Update(fXIns, fYIns);
   fXIns += fYDel;
   fYIns += fYDel;

   /* Generate and display free text: Adapter Name */
   AdapterID = GraficInterface->ScreenA;
   sprintf(WrkStr,"ADAPTER: %s", Api_SysConfig-
       >AdapterID[AdapterID].Description);
   aFreeTx->FreePrint(fXIns, fYIns, WrkStr);
   fYIns += fYDel;

   /* Generate and display free text: OS */
   sprintf(WrkStr,"OS: %s", Api_SysConfig->InfoOSName->Str);
   aFreeTx->FreePrint(fXIns, fYIns, WrkStr);
   fYIns += fYDel;

   /* Generate and display free text: GS */
   sprintf(WrkStr,"GS: DirectX %d.%d", Api_SysConfig->InfoDXVerA,
       Api_SysConfig->InfoDXVerB);
   aFreeTx->FreePrint(fXIns, fYIns, WrkStr);
   fYIns += fYDel;

   /* Generate and display free text: Memory */
   sprintf(WrkStr,"FREE MEMORY: %d kb", Api_SysConfig->InfoMEMAvl);
   aFreeTx->FreePrint(fXIns, fYIns, WrkStr);
   fYIns += fYDel;

   /* Calculate free disk space on volume holding current directory */
   int k = Api_SysConfig->InfoDSDFree[0];
   for (int i=0; i<Api_SysConfig->InfoDSDn; i++) begin{
       if (Api_SysConfig->InfoDSDRoot[i]->Str[0]==Api_SysConfig->InfoEXEDIR-
       >Str[0])
       k = Api_SysConfig->InfoDSDFree[i];
   endif}

   /* Generate and display free text: Disk Space */
   sprintf(WrkStr,"FREE DISK: %d mb", k);
   aFreeTx->FreePrint(fXIns, fYIns, WrkStr);
   fYIns += fYDel;

END_ACTOR_CODE}
```

D101C – User Code (4)

- We have used two different methods for text display, but the paradigm carries over to many other rendering functions.
- The first method uses a statically constructed script for rendering, and minimal user code to update the data rendered
- The second method uses templates defined in the script, and does data management and rendering from within the user code
- The second method is more powerful but complex. A commercial application typically relies on a blend of both methods.

```
// Loose ends – we must not forget to define references for the template
// textout and frame, or connect them to the objects in the script.


DECLARE_GLOBL   CX_Frame2A*    aFreeFr   = NULL;


……


void User_Scene_Init (void* VArg)
BGN_ACTOR_CODE{

   /* Obtain pointers to the Principia objects defined in the script */
   aVarFPS = (SX_Variable*)  Principia->GetReferenceTo("VI_FPS");
   aVarVPF = (SX_Variable*)  Principia->GetReferenceTo("VI_VPF");
   aVarSPF = (SX_Variable*)  Principia->GetReferenceTo("VI_SPF");
   aFreeFr = (CX_Frame2A*)   Principia->GetReferenceTo("F_LEAF_01");
   aFreeTx = (CX_Textout2A*) Principia->GetReferenceTo("TX_FREE");

END_ACTOR_CODE}
```

D101C – Summary

- D101C showed different ways for defining fonts, printing text and integrating user code within the application. Again, highly complex tasks were achieved with few lines of code.
- There are many alternative pathways for achieving the same goal in the Principia framework. The most suitable one depends on the user requirement.
- To achieve the full power of Principia, the user needs to embrace its component framework and become thoroughly familiar with the reference manual. The demos will provide examples of many ways for getting the job done.


PrincipiaV.3.0.0
FPS: 100.0
VPF: 588
SPF: 21
ADAPTER: MOBILITY RADEON 9000
OS: Windows XP
GS: DirectX 9.0
FREE MEMORY: 233901 kb
FREE DISK: 6300 mb

D101D – Basic Audio

- **Add utility navigation buttons to the GUI**
  - Introduce the concept of encapsulating related component groups in self-contained script files
  - This avoids endless scripts and enables reusability of user code segments and script atoms
  - Not much has changed visually, but the concept of architectural organization is so important that it deserves a separate section of its own!
- **Introduce the basics of audio support**
- **Play sound when these buttons are clicked**

D101D – Script (1)

- Principia provides a rich set of complex audio capabilities for building commercial products. This demo shows most basic aspects of sound
- The user must create a sound interface and specify the audio features
- The interface has audio channels to which sounds are assigned
- Each channel can apply effects to its sounds, such as volume. These effects are controlled by means of variables from the script

```
@@@@  Variables controlling channel volume range from 0 to 100, i.e.
@@@@  silence to full blast volume. These variables can be connected
@@@@  to other Principia objects such as sliders to control volume.

##define <SYS_VARIABLE>   =  (  VS_LEV_MIN ,  VARTYPE_FLT ,    0.0  )
##define <SYS_VARIABLE>   =  (  VS_LEV_MAX ,  VARTYPE_FLT ,  100.0  )
##define <SYS_VARIABLE>   =  ( VS_LEV_MAIN ,  VARTYPE_FLT ,   50.0  )
##define <SYS_VARIABLE>   =  ( VS_LEV_USER ,  VARTYPE_FLT ,   50.0  )


@@@@  Define a basic CD-quality audio interface with two audio
@@@@  channels that have individual volume control variables.

##create (SOUND_INTERFACE) as <SOUND_INTERFACE>
   #tag ReqHaveSound      =  (     OPTIONAL )
   #tag ReqStereo         =  (     OPTIONAL )
   #tag Req3DSound        =  (       ABSENT )
   #tag ReqStrictFit      =  (          YES )
   #tag ReqSampleRate     =  (        22050 )
   #tag ReqSampleBits     =  (            8 )
   #tag ReqVolControl     =  (          YES )
   #tag SoundChannel      =  (     ACH_MAIN , VS_LEV_MAIN )
   #tag SoundChannel      =  (     ACH_USER , VS_LEV_USER )
```

D101D – Script (2)

- Akin to <SURFACE> objects that hold visual data, there are a variety of objects that hold audio data. <SOUND_2A> is the most basic of them.
- Like graphic objects, audio objects have definer prototypes that specify their properties. Here we use a predefined prototype from the included standards file.
- As most audio formats are proprietary, Principia V3 can read only WAV files.

```
@@@@  A standard predefined template for simple sounds, prescribing
@@@@  default memory storage for a non-streaming, non-looping sound
@@@@  that can be interrupted in all circumstances.

##define (SNDDEF_SIMPLE) as <DC_GAUDDEF0>
   #tag MemPool          =  (            - )
   #tag Streaming        =  (            - )
   #tag Looping          =  (            - )
   #tag Preemptable      =  (            1 )
   #tag Unstoppable      =  (            - )



@@@@  Two simple sounds are provided in the Common directory of the
@@@@  demo packages. The sounds are assigned to the MAIN audio channel.

##define (A_BEEP_01) as <SOUND_2A>
   #tag File         =  "../Demo_Common/Files_Media/GpSound_Tic01.wav"
   #tag Definer      =  (      SND_BEEP )
   #tag Channel      =  (      ACH_MAIN )

##define (A_BEEP_02) as <SOUND_2A>
   #tag File         = "../Demo_Common/Files_Media/GpSound_Tic02.wav"
   #tag Definer      =  (      SND_BEEP )
   #tag Channel      =  (      ACH_MAIN )
```

D101D – Script (3)

- There are three basic ways to play sound
  - By assigning sound reference to property tags where available (e.g. for buttons)
  - By using sound controller objects that DJ the audio in response to signals
  - Directly from the user code
- This demo shows only the first method, applied to a button object
- Note that different button actions can be mapped to different sounds

```
##define (BTN_UTIL_UNVA) as <BUTTON_2C>
   #tag State     =  (    NORMAL ,           - ,     NONE ,     NONE ,
       - )
   #method        =  (   ON_OVER ,      CH_CRS ,   X_OVER ,   NORMAL ,
       F_BTN_UNIV ,       NONE ,        - )
   #method        =  (   ON_LCLK ,      CH_NAV ,   X_UNVA ,   NORMAL ,
       F_BTN_UNIV ,  A_BEEP_01 ,        - )
   #tag Region    =  (  28 ,  60 )
   #tag Region    =  (  51 ,  46 )
   #tag Region    =  (  69 ,  58 )
   #tag Region    =  (  51 ,  73 )

##define (BTN_UTIL_UNVB) as <BUTTON_2C>
   #tag State     =  (    NORMAL ,           - ,     NONE ,     NONE ,
       - )
   #method        =  (   ON_OVER ,      CH_CRS ,   X_OVER ,   NORMAL ,
       F_BTN_UNIV ,       NONE ,        - )
   #method        =  (   ON_LCLK ,      CH_NAV ,   X_UNVB ,   NORMAL ,
       F_BTN_UNIV ,  A_BEEP_01 ,        - )
   #tag Region    =  (  28 ,  60 )
   #tag Region    =  (  51 ,  46 )
   #tag Region    =  (  69 ,  58 )
   #tag Region    =  (  51 ,  73 )

##define (BTN_UTIL_UNVC) as <BUTTON_2C>
   #tag State     =  (    NORMAL ,           - ,     NONE ,     NONE ,
       - )
   #method        =  (   ON_OVER ,      CH_CRS ,   X_OVER ,   NORMAL ,
       F_BTN_UNIV ,       NONE ,        - )
   #method        =  (   ON_LCLK ,      CH_NAV ,   X_UNVC ,   NORMAL ,
       F_BTN_UNIV ,  A_BEEP_01 ,        - )
   #tag Region    =  (  28 ,  60 )
   #tag Region    =  (  51 ,  46 )
   #tag Region    =  (  69 ,  58 )
   #tag Region    =  (  51 ,  73 )
```

D101D – Script (4)

- The GUI basic built so far – frame, buttons, diagnostics - are required by all demos.
- These are encapsulated in separate, common script file, that can be included in all application scripts
- This requires to divide the scene main page into pages leaving space to insert user-defined components. Here, these are the free text panel and the background image.
- Scripts can be nested and elements defined in a prior script can be overridden.
- The use of shorter, hierarchically organized scripts is a recommended practice.

```
@@@@ The section of script where the new action takes place
@@@@ is now really short. We do not need to mess with the GUI
@@@@ basics for the time being.

##parse "Modularized_GUI.cfg"


@@@@ These are the only components specific to DEMO01D.

##define (S_BACK_01) as <SURFACE>
   . . . . etc
##define (F_BACK_01) as <FRAME_2A>
   . . . . etc
##define (S_LEAF_01) as <SURFACE>
   . . . . etc
##define (F_LEAF_01) as <FRAME_2A>
   . . . . etc
##define (TX_FREE) as <TEXTOUT_2A>
   . . . . etc
##define (PG_LEAF_01) as <PAGE_2A>
   . . . . etc



@@@@ All we need to do is include the GUI component-holder pages
@@@@ at the appropriate layers of the user application.

##define (PG_MAIN) as <PAGE_2A>
   #tag Anchor    =  (             - ,      - )
   #tag Region    =  (             - ,      - ,       - ,      - )
   #tag Element   =  (     F_BACK_01 ,      0 ,       0 ,      - )
   #tag Element   =  (   PG_GUI_BASE ,      0 ,       0 ,      - )
   #tag Element   =  (    PG_LEAF_01 ,    140 ,     180 ,      - )
   #tag Element   =  (   PG_GUI_OVER ,      0 ,       0 ,      - )

##define (BK_MAIN) as <BOOK_2A>
   #tag StartOpen =  (           1 )
   #tag StateID   =  (  MAIN , - ,  - ,   0 ,   0 ,            1 )
   #tag Element   =  (  MAIN , - ,  - ,   0 ,   0 ,  PG_MAIN ,  1 )

##define (SCENE_MAIN) as <SCENE_2A>
   #tag VarStop          =  (      VS_TERM )
   #tag Element          =  (     - ,    - ,      BK_MAIN )

##run (SCENE_MAIN)
```

D101D – Code

- Exactly the same encapsulation can be performed for the custom code sections that manage the common performance display
- All we need to do is include the common code and invoke the procedures defined therein in the appropriate location of the scene user custom code.

```
// The only code needed is that specific
// to each individual demo application.

#define  ROOT_NAME  "..//Product_040701_Demos//Demo_1.01.D_BasicAudio//"
#define  CONF_FILE  "Demo_1.01.D_Main.cfg"
#include "Modularized_GUI.cpp”



void User_Scene_Init (void* VArg)
BGN_ACTOR_CODE{

   /* Execute initialization for the common performance display */
   Common_PerformanceDisplay_Init(VArg);

   . . . Etc.

END_ACTOR_CODE}



void User_Scene_Main (void* VArg)
BGN_ACTOR_CODE{

   /* Execute scene loop operations for common performance display */
   Common_PerformanceDisplay_Main(VArg);

 . . . Etc.

END_ACTOR_CODE}
```

D101D – Summary

- Illustrated the basic use of the audio interface and sound objects
- Illustrated the concept of script and code joint encapsulation into basic building blocks
- We can change the interface appearance by simply referencing another block than Module_GUI_V0. In fact, we will build two more improved versions of the GUI.

Principia V3.0.0 Demos

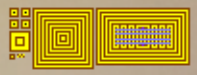
These buttons navigate the nearby performance display and pause/resume the application internal clock

VPF: 1k
FPF: 0k
PPF: 317

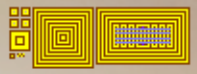
These buttons send general purpose signals that activate various demo objects later on.

ADAPTER: Radeon X1300 Series Secondary
OS: Windows 2000
GS: DirectX 9.0
FREE MEMORY: 2097152 kb
FREE DISK: 75169 mb

Rendering registration test: Memory Surface Implementation



Rendering registration test: Managed Texture Implementation



Principia application clock is HALTED.

This panel displays some basic surface rendering tests patterns and info.

These buttons navigate between different examples within the same demo (only one here, but this will change later ...)

D101E - A Better Common GUI

- **Taste being a matter of opinion, the one thing that can be said for the previous GUI is that is eye-catching.**
  - **"It is better to be noticed badly than ignored goodly"!**
  - **The round main viewport may be original, but it is not very suitable for maximizing the display area.**
  - **Redesign the GUI with a view-maximizing viewport, and add a host of useful features for later.**
  - **Encapsulate this GUI script and code in a modular form that can be included and used by all subsequent demos.**
- **Introduce many new useful concepts:**
  - **Surfaces and their types, formats and contents access.**
  - **Rendering diagnostics for development purposes.**
  - **Control buttons with different states.**
  - **Page-flipping controller.**
  - **Page functions ... and more.**

D101E - A Better Common GUI

- The GUI is encapsulated for easy inclusion in other applications and standard functions:
  - Application quit
  - Application module navigation
  - Application generic purpose buttons
  - Application performance reporting
  - Application pause
  - Globally available SigDEBUG signal is mapped to F1
- Notice how the GUI frame casts a shadow on whatever contents are below. This cool 3D effect is simply implemented using alpha transparency and a diffuse alpha mask.
- The encapsulated GUI script and the demo application script are included here in full.

D101E - Common GUI Script (1)

- Note that not all definition tags of an object need be specified. An omitted tag results to a default "common sense" value for the property specified, usually zero. In the demos, all tags are specified for reference purposes.

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@T.Valkov@(c)2000-2004@@@
@@                                                                                             @@
@@                                                                                             @@
@@    PRODUCT_040701_DEMOS                                                                     @@
@@    SECTION: PRELOAD STANDARD DEFINITIONS                                                    @@
@@                                                                                             @@
@@                                                                                             @@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


##parse "..\..\Principia_Library\Principia_StandardDefinitions_Core_v300.cfg"
##parse "..\..\Principia_Library\Principia_StandardDefinitions_Procedures_v300.cfg"
##parse "../Demo_Common/Files_Scripts/Module_INTERFACES.cfg"


@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@T.Valkov@(c)2000-2004@@@
@@                                                                                             @@
@@                                                                                             @@
@@    PRODUCT_040701_DEMOS                                                                     @@
@@    SECTION: APPLICATION AUDIO OBJECTS                                                       @@
@@                                                                                             @@
@@                                                                                             @@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


##define (A_BEEP_01) as <SOUND_2A>
   #tag File              =  "../Demo_Common/Files_Media/GpSound_Tic01.wav"
   #tag Definer           =  (      SND_BEEP )
   #tag Channel           =  (      ACH_MAIN )

##define (A_BEEP_02) as <SOUND_2A>
   #tag File              =  "../Demo_Common/Files_Media/GpSound_Tic02.wav"
   #tag Definer           =  (      SND_BEEP )
   #tag Channel           =  (      ACH_MAIN )
```

