GEM TN-93-422

Local GEM Computing Meeting

June 8, 1993

Abstract;

Attendees, agenda and presentations of the Local GEM Computing Meeting held on June 8, 1993

LOCAL GEM COMPUTING MEETING 8/JUN/93

JOHN WOMERSLEY	MS 2005	womersley@ssc.gov
Irwin Sheer	MS 2000	Irwin_Sheer@ssc.gov
Gary Word	2005	Gary_Word@ssc.gov
Victor Perevozchikov	MS 2003	PEREV@SSC.GOV
Yuri Fisyak	MS 2005	FISYAK@SSCVX1
HENK Uijterwaal	2005	HENK@PDSF
Sasha Vanyashin	2005	vanyashin@sscvx1
Ken McFarlane	2005	mcfarlan@sscvx1

From: SSCVX1::ADAMS 14-MAY-1993 17:07:52.44
To: ADAMS
CC:
Subj: Agenda for Rice SDC Computing Workshop 5/17-19

===== Message 67 in folder SDC.COMPUTING extracted from VaxNews 3.8

===== Description : Agenda for Rice SDC Computing Workshop 5/17-19
===== From : FNALV::GAINES
===== Path : sscvx1!fnalv!vaxnews!GAINES
===== Posted on : 12-MAY-1993 16:46:29
===== Expires on : 27-MAY-1993 16:46:29
===== Message ID : <00000016_VAXNEWS@fnalv>

SDC Computing Working Group Meeting
May 17 - 19, 1993
Rice University

AGENDA
(Please send any additions to or comments on the agenda to Irwin Gaines at GAINES@FNAL.GOV)
Note that many of these topics will involve discussion by the group rather than formal presentations.

Monday May 17: Tools
9:00 Storage Media: Commercial trends, available products and forecasts
Fayne Sisco (IBM Houston)
Discussion of Unitree
10:30 Graphics standards Nina Vogl (IBM Austin)
SDC Specific graphics packages
12:30-2:00 Lunch
2:00 Programming languages, object oriented paradigms, etc
C++/Fortran Interlanguage Tutorial Glen Kubena/Alicia Harvey
Plans for Inter Language Evaluation Task
SDC/GEM commonality: C++ class libraries?

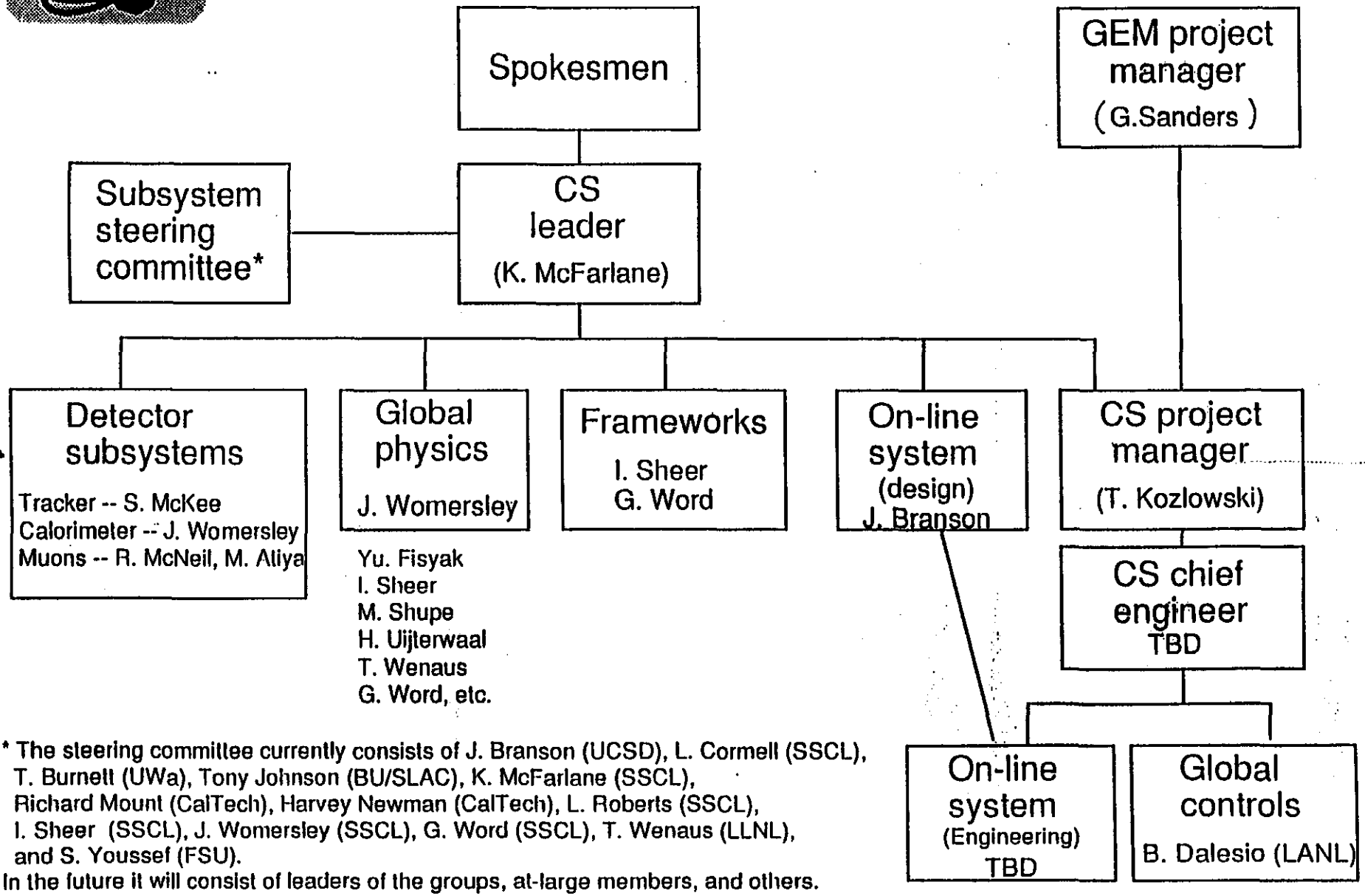
Tuesday May 18: Physics Application Code
9:00 Simulation Task Force Report Shuichi
Plans for SDCSIM upgrades Shuichi
10:00 Package Organization and Management Shuichi
10:30 Physics Coding Standards: Coordinate system, units, naming conventions, etc
11:00 Further discussion of potential SDC/GEM common projects
12:00-1:30 Lunch
1:30 Event structure, now and future
2:00 Data bases
3:00 Code management (depatchyfying SDCSIM) Dave S
4:00 Centrally provided SW services (certification, documentation, maintenance, etc) Soren