D101E - Common GUI Script (2)

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@T.Valkov@(c)2000-2005@@@
@@                                                                                              @@
@@                                                                                              @@
@@    PRODUCT_040701_DEMOS                                                                      @@
@@    SECTION: APPLICATION CURSOR                                                               @@
@@                                                                                              @@
@@                                                                                              @@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


##define (S_CRS_NORM) as <SURFACE>
   #tag File      =  "../Demo_Common/Files_Media/Cursors/Cursor_Norm_C.bmp"
   #tag Alpha     =  "../Demo_Common/Files_Media/Cursors/Cursor_Norm_X.bmp"
   #tag Definer   =  (       DEF_TEX )
   #tag LoadNow   =  (             1 )

##define (S_CRS_OVER) <SURFACE>
   #tag File      =  "../Demo_Common/Files_Media/Cursors/Cursor_Over_C.bmp"
   #tag Alpha     =  "../Demo_Common/Files_Media/Cursors/Cursor_Over_X.bmp"
   #tag Definer   =  (       DEF_TEX )
   #tag LoadNow   =  (             1 )

##define (F_CRS_NORM) as <FRAME_2A>
   #tag Image     =  (    S_CRS_NORM )
   #tag Offsets   =  (     -22 ,  -2 )

##define (F_CRS_OVER) as <FRAME_2A>
   #tag Image     =  (    S_CRS_OVER )
   #tag Offsets   =  (     -24 ,  -2 )

##define (API_CURSOR) as <CURSOR_2A>
   #method        =  (    EXE_HIDE   ,      CH_CRS ,       X_HIDE                )
   #method        =  (    EXE_SETIMG ,      CH_CRS ,       X_NORM ,   F_CRS_NORM )
   #method        =  (    EXE_SETIMG ,      CH_CRS ,       X_OVER ,   F_CRS_OVER )
```

D101E - Common GUI Script (3)

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@T.Valkov@(c)2000-2005@@@
@@                                                                                            @@
@@                                                                                            @@
@@    PRODUCT_040701_DEMOS                                                                    @@
@@    SECTION: APPLICATION INTERFACE FRAME AND BACKGROUNDS                                    @@
@@                                                                                            @@
@@                                                                                            @@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

##define (S_GUI_A) as <SURFACE>
   #tag Definer   =  (       DEF_TEX )
   #tag LoadNow   =  (             1 )
   #tag File      =  "../Demo_Common/Files_Media/GuiL2/Image_FrameA_C.bmp"
   #tag Alpha     =  "../Demo_Common/Files_Media/GuiL2/Image_FrameA_X.bmp"
. . .
##define (S_GUI_H) as <SURFACE>
   #tag Definer   =  (       DEF_TEX )
   #tag LoadNow   =  (             1 )
   #tag File      =  "../Demo_Common/Files_Media/GuiL2/Image_FrameH_C.bmp"
   #tag Alpha     =  "../Demo_Common/Files_Media/GuiL2/Image_FrameH_X.bmp"


##define (F_GUI_A) as <FRAME_2A>
   #tag Image     =  (       S_GUI_A )
. . .
##define (F_GUI_H) as <FRAME_2A>
   #tag Image     =  (       S_GUI_H )


@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@T.Valkov@(c)2000-2005@@@
@@                                                                                            @@
@@                                                                                            @@
@@    PRODUCT_040701_DEMOS                                                                    @@
@@    SECTION: APPLICATION BEHAVIOR CONTROLLERS                                               @@
@@                                                                                            @@
@@                                                                                            @@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

##define <SYS_VARIABLE>  =  (    VS_TERM ,   VARTYPE_LOG ,     - ,   CH_SYS ,   X_TERM )

##define (API_KJAY) as <KJAY_2A>
   #method        =  (    ON_KEYPRS  ,    CH_SYS ,     X_TERM  ,    ESC )
   #method        =  (    ON_KEYPRS  ,    CH_SYS ,    X_DEBUG  ,     F1 )

##define (API_PJAY) as <PJAY_2A>
   #method        =  (        ON_API ,      CH_CRS ,      X_NORM  )
   #method        =  (        ON_LFT ,      CH_CRS ,      X_HIDE  )
   #method        =  (        ON_RGT ,      CH_CRS ,      X_HIDE  )
   #method        =  (        ON_TOP ,      CH_CRS ,      X_HIDE  )
   #method        =  (        ON_BOT ,      CH_CRS ,      X_HIDE  )
```

D101E - Common GUI Script (4)

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@T.Valkov@(c)2000-2005@@@
@@                                                                                              @@
@@                                                                                              @@
@@    PRODUCT_040701_DEMOS                                                                      @@
@@    SECTION: APPLICATION TERMINATION BUTTON                                                   @@
@@                                                                                              @@
@@                                                                                              @@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

##define (S_BTN_TERM) as <SURFACE>
   #tag File      =  "../Demo_Common/Files_Media/GuiL2/Btn_Term_Over_C.bmp"
   #tag Alpha     =  "../Demo_Common/Files_Media/GuiL2/Btn_Term_Over_X.bmp"
   #tag Definer   =  (       DEF_TEX )
   #tag LoadNow   =  (             1 )

##define (F_BTN_TERM) as <FRAME_2A>
   #tag Image     =  (    S_BTN_TERM )

##define (BTN_TERM) as <BUTTON_2C>
   #tag State     =  (    NORMAL ,           - ,     NONE ,     NONE ,            - )
   #method        =  (   ON_OVER ,      CH_CRS ,   X_OVER ,   NORMAL ,   F_BTN_TERM ,       NONE ,        - )
   #method        =  (   ON_LCLK ,      CH_SYS ,   X_TERM ,   NORMAL ,   F_BTN_TERM ,  A_BEEP_01 ,        - )
   #tag Region    =  (  60 ,  30 )
   #tag Region    =  (  80 ,  10 )
   #tag Region    =  ( 100 ,  30 )
   #tag Region    =  (  80 ,  50 )

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@T.Valkov@(c)2000-2005@@@
@@                                                                                              @@
@@                                                                                              @@
@@    PRODUCT_040701_DEMOS                                                                      @@
@@    SECTION: APPLICATION EXAMPLE NAVIGATION BUTTONS                                           @@
@@                                                                                              @@
@@                                                                                              @@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

##define (S_BTN_LEFT) as <SURFACE>
   #tag File      =  "../Demo_Common/Files_Media/GuiL2/Btn_Util_Left_C.bmp"
   #tag Alpha     =  "../Demo_Common/Files_Media/GuiL2/Btn_Util_Left_X.bmp"
   #tag Definer   =  (       DEF_TEX )
   #tag LoadNow   =  (             1 )

##define (F_BTN_LEFT) as <FRAME_2A>
   #tag Image     =  (    S_BTN_LEFT )

##define (BTN_UTIL_LEFT) as <BUTTON_2C>
   #tag State     =  (    NORMAL ,           - ,     NONE ,     NONE ,            - )
   #method        =  (   ON_OVER ,      CH_CRS ,   X_OVER ,   NORMAL ,   F_BTN_LEFT ,       NONE ,        - )
   #method        =  (   ON_LCLK ,      CH_NAV ,   X_ULFT ,   NORMAL ,   F_BTN_LEFT ,  A_BEEP_01 ,        - )
   #tag Region    =  (   8 ,  30 )
```

D101E - Common GUI Script (5)

```
   #tag Region    =  (  30 ,   8 )
   #tag Region    =  (  30 ,  50 )


##define (S_BTN_RIGT) as <SURFACE>
   #tag File      =  "../Demo_Common/Files_Media/GuiL2/Btn_Util_Rigt_C.bmp"
   #tag Alpha     =  "../Demo_Common/Files_Media/GuiL2/Btn_Util_Rigt_X.bmp"
   #tag Definer   =  (       DEF_TEX )
   #tag LoadNow   =  (             1 )

##define (F_BTN_RIGT) as <FRAME_2A>
   #tag Image     =  (    S_BTN_RIGT )

##define (BTN_UTIL_RIGT) as <BUTTON_2C>
   #tag State     =  (    NORMAL ,           - ,     NONE ,     NONE ,           - )
   #method        =  (   ON_OVER ,      CH_CRS ,   X_OVER ,   NORMAL ,   F_BTN_RIGT ,       NONE ,        - )
   #method        =  (   ON_LCLK ,      CH_NAV ,   X_URGT ,   NORMAL ,   F_BTN_RIGT ,  A_BEEP_01 ,        - )
   #tag Region    =  (  10 ,  15 )
   #tag Region    =  (  33 ,  33 )
   #tag Region    =  (  10 ,  55 )


@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@T.Valkov@(c)2000-2005@@@
@@                                                                                              @@
@@                                                                                              @@
@@    PRODUCT_040701_DEMOS                                                                      @@
@@    SECTION: APPLICATION GENERAL PURPOSE UTILITY BUTTONS                                      @@
@@                                                                                              @@
@@                                                                                              @@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


##define (S_BTN_UNIV) as <SURFACE>
   #tag File      =  "../Demo_Common/Files_Media/GuiL2/Btn_Util_UNIV_C.bmp"
   #tag Alpha     =  "../Demo_Common/Files_Media/GuiL2/Btn_Util_UNIV_X.bmp"
   #tag Definer   =  (       DEF_TEX )
   #tag LoadNow   =  (             1 )

##define (F_BTN_UNIV) as <FRAME_2A>
   #tag Image     =  (    S_BTN_UNIV )

##define (BTN_UTIL_UNVA) as <BUTTON_2C>
   #tag State     =  (    NORMAL ,           - ,     NONE ,     NONE ,           - )
   #method        =  (   ON_OVER ,      CH_CRS ,   X_OVER ,   NORMAL ,   F_BTN_UNIV ,       NONE ,        - )
   #method        =  (   ON_LCLK ,      CH_NAV ,   X_UNVA ,   NORMAL ,   F_BTN_UNIV ,  A_BEEP_01 ,        - )
   #tag Region    =  (  13 ,  13 )
   #tag Region    =  (  43 ,  13 )
   #tag Region    =  (  43 ,  31 )
   #tag Region    =  (  13 ,  31 )
```

D101E - Common GUI Script (6)

```
##define (BTN_UTIL_UNVB) as <BUTTON_2C>
   #tag State     =  (    NORMAL ,           - ,     NONE ,     NONE ,           - )
   #method        =  (   ON_OVER ,      CH_CRS ,   X_OVER ,   NORMAL ,   F_BTN_UNIV ,       NONE ,        - )
   #method        =  (   ON_LCLK ,      CH_NAV ,   X_UNVB ,   NORMAL ,   F_BTN_UNIV ,  A_BEEP_01 ,        - )
   #tag Region    =  (  13 ,  13 )
   #tag Region    =  (  43 ,  13 )
   #tag Region    =  (  43 ,  31 )
   #tag Region    =  (  13 ,  31 )

##define (BTN_UTIL_UNVC) as <BUTTON_2C>
   #tag State     =  (    NORMAL ,           - ,     NONE ,     NONE ,           - )
   #method        =  (   ON_OVER ,      CH_CRS ,   X_OVER ,   NORMAL ,   F_BTN_UNIV ,       NONE ,        - )
   #method        =  (   ON_LCLK ,      CH_NAV ,   X_UNVC ,   NORMAL ,   F_BTN_UNIV ,  A_BEEP_01 ,        - )
   #tag Region    =  (  13 ,  13 )
   #tag Region    =  (  43 ,  13 )
   #tag Region    =  (  43 ,  31 )
   #tag Region    =  (  13 ,  31 )

##define (BTN_UTIL_UNVD) as <BUTTON_2C>
   #tag State     =  (    NORMAL ,           - ,     NONE ,     NONE ,           - )
   #method        =  (   ON_OVER ,      CH_CRS ,   X_OVER ,   NORMAL ,   F_BTN_UNIV ,       NONE ,        - )
   #method        =  (   ON_LCLK ,      CH_NAV ,   X_UNVD ,   NORMAL ,   F_BTN_UNIV ,  A_BEEP_01 ,        - )
   #tag Region    =  (  13 ,  13 )
   #tag Region    =  (  43 ,  13 )
   #tag Region    =  (  43 ,  31 )
   #tag Region    =  (  13 ,  31 )


@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@T.Valkov@(c)2000-2005@@@
@@                                                                                            @@
@@                                                                                            @@
@@    PRODUCT_040701_DEMOS                                                                    @@
@@    SECTION: APPLICATION FONTS GENERATION                                                   @@
@@                                                                                            @@
@@                                                                                            @@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


##define (FN_INFO14) as <FONT_3A>
   #tag Foundry   =  "Lucida Console"
   #tag Template  =  (       DEF_TEX )
   #tag Height    =  (            14 )
   #tag Weight    =  (           800 )
   #tag TextClr   =  (      CL_DKBLU )
   #tag BackClr   =  (             - )
   #tag AlignH    =  (    TXTINS_LFT )
   #tag AlignV    =  (    TXTINS_MID )
   #tag ExpandW   =  (             5 )
   #tag ExpandH   =  (             5 )
```

D101E - Common GUI Script (7)

```
##define (FN_INFO12) as <FONT_3A>
   #tag Foundry   =  "Verdana"
   #tag Template  =  (       DEF_TEX )
   #tag Height    =  (            14 )
   #tag Weight    =  (           800 )
   #tag TextClr   =  (      CL_DKRED )
   #tag BackClr   =  (             - )
   #tag AlignH    =  (    TXTINS_LFT )
   #tag AlignV    =  (    TXTINS_MID )
   #tag ExpandW   =  (             5 )
   #tag ExpandH   =  (             5 )

##define (FN_INFO10) as <FONT_3A>
   #tag Foundry   =  "Verdana"
   #tag Template  =  (       DEF_TEX )
   #tag Height    =  (            10 )
   #tag Weight    =  (           800 )
   #tag TextClr   =  (      CL_DKRED )
   #tag BackClr   =  (             - )
   #tag AlignH    =  (    TXTINS_LFT )
   #tag AlignV    =  (    TXTINS_MID )
   #tag ExpandW   =  (             5 )
   #tag ExpandH   =  (             5 )

##save (FN_INFO14) in ("..\Demo_Common\Files_Media\Fonts\Font_INFO14_C.bmp") alpha in ("..\Demo_Common\Files_Media\Fonts\Font_INFO14_X.bmp")
##save (FN_INFO12) in ("..\Demo_Common\Files_Media\Fonts\Font_INFO12_C.bmp") alpha in ("..\Demo_Common\Files_Media\Fonts\Font_INFO12_X.bmp")
##save (FN_INFO10) in ("..\Demo_Common\Files_Media\Fonts\Font_INFO10_C.bmp") alpha in ("..\Demo_Common\Files_Media\Fonts\Font_INFO10_X.bmp")

##delete (FN_INFO14)
##delete (FN_INFO12)
##delete (FN_INFO10)


@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@T.Valkov@(c)2000-2005@@@
@@                                                                                            @@
@@                                                                                            @@
@@    PRODUCT_040701_DEMOS                                                                    @@
@@    SECTION: APPLICATION FONTS RELOADING IF GENERATION IS DONE                              @@
@@                                                                                            @@
@@                                                                                            @@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

##define (FN_INFO14) as <FONT_3A>
   #tag ImgFile   =  "..\Demo_Common\Files_Media\Fonts\Font_INFO14_C.bmp"
   #tag AlfFile   =  "..\Demo_Common\Files_Media\Fonts\Font_INFO14_X.bmp"
   #tag XfdFile   =  "..\Demo_Common\Files_Media\Fonts\Font_INFO14_C.xfd"
   #tag Template  =  (       DEF_TEX )
   #tag AlignH    =  (    TXTINS_LFT )
   #tag AlignV    =  (    TXTINS_TOP )
```

D101E - Common GUI Script (8)

```
##define (FN_INFO12) as <FONT_3A>
   #tag ImgFile   =  "..\Demo_Common\Files_Media\Fonts\Font_INFO12_C.bmp"
   #tag AlfFile   =  "..\Demo_Common\Files_Media\Fonts\Font_INFO12_X.bmp"
   #tag XfdFile   =  "..\Demo_Common\Files_Media\Fonts\Font_INFO12_C.xfd"
   #tag Template  =  (       DEF_TEX )
   #tag AlignH    =  (    TXTINS_LFT )
   #tag AlignV    =  (    TXTINS_TOP )

##define (FN_INFO10) as <FONT_3A>
   #tag ImgFile   =  "..\Demo_Common\Files_Media\Fonts\Font_INFO10_C.bmp"
   #tag AlfFile   =  "..\Demo_Common\Files_Media\Fonts\Font_INFO10_X.bmp"
   #tag XfdFile   =  "..\Demo_Common\Files_Media\Fonts\Font_INFO10_C.xfd"
   #tag Template  =  (       DEF_TEX )
   #tag AlignH    =  (    TXTINS_LFT )
   #tag AlignV    =  (    TXTINS_TOP )

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@T.Valkov@(c)2000-2005@@@
@@                                                                                             @@
@@                                                                                             @@
@@    PRODUCT_040701_DEMOS                                                                     @@
@@    SECTION: APPLICATION DIAGNOSTIC INFORMATION DISPLAY COMPONENTS                           @@
@@                                                                                             @@
@@                                                                                             @@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


##define <SYS_VARIABLE>   =  (    VI_FPS ,   VARTYPE_FLT ,   0.0 )
##define <SYS_VARIABLE>   =  (    VI_GPS ,   VARTYPE_FLT ,   0.0 )

##define (TX_FPS) as <TEXTOUT_2A>
   #tag Plate     =  (          - ,      - ,      - )
   #tag Value     =  (    VI_FPS ,      - ,      - ,  "FPS:%6.1f" , FN_INFO14 )
##define (TX_GPS) as <TEXTOUT_2A>
   #tag Plate     =  (          - ,      - ,      - )
   #tag Value     =  (    VI_GPS ,      - ,      - ,  "GPS:%6.1f" , FN_INFO14 )

##define (PG_INFO_FPS) as <PAGE_2A>
   #tag Anchor    =  (              - ,      - )
   #tag Region    =  (              - ,      - ,       - ,      - )
   #tag Element   =  (        TX_FPS ,      0 ,       0 ,      - )
   #tag Element   =  (        TX_GPS ,      0 ,      18 ,      - )
```

D101E - Common GUI Script (9)

```
##define <SYS_VARIABLE>   =  (    VI_VPF ,   VARTYPE_INT ,     0 )
##define <SYS_VARIABLE>   =  (    VI_SPF ,   VARTYPE_INT ,     0 )
##define <SYS_VARIABLE>   =  (    VI_LPF ,   VARTYPE_INT ,     0 )

##define (TX_VPF) as <TEXTOUT_2A>
   #tag Plate     =  (          - ,     - ,     - )
   #tag Value     =  (    VI_VPF ,     - ,     - ,  "VPF:%6d" , FN_INFO14 )
##define (TX_SPF) as <TEXTOUT_2A>
   #tag Plate     =  (          - ,     - ,     - )
   #tag Value     =  (    VI_SPF ,     - ,     - ,  "SPF:%6d" , FN_INFO14 )
##define (TX_LPF) as <TEXTOUT_2A>
   #tag Plate     =  (          - ,     - ,     - )
   #tag Value     =  (    VI_LPF ,     - ,     - ,  "LPF:%6d" , FN_INFO14 )

##define (PG_INFO_VPF) as <PAGE_2A>
   #tag Anchor    =  (             - ,      - )
   #tag Region    =  (             - ,      - ,       - ,      - )
   #tag Element   =  (        TX_VPF ,      0 ,       0 ,      - )
   #tag Element   =  (        TX_SPF ,      0 ,      18 ,      - )
   #tag Element   =  (        TX_LPF ,      0 ,      36 ,      - )


##define <SYS_VARIABLE>   =  (    VI_TPF ,   VARTYPE_INT ,     0 )
##define <SYS_VARIABLE>   =  (    VI_MPF ,   VARTYPE_INT ,     0 )

##define (TX_TPF) as <TEXTOUT_2A>
   #tag Plate     =  (          - ,     - ,     - )
   #tag Value     =  (    VI_TPF ,     - ,     - ,  "TPF:%6d" , FN_INFO14 )
##define (TX_MPF) as <TEXTOUT_2A>
   #tag Plate     =  (          - ,     - ,     - )
   #tag Value     =  (    VI_MPF ,     - ,     - ,  "MPF:%6d" , FN_INFO14 )

##define (PG_INFO_TPF) as <PAGE_2A>
   #tag Anchor    =  (             - ,      - )
   #tag Region    =  (             - ,      - ,       - ,      - )
   #tag Element   =  (        TX_TPF ,      0 ,       0 ,      - )
   #tag Element   =  (        TX_MPF ,      0 ,      18 ,      - )


##define <SYS_VARIABLE>   =  (    VI_PUF ,   VARTYPE_INT ,     0 )
##define <SYS_VARIABLE>   =  (    VI_VUF ,   VARTYPE_INT ,     0 )

##define (TX_PUF) as <TEXTOUT_2A>
   #tag Plate     =  (          - ,     - ,     - )
   #tag Value     =  (    VI_PUF ,     - ,     - ,  "PUF:%6d" , FN_INFO14 )
##define (TX_VUF) as <TEXTOUT_2A>
   #tag Plate     =  (          - ,     - ,     - )
   #tag Value     =  (    VI_VUF ,     - ,     - ,  "VUF:%6d" , FN_INFO14 )

##define (PG_INFO_PUF) as <PAGE_2A>
   #tag Anchor    =  (             - ,      - )
   #tag Region    =  (             - ,      - ,       - ,      - )
   #tag Element   =  (        TX_PUF ,      0 ,       0 ,      - )
   #tag Element   =  (        TX_VUF ,      0 ,      18 ,      - )
```

D101E - Common GUI Script (10)

```
##define <SYS_VARIABLE>   =  (    VI_MGD ,   VARTYPE_INT ,     0 )
##define <SYS_VARIABLE>   =  (    VI_MGM ,   VARTYPE_INT ,     0 )
##define <SYS_VARIABLE>   =  (    VI_MGS ,   VARTYPE_INT ,     0 )

##define (TX_MGD) as <TEXTOUT_2A>
   #tag Plate     =  (          - ,     - ,     - )
   #tag Value     =  (    VI_MGD ,     - ,     - ,  "MGD:%6d" , FN_INFO14 )
##define (TX_MGM) as <TEXTOUT_2A>
   #tag Plate     =  (          - ,     - ,     - )
   #tag Value     =  (    VI_MGM ,     - ,     - ,  "MGM:%6d" , FN_INFO14 )
##define (TX_MGS) as <TEXTOUT_2A>
   #tag Plate     =  (          - ,     - ,     - )
   #tag Value     =  (    VI_MGS ,     - ,     - ,  "MGS:%6d" , FN_INFO14 )

##define (PG_INFO_MGD) as <PAGE_2A>
   #tag Anchor    =  (              - ,      - )
   #tag Region    =  (              - ,      - ,       - ,      - )
   #tag Element   =  (        TX_MGD ,      0 ,       0 ,      - )
   #tag Element   =  (        TX_MGM ,      0 ,      18 ,      - )
   #tag Element   =  (        TX_MGS ,      0 ,      36 ,      - )


##define <SYS_VARIABLE>   =  (    VI_MVD ,   VARTYPE_INT ,     0 )
##define <SYS_VARIABLE>   =  (    VI_MVM ,   VARTYPE_INT ,     0 )
##define <SYS_VARIABLE>   =  (    VI_MvS ,   VARTYPE_INT ,     0 )

##define (TX_MVD) as <TEXTOUT_2A>
   #tag Plate     =  (          - ,     - ,     - )
   #tag Value     =  (    VI_MVD ,     - ,     - ,  "MVD:%6d" , FN_INFO14 )
##define (TX_MVM) as <TEXTOUT_2A>
   #tag Plate     =  (          - ,     - ,     - )
   #tag Value     =  (    VI_MVM ,     - ,     - ,  "MVM:%6d" , FN_INFO14 )
##define (TX_MVS) as <TEXTOUT_2A>
   #tag Plate     =  (          - ,     - ,     - )
   #tag Value     =  (    VI_MVS ,     - ,     - ,  "MVS:%6d" , FN_INFO14 )

##define (PG_INFO_MVD) as <PAGE_2A>
   #tag Anchor    =  (              - ,      - )
   #tag Region    =  (              - ,      - ,       - ,      - )
   #tag Element   =  (        TX_MVD ,      0 ,       0 ,      - )
   #tag Element   =  (        TX_MVM ,      0 ,      18 ,      - )
   #tag Element   =  (        TX_MVS ,      0 ,      36 ,      - )


##define <SYS_VARIABLE>   =  (    VI_MGT ,   VARTYPE_INT ,     0 )
##define <SYS_VARIABLE>   =  (    VI_MvT ,   VARTYPE_INT ,     0 )

##define (TX_MGT) as <TEXTOUT_2A>
   #tag Plate     =  (          - ,     - ,     - )
   #tag Value     =  (    VI_MGT ,     - ,     - ,  "MGT:%7d" , FN_INFO14 )
##define (TX_MVT) as <TEXTOUT_2A>
   #tag Plate     =  (          - ,     - ,     - )
   #tag Value     =  (    VI_MVT ,     - ,     - ,  "MVT:%7d" , FN_INFO14 )
```

D101E - Common GUI Script (11)

```
##define (PG_INFO_MTT) as <PAGE_2A>
   #tag Anchor    =  (             - ,      - )
   #tag Region    =  (             - ,      - ,       - ,      - )
   #tag Element   =  (        TX_MGT ,      0 ,       0 ,      - )
   #tag Element   =  (        TX_MVT ,      0 ,      18 ,      - )


@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@T.Valkov@(c)2000-2005@@@
@@                                                                                            @@
@@                                                                                            @@
@@    PRODUCT_040701_DEMOS                                                                    @@
@@    SECTION: APPLICATION DIAGNOSTIC PANNEL COMPOSITION                                      @@
@@                                                                                            @@
@@                                                                                            @@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

##define (S_BTN_NEXTOVR) as <SURFACE>
   #tag File      =  "../Demo_Common/Files_Media/GuiL2/Btn_Nav_NextOvr_C.bmp"
   #tag Alpha     =  "../Demo_Common/Files_Media/GuiL2/Btn_Nav_NextOvr_X.bmp"
   #tag Definer   =  (       DEF_TEX )

##define (F_BTN_NEXTOVR) as <FRAME_2A>
   #tag Image     =  ( S_BTN_NEXTOVR )

##define (BTN_NAV_NEXT) as <BUTTON_2C>
   #tag State     =  (    NORMAL ,         NONE ,     NONE ,     NONE ,             - )
   #method        =  (   ON_OVER ,       CH_CRS ,   X_OVER ,   NORMAL , F_BTN_NEXTOVR ,       NONE ,        - )
   #method        =  (   ON_LCLK ,       CH_NAV ,   X_NEXT ,   NORMAL ,             - ,  A_BEEP_01 ,        - )
   #tag Region    =  (   3 ,  15 )
   #tag Region    =  (  31 ,   4 )
   #tag Region    =  (  58 ,  24 )
   #tag Region    =  (  25 ,  41 )

##define (S_BTN_HALTOVR) as <SURFACE>
   #tag File      =  "../Demo_Common/Files_Media/GuiL2/Btn_Nav_HaltOvr_C.bmp"
   #tag Alpha     =  "../Demo_Common/Files_Media/GuiL2/Btn_Nav_HaltOvr_X.bmp"
   #tag Definer   =  (       DEF_TEX )

##define (F_BTN_HALTOVR) as <FRAME_2A>
   #tag Image     =  ( S_BTN_HALTOVR )

##define (BTN_NAV_HALT) as <BUTTON_2C>
   #tag State     =  (    NORMAL ,         NONE ,     NONE ,     NONE ,             - )
   #tag State     =  (    PAUSED ,         NONE ,     NONE ,     NONE ,             - )
   #method        =  (   ON_OVER ,       CH_CRS ,   X_OVER ,   NORMAL , F_BTN_HALTOVR ,       NONE ,        - )
   #method        =  (   ON_LCLK ,       CH_CRS ,   X_PSSP ,   NORMAL ,             - ,          - ,        - )
   #method        =  (   ON_LCLK ,       CH_SYS ,   X_HALT ,   NORMAL ,             - ,  A_BEEP_01 ,        - )
   #method        =  (   ON_OVER ,       CH_CRS ,   X_OVER ,   PAUSED , F_BTN_HALTOVR ,       NONE ,        - )
   #method        =  (   ON_LCLK ,       CH_CRS ,   X_PSSN ,   PAUSED ,             - ,          - ,        - )
   #method        =  (   ON_LCLK ,       CH_SYS ,   X_RESU ,   PAUSED ,             - ,  A_BEEP_01 ,        - )
   #method        =  ( EXE_STATE ,       CH_CRS ,   X_PSSP ,   PAUSED )
   #method        =  ( EXE_STATE ,       CH_CRS ,   X_PSSN ,   NORMAL )
```

D101E - Common GUI Script (12)

```
   #tag Region    =  (   3 ,  15 )
   #tag Region    =  (  31 ,   4 )
   #tag Region    =  (  58 ,  24 )
   #tag Region    =  (  25 ,  41 )

##define (RT_INFO) as <ROTATOR_2A>
   #tag Anchor    =  (            - ,       - )
   #tag Rewind    =  (          YES )
   #tag Selector  =  (  BTN_NAV_NEXT ,     148 ,       -1 )
   #tag Selector  =  (  BTN_NAV_HALT ,     150 ,       47 )
   #tag Element   =  (   PG_INFO_FPS ,      16 ,       17 )
   #tag Element   =  (   PG_INFO_VPF ,      16 ,       17 )
   #tag Element   =  (   PG_INFO_TPF ,      16 ,       17 )
   #tag Element   =  (   PG_INFO_PUF ,      16 ,       17 )
   #tag Element   =  (   PG_INFO_MGD ,      16 ,       17 )
   #tag Element   =  (   PG_INFO_MVD ,      16 ,       17 )
   #tag Element   =  (   PG_INFO_MTT ,      16 ,       17 )
   #method        =  (      EXE_INCR , CH_NAV ,   X_NEXT )

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@T.Valkov@(c)2000-2005@@@
@@                                                                                            @@
@@                                                                                            @@
@@    PRODUCT_040701_DEMOS                                                                    @@
@@    SECTION: APPLICATION INCLUDABLE DISPLAY ELEMENTS                                        @@
@@                                                                                            @@
@@                                                                                            @@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

##define (PG_GUI_BASE) as <PAGE_2A>
   #tag Anchor    =  (            - ,       - )
   #tag Region    =  (            - ,       - ,        - ,       - )
   #tag Element   =  (       F_GUI_A ,       0 ,        0 ,       - )
   #tag Element   =  (       F_GUI_B ,     512 ,        0 ,       - )
   #tag Element   =  (       F_GUI_C ,       0 ,      704 ,       - )
   #tag Element   =  (       F_GUI_D ,     512 ,      640 ,       - )
   #tag Element   =  (       F_GUI_E ,     768 ,       64 ,       - )
   #tag Element   =  (       F_GUI_F ,       0 ,       64 ,       - )
   #tag Element   =  (       F_GUI_G ,       0 ,      192 ,       - )
   #tag Element   =  (       F_GUI_H ,     960 ,      576 ,       - )
   #tag Element   =  (       RT_INFO ,     800 ,      150 ,       - )
   #tag Element   =  (      BTN_TERM ,     870 ,       14 ,       - )
   #tag Element   =  ( BTN_UTIL_LEFT ,     914 ,      424 ,       - )
   #tag Element   =  ( BTN_UTIL_RIGT ,     955 ,      421 ,       - )
   #tag Element   =  ( BTN_UTIL_UNVA ,     945 ,      265 ,       - )
   #tag Element   =  ( BTN_UTIL_UNVB ,     945 ,      302 ,       - )
   #tag Element   =  ( BTN_UTIL_UNVC ,     945 ,      338 ,       - )
   #tag Element   =  ( BTN_UTIL_UNVD ,     945 ,      374 ,       - )

##define (PG_GUI_OVER) as <PAGE_2A>
   #tag Element   =  (    API_CURSOR ,       0 ,        0 ,       - )
   #tag Element   =  (      API_PJAY ,       0 ,        0 ,       - )
   #tag Element   =  (      API_KJAY ,       0 ,        0 ,       - )
```

D101E - Common GUI Script

Changes in the common GUI script

- Although most items in the common GUI script are already familiar from before, there are several new components and functions intended to support basic common needs across all demos.

Rotators

- One of the main visible changes is the diagnostic panel division in sections, only one of which is displayed at time.
- A new separate button is used to flip between the “pages” of the diagnostic panel. The new buttons are not built in the GUI, but have their own default frame for each state.
- This is implemented using a <ROTATOR> object, which action is to flip sequentially between components
- A <ROTATOR> object can contain anything – frames, books, buttons, and is an extremely powerful control

D101E - Common GUI Script

- There are several new buttons with increased complexity, that pays off in terms of power.
- Consider the <PAUSED> button:
  - The new button has two states – NORMAL and PAUSED.
  - In addition of sending signals to the system interface and the cursor, when clicked, the button sends signals to itself to switch state. These depend on the current button state.
  - A button can have many states, with different appearance and actions in every state.
- There are several new multi-state buttons
  - Buttons that pause the application and send signals to the rotator to flip the diagnostic display.
  - Buttons that send general purpose signals, such as flipping example rotators (green arrows).
- The common GUI has also a host of textouts to drive the new diagnostic outputs.

D101E - Common GUI Code (1)

- **The user code driving the GUI (primarily an update of the diagnostics data) can be also encapsulated in a common file and included in all other demos:**

```
#define PROD_NAME   NULL
#define PROD_VERS   NULL
#define INST_FILE   NULL
#define CORE_DUMP   "Files_Runtime//CoreDump.txt"

//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@T.Valkov(c)2000-2004@@//
//@  MODULE: Application Definition                                                            @//
//@  GROUPS: User custom code                                                                  @//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//

#define   THEN_PRINT_INFO_TEXT    GraficInterface->PrintText(pX, pY, aText->Str, aFont); pY+=nL;

#define   TEST_PRINT_INFO_TEXT_YN(Msg)  \
   if (CapsTest) then{                  \
      aText->Assign((Msg), "Y");        \
   }else{                               \
      aText->Assign((Msg), "N");        \
   endif} THEN_PRINT_INFO_TEXT;

//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@T.Valkov(c)2000-2004@@//
//@  MODULE: Global Variable Declaration                                                       @//
//@  GROUPS: User custom code                                                                  @//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//

DECLARE_GLOBL   SX_Variable*   aVarFPS   = NULL; // Variable containing performance data
DECLARE_GLOBL   SX_Variable*   aVarGPS   = NULL; // Variable containing performance data
DECLARE_GLOBL   SX_Variable*   aVarVPF   = NULL; // Variable containing performance data
DECLARE_GLOBL   SX_Variable*   aVarFPF   = NULL; // Variable containing performance data
. . .
DECLARE_GLOBL   SX_Variable*   aVarMGM   = NULL; // Variable containing performance data
DECLARE_GLOBL   SX_Variable*   aVarMGS   = NULL; // Variable containing performance data
DECLARE_GLOBL   SX_Variable*   aVarMVD   = NULL; // Variable containing performance data
DECLARE_GLOBL   SX_Variable*   aVarMVM   = NULL; // Variable containing performance data
DECLARE_GLOBL   SX_Variable*   aVarMVS   = NULL; // Variable containing performance data
DECLARE_GLOBL   SX_Variable*   aVarMGT   = NULL; // Variable containing performance data
DECLARE_GLOBL   SX_Variable*   aVarMVT   = NULL; // Variable containing performance data
```

D101E - Common GUI Code (2)

```
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@T.Valkov(c)2000-2004@@@//
//@                                                                                             @//
//@  MODULE: Common_PerformanceDisplay_Init()                                                   @//
//@  GROUPS: User custom code                                                                   @//
//@                                                                                             @//
//@  This module contains user-supplied instructions to be executed prior to scene rendering   @//
//@  ........................................................................................   @//
//@                                                                                             @//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//


void Common_PerformanceDisplay_Init (void* VArg)
BGN_ACTOR_CODE{

   /* Obtain pointers to the Principia objects defined in the script */
   aVarFPS  = (SX_Variable*)  Principia->GetReferenceTo("VI_FPS");
   aVarGPS  = (SX_Variable*)  Principia->GetReferenceTo("VI_GPS");
   aVarVPF  = (SX_Variable*)  Principia->GetReferenceTo("VI_VPF");
   aVarFPF  = (SX_Variable*)  Principia->GetReferenceTo("VI_FPF");
   aVarPPF  = (SX_Variable*)  Principia->GetReferenceTo("VI_PPF");
   aVarSPF  = (SX_Variable*)  Principia->GetReferenceTo("VI_SPF");
   aVarTPF  = (SX_Variable*)  Principia->GetReferenceTo("VI_TPF");
   aVarMPF  = (SX_Variable*)  Principia->GetReferenceTo("VI_MPF");
   aVarLPF  = (SX_Variable*)  Principia->GetReferenceTo("VI_LPF");
   aVarPUF  = (SX_Variable*)  Principia->GetReferenceTo("VI_PUF");
   aVarVUF  = (SX_Variable*)  Principia->GetReferenceTo("VI_VUF");
   aVarMGD  = (SX_Variable*)  Principia->GetReferenceTo("VI_MGD");
   aVarMGM  = (SX_Variable*)  Principia->GetReferenceTo("VI_MGM");
   aVarMGS  = (SX_Variable*)  Principia->GetReferenceTo("VI_MGS");
   aVarMVD  = (SX_Variable*)  Principia->GetReferenceTo("VI_MVD");
   aVarMVM  = (SX_Variable*)  Principia->GetReferenceTo("VI_MVM");
   aVarMVS  = (SX_Variable*)  Principia->GetReferenceTo("VI_MVS");
   aVarMGT  = (SX_Variable*)  Principia->GetReferenceTo("VI_MGT");
   aVarMVT  = (SX_Variable*)  Principia->GetReferenceTo("VI_MVT");

   /* Activate the Principia general debug signal */
   Api_SigDebug = new PX_Signal("CH_SYS", "X_DEBUG");

END_ACTOR_CODE}
```

D101E - Common GUI Code (3)

```
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@T.Valkov(c)2000-2004@@@//
//@                                                                                          @//
//@  MODULE: Common_PerformanceDisplay_Main()                                                @//
//@  GROUPS: User custom code                                                                @//
//@                                                                                          @//
//@  This module contains user-supplied instructions to be executed during scene rendering..   @//
//@  .......................................................................................   @//
//@                                                                                          @//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//

void Common_PerformanceDisplay_Main (void* VArg)
BGN_ACTOR_CODE{

   /* Update the value of the display variables with the appropriate internal metrics */
   aVarFPS->Assign(Api_PerfMonitor->PerfRepFPS);
   aVarGPS->Assign(Api_PerfMonitor->PerfRepGPS);
   aVarVPF->Assign(Api_PerfMonitor->PerfRepVPF/1000);
   aVarFPF->Assign(Api_PerfMonitor->PerfRepFPF/1000);
   aVarPPF->Assign(Api_PerfMonitor->PerfRepPPF);
   aVarSPF->Assign(Api_PerfMonitor->PerfRepSPF);
   aVarTPF->Assign(Api_PerfMonitor->PerfRepTPF);
   aVarMPF->Assign(Api_PerfMonitor->PerfRepMPF);
   aVarLPF->Assign(Api_PerfMonitor->PerfRepLPF);
   aVarPUF->Assign(flt(Api_PerfMonitor->PerfRepPUF/1024)/1024.0f);
   aVarVUF->Assign(flt(Api_PerfMonitor->PerfRepVUF/1024)/1024.0f);
   aVarMGD->Assign(flt(Api_PerfMonitor->PerfMGD/1024)/1024.0f);
   aVarMGM->Assign(flt(Api_PerfMonitor->PerfMGM/1024)/1024.0f);
   aVarMGS->Assign(flt(Api_PerfMonitor->PerfMGS/1024)/1024.0f);
   aVarMVD->Assign(flt(Api_PerfMonitor->PerfMVD/1024)/1024.0f);
   aVarMVM->Assign(flt(Api_PerfMonitor->PerfMVM/1024)/1024.0f);
   aVarMVS->Assign(flt(Api_PerfMonitor->PerfMVS/1024)/1024.0f);
   aVarMGT->Assign(flt(Api_PerfMonitor->PerfMGT/1024)/1024.0f);
   aVarMVT->Assign(flt(Api_PerfMonitor->PerfMVT/1024)/1024.0f);

END_ACTOR_CODE}
```

- **Note that instead of calling ::FreePrint to display the text, the common GUI updates variables connected to each textout. This is a much more robust method.**

D101E – Using the Common GUI

- **Using the new GUI anywhere is now a simple matter of one line of script include in our demo:**

```
##parse "../Demo_Common/Files_Scripts/Module_GUIDEM.cfg"
```

- **… and include+two calls in our user code !!!**

```
#include              "../Demo_Common/Files_Code/Module_GUIDEM.cpp"


void User_Scene_Init (void* VArg)
BGN_ACTOR_CODE{

   /* Execute initialization for the common performance display components */
   Common_PerformanceDisplay_Init(VArg);

   /* Local instructions */
   . . .

END_ACTOR_CODE}



void User_Scene_Main (void* VArg)
BGN_ACTOR_CODE{

   /* Execute scene loop operations for the common performance display */
   Common_PerformanceDisplay_Main(VArg);

 /* Local instructions */
   . . .

END_ACTOR_CODE}
```

D101E – Demo Script (1)

- **Our demo script can now focus at the new task at hand without being cluttered with the GUI.**

```
##define (S_REG03_S) as <SURFACE>
   #tag Definer   =  (       DEF_S2D )
   #tag File      =  "../Demo_Common/Files_Media/Calibration/Image_Reg03.bmp"
   #tag LoadNow   =  (             1 )
. . .
##define (S_REG30_T) as <SURFACE>
   #tag Definer   =  (       DEF_TEX )
   #tag File      =  "../Demo_Common/Files_Media/Calibration/Image_Reg30.bmp"
   #tag LoadNow   =  (             1 )

##define (F_REG03_S) as <FRAME_2A>
   #tag Image     =  (     S_REG03_S )
. . .
##define (F_REG30_T) as <FRAME_2A>
   #tag Image     =  (     S_REG30_T )

##define (PG_RTEST_S) as <PAGE_2A>
   #tag Anchor    =  (             - ,      - )
   #tag Region    =  (             - ,      - ,       - ,      - )
   #tag Element   =  (     F_REG05_S ,      0 ,       0 ,      - )
   #tag Element   =  (     F_REG05_S ,      0 ,       6 ,      - )
   #tag Element   =  (     F_REG05_S ,      6 ,       0 ,      - )
   #tag Element   =  (     F_REG05_S ,      6 ,       6 ,      - )
   #tag Element   =  (     F_REG11_S ,      0 ,      12 ,      - )
   #tag Element   =  (     F_REG32_S ,     12 ,       0 ,      - )
   #tag Element   =  (     F_REG30_S ,     45 ,       0 ,      - )
   #tag Element   =  (     F_REG03_S ,      0 ,      24 ,      - )
   #tag Element   =  (     F_REG04_S ,      4 ,      24 ,      - )

##define (PG_RTEST_T) as <PAGE_2A>
   #tag Anchor    =  (             - ,      - )
   #tag Region    =  (             - ,      - ,       - ,      - )
   #tag Element   =  (     F_REG05_T ,      0 ,       0 ,      - )
   #tag Element   =  (     F_REG05_T ,      0 ,       6 ,      - )
   #tag Element   =  (     F_REG05_T ,      6 ,       0 ,      - )
   #tag Element   =  (     F_REG05_T ,      6 ,       6 ,      - )
   #tag Element   =  (     F_REG11_T ,      0 ,      12 ,      - )
   #tag Element   =  (     F_REG32_T ,     12 ,       0 ,      - )
   #tag Element   =  (     F_REG30_T ,     45 ,       0 ,      - )
   #tag Element   =  (     F_REG03_T ,      0 ,      24 ,      - )
   #tag Element   =  (     F_REG04_T ,      4 ,      24 ,      - )
```

D101E – Demo Script (2)

- **After defining the test pattern images, the script defines their background and assembles the scene (including the GUI top and bottom pages).**

```
##define (S_LEAF_01) as <SURFACE>
   #tag File      =  "../Demo_1.01.E_BasicGUI/Files_Media/Image_MatteBack-C.bmp"
   #tag Alpha     =  "../Demo_1.01.E_BasicGUI/Files_Media/Image_MatteBack-X.bmp"
   #tag Definer   =  (       DEF_TEX )
   #tag LoadNow   =  (             1 )

##define (F_LEAF_01) as <FRAME_2A>
   #tag Image     =  (     S_LEAF_01 )

##define (TX_FREE) as <TEXTOUT_2A>
   #tag Plate     =  (         - ,     - ,     - )
   #tag Value     =  (         - ,     - ,     - ,            - ,   FN_INFO12 )

##define (PG_LEAF_01) as <PAGE_2A>
   #tag Anchor    =  (             - ,      - )
   #tag Region    =  (             - ,      - ,       - ,      - )

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@T.Valkov@(c)2000-2005@@@
@@    PRODUCT_040701_DEMOS                                                                    @@
@@    SECTION: APPLICATION SCENE COMPONENTS                                                   @@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

##define (PG_MAIN) as <PAGE_2A>
   #tag Anchor    =  (             - ,      - )
   #tag Region    =  (             - ,      - ,       - ,      - )
   #tag Element   =  (     F_BACK_01 ,      0 ,       0 ,      - )
   #tag Element   =  (   PG_GUI_BASE ,      0 ,       0 ,      - )
   #tag Element   =  (    PG_LEAF_01 ,    140 ,     180 ,      - )
   #tag Element   =  (   PG_GUI_OVER ,      0 ,       0 ,      - )

##define (BK_MAIN) as <BOOK_2A>
   #tag StartOpen =  (            1 )
   #tag StateID   =  (         MAIN ,     - ,     - ,       0 ,      0 ,                  1 )
   #tag Element   =  (         MAIN ,     - ,     - ,       0 ,      0 ,    PG_MAIN ,    1 )

##define (SCENE_MAIN) as <SCENE_2A>
   #tag VarStop           =  (      VS_TERM )
   #tag Element           =  (     - ,    - ,      BK_MAIN )
```

D101E – Demo Code (1)

- **The demo code likewise uses includes to take care of the common GUI tasks, and focuses on the display of the test pattern panel.**

```
DECLARE_GLOBL   CX_Frame2A*    aFreeFr   = NULL; // Frame background for free text
DECLARE_GLOBL   CX_Textout2A*  aFreeTx   = NULL; // Working textout for free text display
DECLARE_GLOBL   CX_Page2A*     aPgTstS   = NULL; // Registration test page for surfaces
DECLARE_GLOBL   CX_Page2A*     aPgTstT   = NULL; // Registration test page for textures
DECLARE_GLOBL   DWORD          AdapterID = 0x00; // Adapter ID index
DECLARE_GLOBL   int            fXIns     = 0x00; // Free text initial insertion point
DECLARE_GLOBL   int            fYIns     = 0x00; // Free text initial insertion point
DECLARE_GLOBL   int            fYDel     =   20; // Free text interline spacing
DECLARE_GLOBL   LONGSTRING     WrkStr    = "\0"; // Working string for free text display


//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@T.Valkov(c)2000-2004@@@//
//@                                                                                            @//
//@  MODULE: User_Scene_Init()                                                                 @//
//@  GROUPS: User custom code                                                                  @//
//@                                                                                            @//
//@  This module contains user-supplied instructions to be executed prior to scene rendering   @//
//@  .......................................................................................   @//
//@                                                                                            @//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@//


void User_Scene_Init (void* VArg)
BGN_ACTOR_CODE{

   /* Execute initialization for the common performance display components */
   Common_PerformanceDisplay_Init(VArg);

   /* Obtain pointers to the Principia objects defined in the user script */
   aFreeFr = (CX_Frame2A*)   Principia->GetReferenceTo("F_LEAF_01");
   aFreeTx = (CX_Textout2A*) Principia->GetReferenceTo("TX_FREE");
   aPgTstS = (CX_Page2A*)    Principia->GetReferenceTo("PG_RTEST_S");
   aPgTstT = (CX_Page2A*)    Principia->GetReferenceTo("PG_RTEST_T");

END_ACTOR_CODE}
```

D101E – Demo Code (2)

- A page function is used to display the pattern and informational data (instead of doing it in the script). Note how anchor positions are obtained.

```
void User_Page_FText (void* VArg)
BGN_ACTOR_CODE{

   /* Get the insertion point of the free text page as origin for this display */
   fXIns  = ((CX_Page2A*) VArg)->AnchorAbsX;
   fYIns  = ((CX_Page2A*) VArg)->AnchorAbsY;
   fYDel  = 20;

   /* Display the free frame and set the insertion point for text */
   aFreeFr->Update(fXIns, fYIns);
   fXIns += fYDel;
   fYIns += fYDel;

   /* Generate and display free text: Adapter Name */
   AdapterID = GraficInterface->ScreenA;
   sprintf(WrkStr,"ADAPTER: %s", Api_SysConfig->AdapterID[AdapterID].Description);
   aFreeTx->FreePrint(fXIns, fYIns, WrkStr);
   fYIns += fYDel;

   /* Generate and display free text: OS */
   sprintf(WrkStr,"OS: %s", Api_SysConfig->InfoOSName->Str);
   aFreeTx->FreePrint(fXIns, fYIns, WrkStr);
   fYIns += fYDel;

   /* Generate and display free text: GS */
   sprintf(WrkStr,"GS: DirectX %d.%d", Api_SysConfig->InfoDXVerA, Api_SysConfig->InfoDXVerB);
   aFreeTx->FreePrint(fXIns, fYIns, WrkStr);
   fYIns += fYDel;

   /* Generate and display free text: Memory */
   sprintf(WrkStr,"FREE MEMORY: %d kb", Api_SysConfig->InfoMEMAvl);
   aFreeTx->FreePrint(fXIns, fYIns, WrkStr);
   fYIns += fYDel;

   /* Calculate free disk space on volume holding current directory */
   int k = Api_SysConfig->InfoDSDFree[0];
   for (int i=0; i<Api_SysConfig->InfoDSDn; i++) begin{
       if (Api_SysConfig->InfoDSDRoot[i]->Str[0]==Api_SysConfig->InfoEXEDIR->Str[0])
       k = Api_SysConfig->InfoDSDFree[i];
   endif}
```

D101E – Demo Code (2)

- **In this demo, we desire to display components from within the user code. This is done with calls to ::Update() and ::FreePrint() in the page fcn.**

```
   /* Generate and display free text: Disk Space */
   sprintf(WrkStr,"FREE DISK: %d mb", k);
   aFreeTx->FreePrint(fXIns, fYIns, WrkStr);
   fYIns += fYDel;   fYIns += fYDel;

   /* Display the registration pattern test pages */
   strcpy(WrkStr,"Rendering registration test: Memory Surface Implementation");
   aFreeTx->FreePrint(fXIns, fYIns, WrkStr);      fYIns += fYDel;
   aPgTstS->Update(fXIns, fYIns);                 fYIns += (40);

   /* Display the registration pattern test pages */
   strcpy(WrkStr,"Rendering registration test: Managed Texture Implementation");
   aFreeTx->FreePrint(fXIns, fYIns, WrkStr);
   fYIns += fYDel;
   aPgTstT->Update(fXIns, fYIns);
   fYIns += (50);

   /* Display the application status */
   if (Principia->Paused) then{
      strcpy(WrkStr,"Principia application clock is HALTED.");
   }else{
      strcpy(WrkStr,"Principia application clock is running.");
   endif}
   aFreeTx->FreePrint(fXIns, fYIns, WrkStr);

END_ACTOR_CODE}

BGN_PRINCIPIA_MAIN (PROD_NAME, PROD_VERS, ROOT_NAME, INST_FILE, CORE_DUMP){

   /* Bind the user custom code to particular application objects */
   Principia->BindUserProcInit ( "SCENE_MAIN" , User_Scene_Init );
   Principia->BindUserProcMain ( "SCENE_MAIN" , User_Scene_Main );
   Principia->BindUserProcTerm ( "SCENE_MAIN" , User_Scene_Term );
   Principia->BindUserProcMain ( "PG_LEAF_01" , User_Page_FText );

   /* Execute the application main Principia script */
   Principia->ParseFile(CONF_FILE);

END_PRINCIPIA_MAIN}
```

D101E – Surfaces and Textures

- **Virtually all graphic applications use flat (2D) images for GUI elements, buttons, user interaction dialogs...etc.**
  - Principia provides the CX_Frame** series of control components to display such images
- **There are two common ways for rendering flat 2D frames:**
  - Direct memory copy (blit) of the frame image to the display buffer. This does not involve 3D rendering, and is 100% accurate, but does not provide the full 3D GPU special effects such as transparency, alpha mapping or blending.
  - This method will be referred to as SURFACE-based, because it uses GPU plain memory surfaces (and platform objects).
  - Render an artificially positioned 3D rectangle on the GPU with the image as mapped texture. This is required to render transparent frames or make use of GPU-based image processing effects.
  - This method will be referred to as TEXTURE-based, because it uses GPU texture memory (and platform objects).

D101E – Surfaces and Textures

- **Besides flat frame display, surfaces/textures have a wide variety of uses in multimedia:**
  - Sprite images
  - Textures for 3D geometries
  - Environment maps for 3D geometries
  - Vertex data maps
  - Numeric data maps
  - And more...
- **Principia combines GPU surfaces and textures in a single fundamental graphic component:**
  - Implemented in the GX_Surface object
  - Created using the <SURFACE> tag in script

D101E – Surfaces and Textures

- In fact, the Principia <SURFACE> has many descriptive characteristics for various uses:
  - Driver surface type (surface, texture, cube, volume).
  - Format for encoding data (A8R8G8B8, L16 ...etc).
  - Memory location code (System, Device, Managed).
  - Usage hint code (RT, Depth-Stencil, dynamic ...etc).
  - Mip levels depth for mipmapping
  - Intrinsic filter codes for driver access
  - Locking code for DMA access.
  - And much more ...
- These properties are encapsulated in a surface definer object. Principia spares the user from the internal complexity of their management.

D101E – Surfaces and Textures

- **The choice of surface format and file form is closely determined by the intended use:**
  - **Regular texturing of 3D primitives: X8/A8R8G8B8 stored in BMP/PNG files.**
  - **Low-precision numeric data: L8/X8R8G8B8 8-bit grayscale stored in BMP/PNG files.**
  - **High-precision numeric data: L16/X16R16G16B16 16-bit/32-bit grayscale stored in PNG files.**
  - **Specialized shader usage in render maps: G16B16 or F32 procedural formats channel-stored in PNG files.**
- **Principia makes format and file form management a simple matter of entering the desired format in the script!**

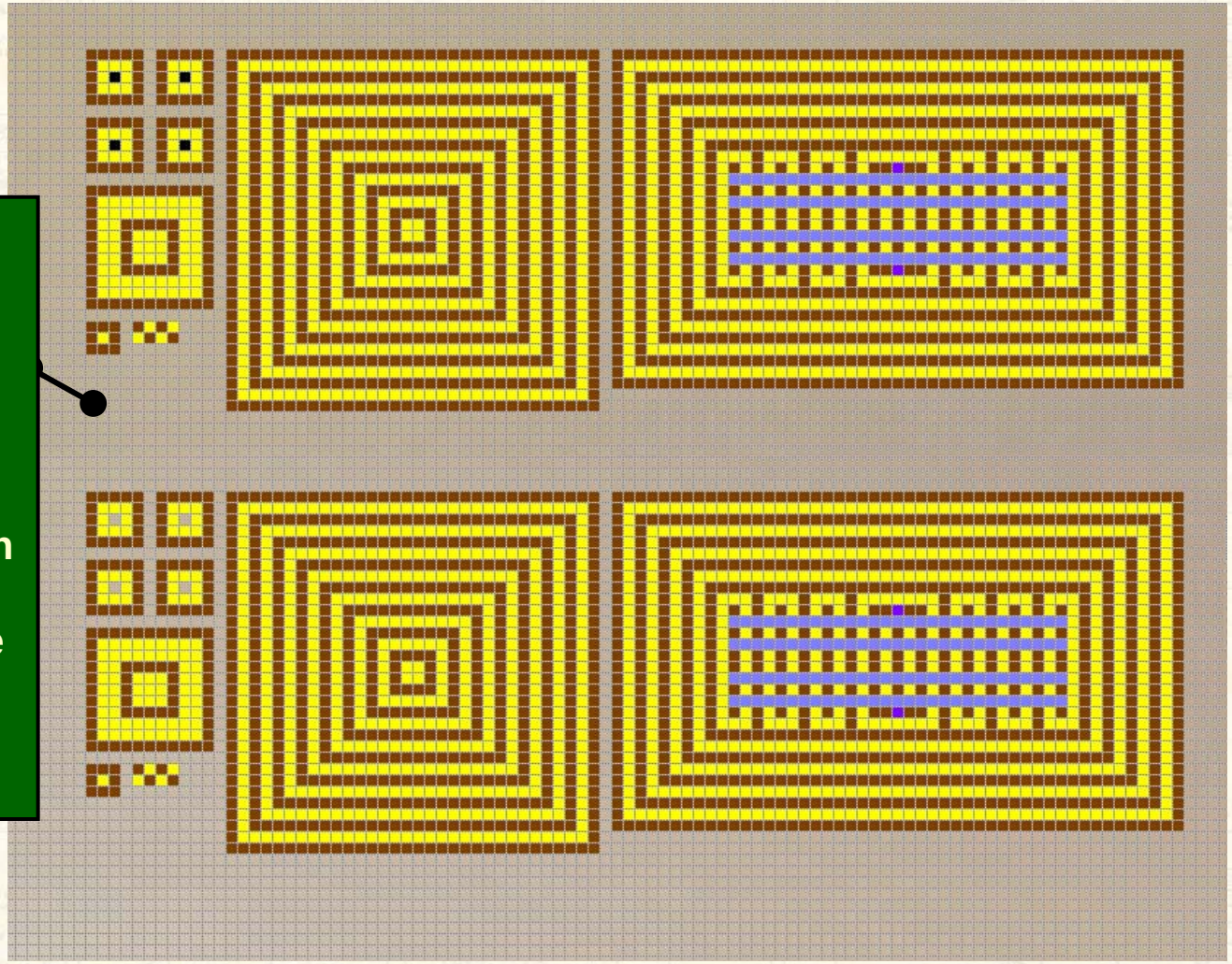
D101E – Surfaces and Textures

- **The surface type used is controlled by the definer given to the script when creating a <SURFACE> image object.**
  - In the standard library include, <DEF_TEX> specifies a TEXTURE type, while <DEF_S2D> specifies a linear memory SURFACE type.
- **The two funny patterns displayed in the free text panel.**
  - Renditions of several small images at carefully chosen pixel positions.
  - Top/Bottom patterns use the DEF_S2D and DEF_TEX definers respectively.
- **Many applications require single-pixel accuracy when positioning and rendering of flat images.**
  - Different environments may have different texture sampling and pixel coordinate round-off conventions, that complicate the use of DEF_TEX definers for rendering flat images as mapped textures.
  - Principia makes this considerable complexity transparent to the user. All you need to do is provide a definer and decide where to display the image. When porting to new environments (e.g. other than Windows/DirectX), the registration test provided in this demo will reveal any problems quickly.
  - Images with 1-pixel thickness may not be rendered accurately using the TEXTURE method. These images should be rendered using the SURFACE method or have an additional row of pixels added.

D101E – Test Registration Patterns

At high pixel magnification level, the correct test pattern should look as shown here.

The lower pattern uses textures. Can you spot the minor difference from transparent pixels?



D101E – Surfaces and Textures

- **Direct access of surface (pixel) data**
  - **Must be preceded by ::StartDmaMode() call**
  - **Must be terminated by ::StopDmaMode() call**
  - **Individual pixel access by ::Wrte/ReadPixel() calls**
    - Value: raw pixel, float, ARGB dwords, ARGB floats
    - Address mode: IJ or UV
    - Function automatically interprets the format
  - **For faster access to entire contents, use the GX_Surface::Import/ExportData() call, which returns SX_Grid1R color or float objects.**
  - **For truly fast access by knowledge users, the DMA calls expose the GX_Surface::GpuData pointer.**
- **Surface cannot be used by driver until DMA ends. Principia access methods require DMA.**

D101E – Principia Timers

- **The Principia system interface provides several controllable, configurable user clocks, that provide timing control for the application built.**
  - **Clocks and timers are configured during the creation of the system interface.**
  - **The clock state, time, units ...etc can be accessed from within the user program via the Principia API.**
- **A very basic function is to pause the main application clock. This suspends the user application without interrupting interaction**
  - **Here, this is done by sending the right signals to the user interface from a control button. We also poll the API paused state to inform the user accordingly.**

D101E – Diagnostics (1)

- **The new diagnostic panel features a seemingly bewildering array of performance indicators.**
  - They will all prove necessary when tuning and optimizing demanding commercial-grade applications (or running on older GPUs).
- **Per-Second Counters**
  - FPS: Full application frames per second counter
  - GPS: Graphing frames per second. This is the what the application FPS would be if it did nothing but render the image. The difference between FPS and GPS measures the overhead of the user game code
- **Per-Frame Counters, GPU Render Load**
  - VPF: Vertices rendered per frame. Key 3D metric.
  - FPF: Triangle faces rendered per frame. Key 3D metric.
  - PPF: Primitive render calls per frame. Key 3D metric.

D101E – Diagnostics (2)

Per-Frame Counters, GPU/CPU Operations Load

- SPF: GPU state changes per frame. Typically associated with materials. Since state changes are time-expensive, a good design minimizes SPF.
- TPF: Transformation changes per frame. Significant when running skeletal animation models replicated across many objects. Includes geometry transformations, material and lighting changes.
- MPF: Matrix multiplications per frame. Important when running complex skeletal models with multiple levels.
- LPF: GPU memory locks per frame. Also an expensive operation, the fewer the better. Typically, associated with dynamic changes of graphic content, and with direct client application manipulation of graphic data.

Per-Frame Counters, Client Application GPU DMA Flow

- PUF: Pixel application flow rate in kB per frame. When the user generates or moves image data on the fly, this counter measures the pixels moved per frame in DMA mode. This is not a count of pixels rendered by the GPU.
- VUF: Vertex application flow rate in kB per frame. When the user generates or moves vertices on the fly, this counter measures the vertices moved per frame in DMA mode. This is not a count of vertices rendered by the GPU.

D101E – Diagnostics (3)

- **Memory Load –Graphic Buffers**
  - MGD: Requested memory load for the GPU in kB load into device memory. This includes the primary display, swap-chain and depth buffer memory.
  - MGM: Requested memory load for the managed pool in terms of kB attempted to load. The GS – if available - manages this load between GPU, AGP and system in a way that the user cannot control.
  - MGS: Requested memory load for the system. This memory space is not particularly suitable for fast rendering but it is easily accessible to the user.
- **Memory Load – Vertex Buffers**
  - MVD: Vertex buffer kB load equivalent to MGD
  - MVM: Vertex buffer kB load equivalent to MGM
  - MVS: Vertex buffer kB load equivalent to MGS
- **Memory Load – Totals**
  - MGT: Total memory load from graphic buffers
  - MVT: Total memory load from vertex buffers
  - MAT: Total memory load from allocated API data

D101E – Diagnostics (4)

- **Where does all this data come from?**
  - Api_SysConfig() is an automatic global Principia component that holds extensive information on the current hardware and software platform.
  - Api_PerfMonitor() is an automatic global Principia component created in DIAGNOSTIC mode (see next), that holds myriads of performance counters on many things, ranging from GPU requests flow to box tests ...etc.
  - The graphic interface has its own internal counters (not polled here) used for frame rate stabilization and internal GPU request flow optimization.
- **There are two instances of internal frame-basis counters from Api_PerfMonitor() – local and reported.**
  - The local counter has the stats for the last frame rendered. The reported counter is sampled at 500ms intervals. The GUI display uses the reported counter. The local counter changes too fast to be displayable in a GUI.
- **By including the common .cfg and .cpp files of the diagnostic display, any application can examine these metrics, and identify performance bottlenecks such as excessive locks, swaps or DMA moves.**

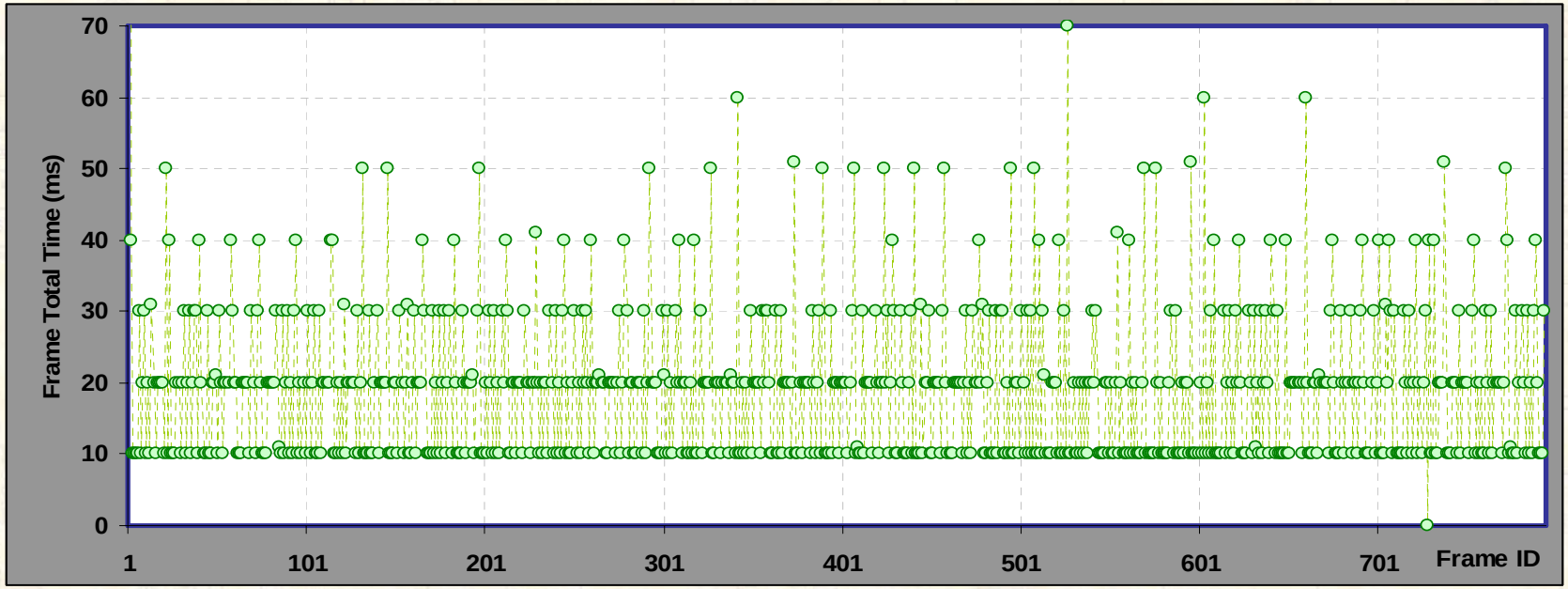
D101E – Diagnostic Levels (1)

- The output of diagnostics is controlled by flags that can be defined in the user application. Much of the data generated by diagnostic is saved in the core dump file, that must be defined as shown.
- The PCM_DEVELOPMENT_DIAG flag acts as master controller for diagnostic info. If the flag is 0 (default), diagnostic code is not compiled and there is no overhead whatsoever. Use this for production code.
- If the flag is 1, diagnostic data will be generated on the fly as determined by the other flags. When the API exits, the signal state of the system interface will be dumped in the specified diagnostic file.
- The PCM_DIAGMODE_PERFSTATS flag generates the performance data shown in our GUI panel.

```
// In API main or USER main
#define  PCM_DEVELOPMENT_DIAG      1
#define  PCM_DIAGMODE_RANGETEST    1
#define  PCM_DIAGMODE_PERFSTATS    1
#define  PCM_DIAGMODE_PERFFPSDATA  1


// In USER main
#define CORE_DUMP   "Files_Runtime//CoreDump.txt"
```

```
This creates a CoreDump.txt bucket in the Files_Runtime directory of
each demo. Among others, this core dump file features a snapshot of
the signal content of each channel of the system interface:

C0+1  =X_NULL+B6767C0,
C1+0  =
C2+4  =X_NORM+E5E0B8, X_NORM+E5E0B8, X_NORM+E5E0B8, X_NORM+E5E0B8,
C3+0  =
C4+0  =
C5+0  =

The format is

Channel# in order of definition + signals queued = signal ID
list+address of signal generating object.

Here, there are no problems, except maybe for a gradual accumulation
of unprocessed cursor normal state signals.
```

D101E – Diagnostic Levels(2)



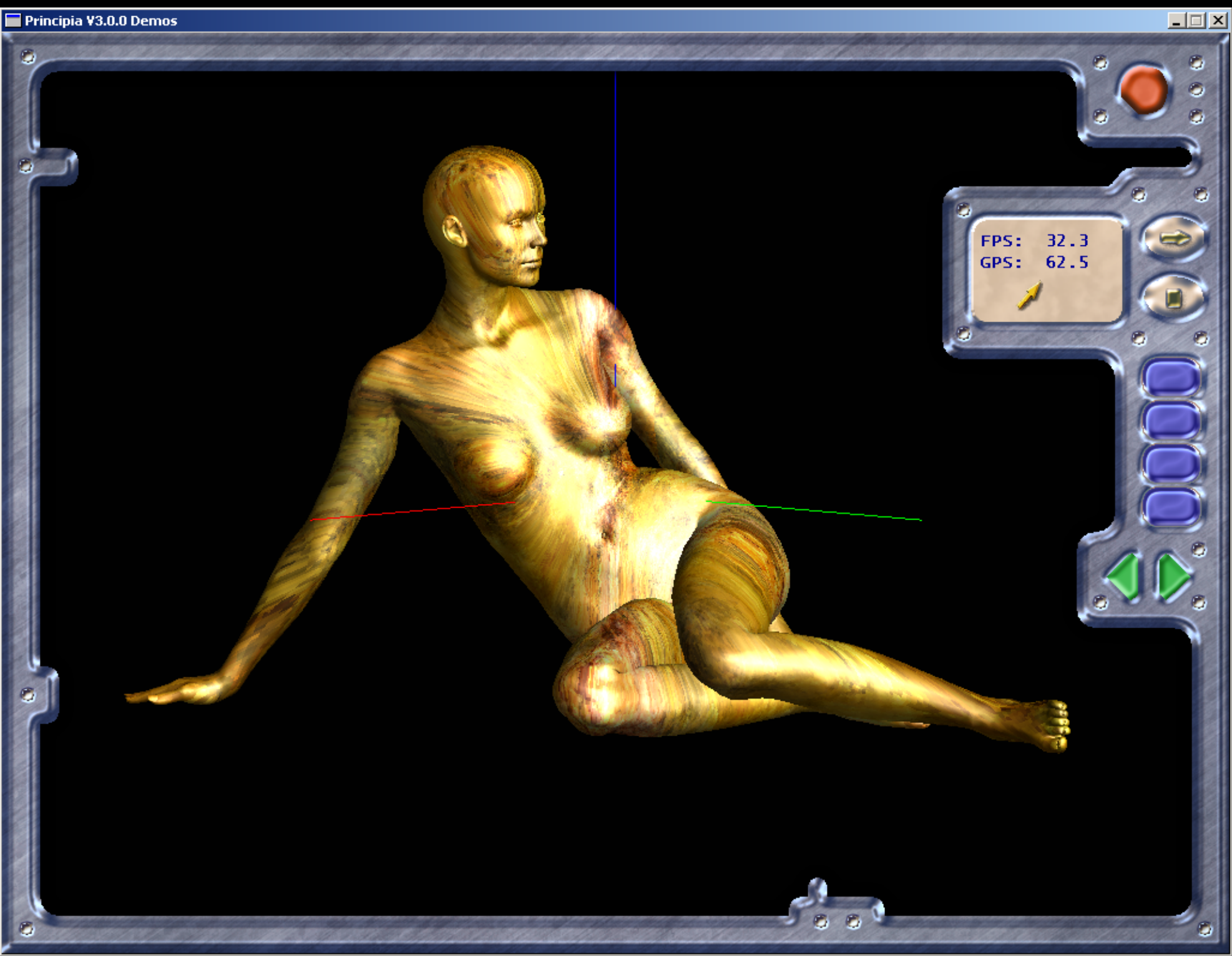
- The PCM_DIAGMODE_PERFFPS flag places in the core dump streaming frame performance data, such as the frame millisecond intervals shown above.
- One can examine the stream to see how is peformance really made up. In this example, we see the VSYNC forcing of the monitor. The peak occurs when the application window is put in the background. Overall, the application runs at 50fps without problems.

NOTE – Update Docs

- NB:
- Principia3 uses the PX_PerfMonitor() scriptable instrumentation performance analysis component, which replaces all of the above
- Update D101E accordingly

D101E – Summary

- Created an improved encapsulated modular GUI framework, that features all that is needed to build self-contained mini-application demos.
- Introduced more fundamental concepts
  - Surfaces and their types
  - State-changing controls
  - Nesting of controls within controls
- Provided basic insight into how the user can access some key pieces of the full Principia diagnostic and tuning toolbox without coding
  - Rich performance statistics anytime
  - Diagnostic levels and core dumps


Principia V3.0.0 Demos
FPS: 32.3
GPS: 62.5

D101F – Basic Rendering

- Ok – so far, we had an arguably cool and functional application interface, built with a minimum of development effort.
- This is still a far cry from the worlds filled with characters and objects of commercial games. Let's take a step in this direction:
  - Render an illuminated 3D object in empty space
  - Show the coordinate axis of the space
  - Show the basics of creating artificial environments and rendering game play elements with Principia.
- Seem simple – but buckle up, for this demo introduces as many new concepts as all the previous demos taken together!

D101F - Architecture Theory

- **The components of multimedia entertainment applications fall into three broad types**
  - **Framework components, that enable the use and realization of the application, but are not part of the entertainment experience per se.**
  - **Entertainment / media components, such as worlds, characters, objects that create the entertainment experience that absorbs the user.**
  - **Supporting components, that exist behind the scenes to implement the framework and entertainment components, and deliver the product.**
- **Principia provides a rich variety of components of each type for creating fully immersive multimedia entertainment applications. These components are organized hierarchically.**

D101F - Architecture Theory

- **Major categories of Principia components. For more complete information, please consult the Principia Reference Manual**
  - Functions
  - Base components
  - Helper components
  - Data encapsulators
  - Input, output and communication devices
  - Interfaces
  - Animatronics components
  - Graphic components
  - Audio components
  - Effects
  - Controls
  - Viewers
  - Object primitives
  - Operations
  - Worlds and world subcomponents
  - Procedures
  - System components
  - Productization components

D101F – New Concepts

- Demos 01A-01E introduced essentially framework components, such as 2D interfaces and controls.
- The object in D101F will be rendered as true 3D object that is part of a world. This will require the use of several new entertainment components
  - Cameras and lights
  - Materials and shaders
  - Vertex sets (geometry)
  - Meshes
  - Kinexes
  - Primitive objects
  - A world that contains the objects
  - A viewer to show what's in this world
- Although not very immersive, our simple sphere shows the basic concepts for building complex realistic multimedia environments with minimum effort.

D101F – New Concepts

- **Rendering framework**
  - A particular way to structure/sequence multimedia data and deliver it to the GPU/CPU for presentation.
  - Rendering frameworks can be nested (a high level structure can have many implemented variants).
- **Principia v3 topmost rendering frameworks**
  - ***Scheduled programmable pipeline framework.*** The application queues the presentation sequence for the GPU before rendering. The GPU then renders the sequence using programmable shaders. This is the standard for today's entertainment applications.
  - ***Scheduled fixed function pipeline framework***. Same as above, except that the GPU is managed as a state machine with rendering states set by material. This is a legacy rendering approach that still works well.
- **Most of the demos use shaders. You can read chapter 1-11 for using the legacy fixed pipeline.**

D101F – New Concepts

- At an abstract level, all current multimedia applications feature a natural hierarchy
  - Entertainment experience of games, movies…etc = controlled progression of scenes evolving in time
- Principia implements the hierarchy as follows
  - Scenes = made up of control components, principally books that implement user navigation
  - Books = made up of control components, mainly pages that implement context-sensitive content
  - Page = collection of any control components that constitute the content at a specific context and at a specific point in the user experience.
- What's on the page that presents our beautiful sculpture in its 3D world ?

D101F – New Concepts

World Viewer

- A fundamental control component that shows what is in the “world” of our entertainment experience.
- An application may feature many viewers and other related controls. There are many types of viewers. This demo features the simplest VIEW_3A viewer.

World

- A fundamental container of the entertainment components. There are many types of worlds built in Principia. This demo features the simplest type of world, WORLD_3A.
- Worlds are made of layers, operations and may have many other things, but we will leave these alone for now.

Objects

- One of the many things that a world can contain. Principia provides many different object types. Our sculpture is one of the simplest. Objects are made visible when the viewer renders the world.

D101F – New Concepts

Meshes:

- A mesh can be thought of as the smallest unit that can be rendered independently. It is a self contained rendering atom. Most objects are made of meshes.
- A mesh is usually realized as a sequence of geometries, materials, and few other optional items such as animation keyframes to be rendered together.

Geometry

- Principia offers many ways to represent geometry. Here, we use the simplest Vertex_Set3A - a set of 3D vertices that describe the surface geometry of a mesh or a part thereof

Materials

- Materials are essential and intricate components. They specify how the GPU will skin the 3D geometry and make it visible.
- Materials may contain shaders, GPU state and texture commands, base surface material lighting properties, effects, and much more.

D101F – New Concepts

Camera

- A component that defines how the user is looking at the world and objects therein, the lighting modality of the scene, where is the image rendered ...etc.
- Principia provides a rich set of camera components ranging from the simplest view port to complete cinematographic set-ups with multiple lights, motion tracking and much more.

Lights

- Components that provide illumination effects for the world and objects therein. This demo shows one of the simplest camera and lighting implementations.
- Strictly speaking, we do not need light objects when rendering with shaders, but they are a must when rendering with the fixed function pipeline.

D101F – New Concepts

Kinexes

- Kinexes are essential Principia components that hold location and state information for object, and may additionally affect GPU states when rendering.
- Internally, location is described using the 4x4 matrix convention commonly employed in 3D graphics. Principia handles both the DirectX and OpenGL matrix row-column conventions.
- Kinexes provide a variety of mechanisms for specifying and tracking object location and state.
- Every object has its own kinex. The ones used here are simple since our statue does not move (yet).

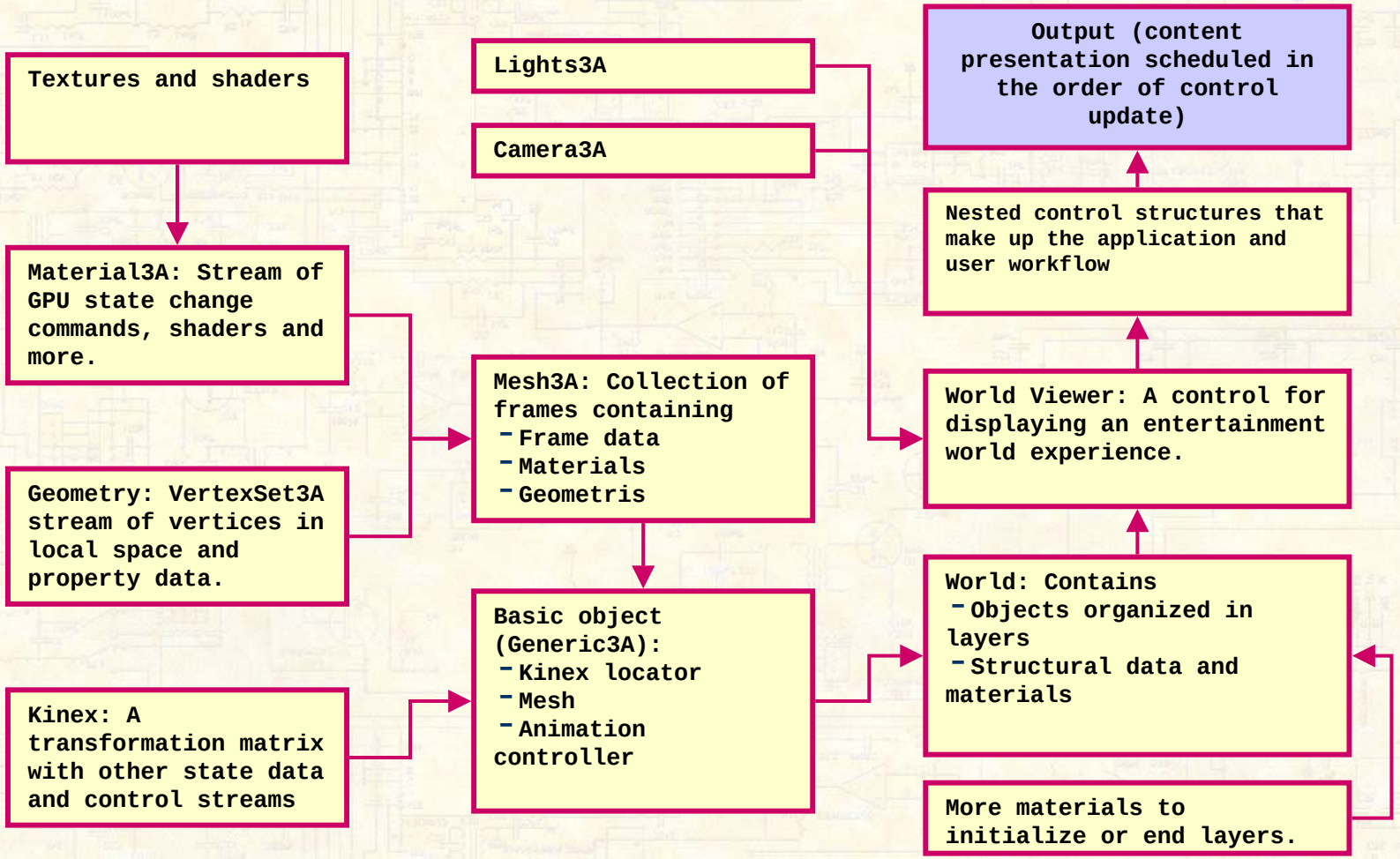
Animation Controllers

- Not used here. These are data structures that describe how the object kinex changes over time.

D101F – Implementation

- **Principia V3 is not a static game engine:**
  - There are many ways to organize and present the content within a given rendering framework.
- **The power of Principia:**
  - Rich variety of implementation methods and rendering frameworks.
  - Multitude of components, their properties and their methods – optimized for production.
  - Flexibility in putting it all together with a minimum mix of coding or standard scripting.
- **Demo_101F needs only scripts. There is no need for user code. The GUI is the same common include covered in previous demos.**

D101F – Implementation



D101F – World Viewer & Scene

- Simple scene, one book, single page
- The page includes the two standard GUI pages defined, and our most basic Viewer3A.
- The viewer refers to a world and to a camera used to illuminate and render/visualize it.

```
##define (VIEWER_MAIN) as <WORLDVIEWER_3A>
   #tag World            =  (           WRLD_MAIN )
   #tag Camera           =  (         CAMERA_MAIN )

##define (PG_MAIN) as <PAGE_2A>
   #tag Anchor    =  (             - ,      - )
   #tag Region    =  (             - ,      - ,       - ,      - )
   #tag Element   =  (   VIEWER_MAIN ,      0 ,       0 ,      - )
   #tag Element   =  (   PG_GUI_BASE ,      0 ,       0 ,      - )
   #tag Element   =  (   PG_GUI_OVER ,      0 ,       0 ,      - )

##define (BK_MAIN) as <BOOK_2A>
   #tag StartOpen =  (           1 )
   #tag StateID   =  (         MAIN ,     - ,     - ,       0 ,      0 ,                1 )
   #tag Element   =  (         MAIN ,     - ,     - ,       0 ,      0 ,    PG_MAIN ,   1 )

##define (SCENE_MAIN) as <SCENE_2A>
   #tag VarStop          =  (      VS_TERM )
   #tag Element          =  (     - ,    - ,      BK_MAIN )

##run (SCENE_MAIN)
```

D101F – World

- The simplest possible world class <WORLD_3A> contains nothing but isolated objects.
- All world classes feature a multi-layer system for managing the variety of objects they contain.
- Our world has the two objects shown – the sculpture and the axis system, in a single layer.

```
##define (WRLD_MAIN) as <WORLD_3A>
   #tag Layer_Decl  =  (     L_ITEMS ,   "ITEMS" , M_RESET , M_RESET , 16 )
   #tag Layer_Item  =  (     L_ITEMS ,              O_AXISYS )
   #tag Layer_Item  =  (     L_ITEMS ,              O_SCULPT )
```

- In addition, the world applies a GPU-resetting material before and after rendering the objects layer. This ensures controls render correctly.

D101F – Lights

- Our world has a single directional light with properties as defined in the script below.
- The <VsReg> tags place the relevant light properties in the desired GPU constant registers, so that these are accessible by the shader programs.
- This light serves both fixed pipeline rendering (for the axis) and shader rendering (for the sculpture)

```
##define (LITE_MAIN) as <LIGHT_3A>                    'Can be used for both fixed and shader renders
   #tag Type              =  (        LIGHT_DIRECT )  'Directional light
   #tag ColAmbi           =  (      55,   55,   55 )  'Ambient strength
   #tag ColDiff           =  (     190,  190,  190 )  'Diffuse strength
   #tag ColSpec           =  (     200,  200,  200 )  'Speclar strength
   #tag Position          =  (     0.0,  0.0,  0.0 )  'Light position
   #tag Direction         =  (    -0.3, -1.0, -0.5 )  'Light direction
   #tag Range             =  (                999.9 )  'Not applicable, range
   #tag Falloff           =  (                  1.0 )  'Not applicable, falloff
   #tag Theta             =  (                 25.0 )  'Not applicable, spot theta
   #tag Phi               =  (                 45.0 )  'Not applicable, spot phi
   #tag Coeffs            =  (  1.0, 0.0, 0.0, 0.0 )  'Not applicable, spot coeffs
   #tag VsReg_Dir         =  (                   30 )  'Shader register to place light incident dir
   #tag VsReg_Ambi        =  (                   31 )  'Shader register to place light color, ambient
   #tag VsReg_Diff        =  (                   32 )  'Shader register to place light color, diffuse
   #tag VsReg_Spec        =  (                   33 )  'Shader register to place light color, specular.
```

D101F – Camera

- There are many ways to specify even a simple <CAMERA_3A>. Here, its position, orientation and viewing parameters are simple constants from the script.
- Our camera set-up has a single light that will be activated when the camera itself is activated in the viewer.
- The camera sets the transpose composite view-projection matrix in vertex shader register #4, and its location in register #34. This data is used by the sculpture shader program.

```
##define (CAMERA_MAIN) as <CAMERA_3A>                'Can be used for both fixed and shader renders
   #tag Target           =  (               NONE )  'Render on the back buffer
   #tag Viewport         =  (  - ,  - ,   - ,   - )  'Use full extent of render target
   #tag View.Perspec     =  (                 NO )  'Use simple isometric render
   #tag View.EyePos      =  (   4.5,   5.0,  45.0 )  'Scalar camera DEA position
   #tag View.LookPos     =  (   0.0,   0.0,   0.0 )  'Scalar camera look-at position
   #tag View.Angle       =  (                35.0 )  'Scalar camera angle
   #tag View.UpVec       =  (   0.0,   0.0,   1.0 )  'Scalar camera up-vector
   #tag View.Aspect      =  (                 1.0 )  'Scalar camera aspect ratio
   #tag View.ZNear       =  (                 0.0 )  'Scalar camera depth limit
   #tag View.ZFar        =  (               255.0 )  'Scalar camera depth limit
   #tag Light            =  (   LITE_MAIN ,     1 )  'Main light is on
   #tag VsReg_VPT        =  (                   4 )  'Shader register to place (VxP)' matrix
   #tag VsReg_EPos       =  (                  34 )  'Shader register to place cam location

##define <SYS_VARIABLE>   =  (  CSHD_MSPEC ,  VARTYPE_FSR ,  8.00, 8.00, 8.00, 8.00 ) 'Specular power
```

D101F – Helper Components

- **Principia variables are used wherever data needs to be encapsulated. Here, a variable contains the RGBA material specular power, for use by the sculpture material.**

```
##define <SYS_VARIABLE>   =  (  CSHD_MSPEC ,  VARTYPE_FSR ,  8.00, 8.00, 8.00, 8.00 ) 'Specular power
```

- **Confused as to what all this means? Read chapters 1-11, 1-03 and 1-04 for the basics on illumination, materials, vertex streams, transforms, GPU states, shaders ...etc. You do not need to be familiar with their programming, only with the ideas behind the technology.**

D101F – Sculpture Object

- Our sculpture has a single mesh. It has no animation. The mesh has a single material and geometry. When presented, the GPU applies the material and renders the geometry.

```
##define (G_SCULPT) as <VERTEXSET_3A>
   #tag File      =  "Files_Media/Sculpture_F.3DS"
   #tag Definer   =  ( VTD_DCL_FILE )
   #tag Indexed   =  (          YES )
   #tag LoadNow   =  (            1 )

##define (R_SCULPT) as <MESH_3A>
   #tag Component =  (            0 ,    M_SCULPT ,  G_SCULPT )

##define (KX_SCULPT) as <KINEX_S>
   #tag Script    =  "Decl(M);RotZ(300.0);Mul(1.5,1.5,1.5);Mov(0.0,0.2,-0.5);Set(tM,0);"

##define (O_SCULPT) as <GENOBJECT_3A>
   #tag Construct =  (  KX_SCULPT  ,    R_SCULPT ,       NONE )
```

- Its location-state is determined by the KX_SCULPT kinex. It rotates, scales and translates the base geometry before rendering, and places the transpose of this transform matrix (model transform) into vertex shader register #0.
- The vertex set geometry is read from a 3DS file, created using 3DSMax (itself imported from Daz3D). The DCL_FILE definer tells Principia that we will render with shaders, and that the vertex format will be configured based on the file data.

D101F – Sculpture Material

- Materials look hairy – and they are. This demo uses three materials: one to reset the GPU, one to render the axis lines, and one to render the sculpture.
- Materials may contain a sequence of GPU state change commands in preparation for rendering an associated vertex stream, including texture specification, texture operation settings...etc.
- The material definitions available in Principia V3 reflect the standard DirectX9/OpenGL GPU state engines, which in turn mirror common hardware standards
- The material in addition tells which shaders to use, and what additional data to use to set shader constants.
- ***Remember: once a material state is set, it remains in force until changed by another material!***

```
##define (TX_COPPER) as <SURFACE>
   #tag Definer   =  (       DEF_TEX )
   #tag File      =
      "../Demo_Common/Files_Media/Textures/Tex_Metal_BurnishedCopper_51
      2.bmp"
   #tag ForceUID  =  "SFT"
   #tag LoadNow   =  (             1 )


##define (M_SCULPT) as <MATERIAL_3A>
   #tag RS      =  (               RS_LIGHTING ,              0 )
   #tag RS      =  (               RS_FILLMODE ,        RA_SOLID )
   #tag RS      =  (               RS_CULLMODE ,      RA_CULLCCW )
   #tag RS      =  (  RS_AMBIENTMATERIALSOURCE ,     RA_MATERIAL )
   #tag RS      =  (  RS_DIFFUSEMATERIALSOURCE ,       RA_COLOR1 )
   #tag RS      =  (                RS_ZENABLE ,              1 )
   #tag RS      =  (           RS_ZWRITEENABLE ,              1 )
   #tag RS      =  (        RS_ALPHATESTENABLE ,              0 )
   #tag RS      =  (       RS_ALPHABLENDENABLE ,              0 )
   #tag VShader =  (                 VS_BLINN )
   #tag PShader =  (               PS_GOURAUD )
   #tag TX      =  (             TX_COPPER , 0 )
   #tag VsVar   =  (          35 , CSHD_MSPEC )

@@@@ Turn off lighting, enable depth buffering, disable alpha blending,
@@@@ use the vertex and pixel shaders given, set TX_COPPER in texture
@@@@ slot zero and set the value of CSHD_MSPEC in shader register #35.


@@@@ Note the texture uses a DEF_TEX definer, not a DEF_S2D.
```

D101F – Shaders

- Shaders are small snippets of GPU machine code that describe how to render the presented data stream.
- Principia shaders can be written in assembly or in high-level languages such as HLSL (as here).
- Principia can automatically mange banks of shaders to adapt to different hardware levels.

```
##define (VS_BLINN) as <VSHADER_3A>
   #tag File      =  "VS_BlinnHalf.shd"
   #tag ShaderFcn =  "VS_Blinn"
   #tag ShaderAsm =  (           NO )
   #tag ShaderVs  =  (          1.1 )

##define (PS_GOURAUD) as <PSHADER_3A>
   #tag File      =  "PS_Gouraud.shd"
   #tag ShaderFcn =  "PS_Gouraud"
   #tag ShaderAsm =  (           NO )
   #tag ShaderVs  =  (          1.2 )
```

D101F – Vertex Shader

- The vertex shader implements standard per-vertex Blinn lighting.
- It transforms vertex positions and normals, calculates diffuse and specular terms, and transfers texture coordinates to the pixel shader.
- Chapter 1-04 covers shader design with many examples.

```
float4x4 mWrld     : register (c0);   // Api: world matrix
float4x4 mViewProj : register (c4);   // Api: VxP   matrix
float3   vLDir     : register (c30);  // Api: light dir
float4   vLAmbi    : register (c31);  // Api: ambient color
float4   vLDiff    : register (c32);  // Api: diffuse color
float4   vLSpec    : register (c33);  // Api: spcular color
float3   vEPos     : register (c34);  // Api: eye pos
float3   vMSpec    : register (c35);  // Api: material power

float4   wPos      ;                  // Wrk: Position vector, world space
float3   wNorm     ;                  // Wrk: Normal vector, world space
float3   wLite     ;                  // Wrk: Light incidence vector, world space
float3   wView     ;                  // Wrk: View vector, world space
float3   wHalf     ;                  // Wrk: Halfway vector, world space
float3   LdotN     ;                  // Wrk: Scalar product
float3   HdotN     ;                  // Wrk: Scalar product

struct VS_INPUT
{   // Inbound vertex buffer stream
    float3 vPosition        : POSITION ;
    float3 vNormal          : NORMAL   ;
    float2 vTexCoords       : TEXCOORD0; };

struct VS_OUTPUT
{   // Extant processed vertex stream
    float4 vPosition        : POSITION ;
    float4 vDiffuse         : COLOR0   ;
    float4 vSpecular        : COLOR1   ;
    float2 vTexCoord0       : TEXCOORD0; };

VS_OUTPUT VS_Blinn(const VS_INPUT In) { VS_OUTPUT Out     = (VS_OUTPUT) 0;

    /* Position world-view-projection transform and position output */
    wPos            = mul(float4(In.vPosition,1.0f), mWrld);
    Out.vPosition   = mul(wPos, mViewProj);

    /* Normals world transform */
    wNorm           = mul(In.vNormal, (float3x3) mWrld);
    wNorm           = normalize(wNorm);

    /* Common light, view and half vectors in world space */
    wLite           = normalize(-vLDir);    // Lite pos - Vertex pos
    wView           = vEPos - wPos.xyz;     // View pos - Vertex pos
    wView           = normalize(wView);
    wHalf           = 0.5*(wLite+wView);
    wHalf           = normalize(wHalf);

    /* Lambert diffuse and ambient per-vertex color output */
    LdotN           = saturate(dot(wNorm, wLite));
    Out.vDiffuse.rgb  = vLDiff* LdotN +vLAmbi;
    Out.vDiffuse.a    = float1(1.0);

    /* Blinn specular per-vertex color output */
    HdotN           = saturate(dot(wNorm, wHalf));
    Out.vSpecular.rgb = vLSpec*pow(HdotN, vMSpec);
    Out.vSpecular.a   = float1(1.0);

    /* Texture coordinates */
    Out.vTexCoord0.xy = In.vTexCoords.xy ;

/* Output stream to pixel shader */
return Out; }
```

D101F – Pixel Shader

- The pixel shader implements standard Gouraud lighting.
- It samples the texture, multiplies by the diffuse component and adds the specular components.
- Note that all constants in the shaders are pre-loaded by the material applied, and by the object kinex.
- Note that the material disables the fixed function lighting, and the geometry uses a non-FVF definer.

```
sampler  Tex0      ; // Api texture stage0


struct VS_OUTPUT
{   // Extant processed vertex stream
    float4 vPosition        : POSITION ;
    float4 vDiffuse         : COLOR0   ;
    float4 vSpecular        : COLOR1   ;
    float2 vTexCoord0       : TEXCOORD0; };

struct PS_OUTPUT
{   // Extant pixel shader stream
    float4 Color            : COLOR0   ; };


PS_OUTPUT PS_Gouraud(VS_OUTPUT In)
{   PS_OUTPUT Out = (PS_OUTPUT)0;

    /* Pixel color generation */
    Out.Color  = tex2D(Tex0, In.vTexCoord0);
    Out.Color *= In.vDiffuse;
    Out.Color += In.vSpecular;

    /* Shader output */
return Out;}
```

D101F – Axis Object

- The axis system is rendered using the fixed function pipeline. This is done by turning off the shaders, and using an FVF geometry.
- We can use only the vertex or pixel portion of the fixed pipeline by turning off only the vertex or pixel shader.
- Our axis object does not need lighting, and its GPU states are correspondingly simple. We use wireframe mode to render only the lines.
- Note that the kinex is defined inline in the object. There is no need to worry about shader constants here.

```
##define (M_CRLINE) as <MATERIAL_3A>
   #tag RS      =  (               RS_LIGHTING ,              0 )
   #tag RS      =  (               RS_CULLMODE ,     RA_CULLNONE )
   #tag RS      =  (               RS_FILLMODE ,    RA_WIREFRAME )
   #tag RS      =  (  RS_AMBIENTMATERIALSOURCE ,       RA_COLOR1 )
   #tag RS      =  (  RS_DIFFUSEMATERIALSOURCE ,       RA_COLOR1 )
   #tag TS      =  (                TS_COLOROP ,      TO_DISABLE ,   0 )
   #tag TS      =  (                TS_ALPHAOP ,      TO_DISABLE ,   0 )
   #tag TX      =  (                     - , 0 )
   #tag VShader =  (                     NONE )
   #tag PShader =  (                     NONE )

##define (G_AXISYS) as <VERTEXSET_3A>
   #tag File      =
      "../Demo_Common/Files_Media/Calibration\Mesh_UnitAxisFrame.vsf"
   #tag Definer   =  ( VTD_FVF_FILE )
   #tag LoadNow   =  (           1 )

##define (R_AXISYS) as <MESH_3A>
   #tag Instance  =  (     INST_REF )
   #tag Component =  (           0 ,    M_CRLINE ,  G_AXISYS )

##define (O_AXISYS) as <GENOBJECT_3A>
   #tag Instance  =  (     INST_REF )
   #tag Construct =  (   "Decl(M);" ,    R_AXISYS ,       NONE )
```

D101F – Axis Geometry

- There are four ways for specifying 3D vertex geometry in Principia:
  - Read from standard 3D file formats such as 3DS.
  - Read from Principia formats such as VSF and VSD.
  - Procedural generation
  - On the fly calculation
- The geometry of the axis is so simple that we can create a VSF file by hand
- VSF files can be text or binary. They can have significant complexity, or be as simple as here.

```
@@@ The content of the "Files_Media\Calibration\Mesh_UnitAxisFrame.vsf“
@@@ forms a mesh of three flattened triangles with vertices of different
@@@ color to represent each coordinate axis (x=red, y=grn, z=blu)


TEXT
VSF3
UNIT_SIZE_XYZ_AXIS_FRAME
@@ Vertex format specification
POS:3F, COLOR:COLOR
@@ Primitive type
4
@@ Indexed flag
0
@@ Reserved bootstrap section
0
@@ Total number of vertices and indices
9
0
@@ Number of frames
1
@@ Frame#0: Primitive first vertex index, primitive count, nb of vertices
0  3  9
@@ Frame#0: Locus of first frame in index array and index count per frame
0  0
@@ Vertex data
0.000   0.000   0.000    FFFF0000
0.500   0.000   0.000    FFFF0000
1.000   0.000   0.000    FFFF0000
0.000   0.000   0.000    FF00FF00
0.000   0.500   0.000    FF00FF00
0.000   1.000   0.000    FF00FF00
0.000   0.000   0.000    FF0000FF
0.000   0.000   0.500    FF0000FF
0.000   0.000   1.000    FF0000FF
```

D101F – Interfaces and Color

- **Interfaces will be covered later. For now, if we examine the graphic interface defined in the common interfaces script, we will notice three important tags for 3D rendering:**
  - **Fill the display buffer with solid color at the beginning of every frame. Since we have no background image anymore, not doing this will lead to artifacts from previous frames lingering on.**
  - **Clear the depth buffer at every frame too by filling it with ones. Since we are rendering 3D scenes with depth testing, this ensures that frame elements are overlaid properly without interference from the previous frame.**
- **Many Principia tags require the specification of color. There are four ways for doing so:**
  - **As a 32bit ARGB hex number**
  - **As an R,G,B byte triplet. Such color is always 100% opaque**
  - **As an R,G,B,A byte quad with A=0x00 being 100% transparent**
  - **As a reference to a color object**
- **Externally and internally, Principia typically manages color as a standard 32-bit ARGB word. Remember this!**

```
##create (VISUAL_INTERFACE) as <VISUAL_INTERFACE>

   . . .
   #tag Clear_Color       =  (   0xFF010101 ,  YES )
   #tag Clear_Depth       =  (          1.0 ,  YES )
   #tag Clear_Stencil     =  (            0 ,   NO )
   . . .

@@@ Alternatively we could have used a reference to the
@@@ standard solid black from the standard include library.
@@@ This color is defined as 0xFF010101 and not 0x00000000,
@@@ to avoid confusion with transparency

##define <COLOR_RGB>      =  ( CL_BLACK  ,   1,  1,  1)
```

D101F – Building Blocks Recap

- **GX_Camera3A::AX_Graphic**
  - Core Principia object used to describe the 3D viewing and projection transformations in layman parameters
- **GX_Light3A::AX_Graphic**
  - Core Principia light source object used to illuminate the scene as seen through the camera
- **GX_Material3A::AX_Graphic**
  - Collection of GPU state commands and base material properties (some of which may be sourced for lighting calculations depending on RS_*SOURCE)
- **GX_VSet3A::AX_Graphic**
  - Vertex stream primitive object. This is basically the discretized 3D geometry of the sculpture. More on this in Chapter 3.
- **GX_Mesh3A::AX_Graphic**
  - Collection of primitives and materials rendered together (here, the assembly has one material and one geometry). The assembly of complex objects will be covered in many subsequent chapters.
- **BX_Generic3A::AX_Object, WX_World3A::AX_World**
  - The sculpture as an object belonging to a world we can show. More in the “Objects” and “Worlds” chapters
- **CX_View3A::CX_Control**
  - Basic 3D viewer control. More in the “Viewers” chapter

D101F – Summary

- Rendered simple 3D objects within the GUI previously created, using both shaders and the fixed function pipeline.
- Introduced a basic framework and many fundamental concepts for creating and rendering the environments - or worlds – of our entertainment applications
- This concludes Chapter 1 of the demo collection. We can now start exploring the components and capabilities offered by Principia, starting with controls, and eventually progressing to complete commercial-grade applications.

Chapter 1 - Lessons

- How to write, compile and run a basic Principia application
- How to display an image on screen
- How to assemble basic scenes, books and pages
- How to create variables for holding information
- How to interleave custom user code with Principia scripts
- How to define and execute/display components
- The basics of interfaces, channels and signals
- Nesting controls to create display contents
- How to create and interact with a basic cursor
- How to implement basic transparency
- How to use basic frames to create a GUI
- How to use basic buttons
- How to incorporate simple effects on the buttons
- How to use a rotator to scroll thru display elements

Chapter 1 - Lessons

- How to create glyph fonts and print text
- How to access scripted components and Principia internal variables from within user code
- How to obtain core diagnostic and performance data
- How to encapsulate functionality within standalone code and script packets for use in other applications
- How to use the standard include library
- How to add audio capabilities and play basic sounds
- How to use definers to select the right type and memory location for media objects e.g. images and vertex sets
- How to work with basic color and alpha transparency
- How to cast shadows and glows around images

Chapter 1 - Lessons

- How to use shaders and/or the fixed-pipeline rendering frameworks to render 3D objects
- How to light and texture 3D objects using material
- How to input geometry from one of the many 3D file formats supported
- How to send basic render state commands to the GPU prior to rendering a vertex stream
- How to blend ambient and diffuse lighting properties to create shadows and highlights in a simple lighting model
- How to render lines
- How to use kinexes to place objects at different points, orientation and states in the 3D world
- How to create a basic 3D world and a viewer thereof integrated in the display control stream

Chapter 1 – The End

- In summary, we covered quite a bunch of basic capabilities that come handy for any application !
- We also start to see one incipient architectural framework for building Principia applications:
  - In script, define components, workflow chunks and scenes that make up the application, plus all the basic supporting infrastructure such as interfaces.
  - In user code, obtain references to the defined components that the application will manipulate.
  - In user code, bind user procedures to components where the application will manipulate contents.
  - Write the user procedures to implement the custom application behavior of these components.
- With this, we are ready to go forth and explore all that Principia has to offer !