Wednesday May 19: SDC Software Environment
9:00-10:00 SDC Workgroup Cluster Alan
Utopia/WASH/batch systems Alan
Discussion of proposed batch systems for PDSF Soren
SDC Remote Computing Centers (postponed to KEK workshop)
10:00-10:30 Application Servers Mark
11:00-12:00 Standard SDC (SSC?) UNIX environment Discussion, provoked by Irwin
An SDC tree? ditto
12:00-1:30 Lunch
1:30 Data modelling task force
2:00 An Object-Oriented Programming Framework Forest Rouse
3:00 CORE SW Soren
demo Mark
prototype
plans/schedule Soren

LOGISTICS:

Joint June 30 July 2

Computing Subsystem Organization



* The steering committee currently consists of J. Branson (UCSD), L. Cormell (SSCL), T. Burnett (UWa), Tony Johnson (BU/SLAC), K. McFarlane (SSCL), Richard Mount (CalTech), Harvey Newman (CalTech), L. Roberts (SSCL), I. Sheer (SSCL), J. Womersley (SSCL), G. Word (SSCL), T. Wenaus (LLNL), and S. Youssef (FSU).
In the future it will consist of leaders of the groups, at-large members, and others.

( ) = Acting

KM 5/28/93

Overview Of 3D Graphics Libraries

Nina Vogl

International Technical Support Center

Austin, Texas

1992

Trademarks and Acknowledgments

™ AIX is a registered trademark of International Business Machines Corp.

™ AIX-GL is a trademark of International Business Machines

™ AIXwindows is a trademark of International Business Machines Corp.

® graPHIGS is a registered trademark of International Business Machines Corp.

® IBM is a registered trademark of International Business Machines Corp.

™ IRIS is a trademark of Silicon Graphics, Inc.

® Micro Channel is a registered trademark of International Business Machines Corporation

® OSF/Motif is a registered trademark of Open Software Foundation, Inc.

® UNIX is a registered trademark of Unix System Laboratories, Inc.

® X Window System is a registered trademark of Massachusetts Institute of Technology

Related Publications & Acknowledgments

Using the graPHIGS Programming Interface
Presentation Guide
W.G.Tuel, C.Warrick, B.Straka
G320-3550

GL for the RISC System/6000
Programming Guide
GG24-3862

3D Computer Graphics Concepts
An Overview
GG24-3836

Agenda

- *Introduction*
  - 3D Graphics APIs available under AIX
- *IBM GL*
  - What It Is - Implementation Specifics
  - What It Is - From The Technical Perspective
  - GL Functions & Specifics
  - Characterization - How GL Differs From Other 3D Graphics APIs
  - Large Base Of Existing Applications In Several Areas
  - Drawbacks?!
- *PHIGS Based 3D Programming Interfaces*
  - The PHIGS Standard
  - IBM graPHIGS
  - PEX
- *GL vs. PHIGS - Which 3D Graphics Library*

3D Graphics APIs Under AIX

- *Available 3D Graphics APIs are:*
  - Graphics Library (GL)
  - graPHIGS
  - PEX

- *Packaging*
  - AIXwindows Environment/6000 1.3:contains X11 & Motif
  - AIXwindows Environment/6000 3D:contains all 3D graphics APIs, server extensions...

IBM GL

- *What It Is - Implementation Specifics*
- *What It Is - From The Technical Perspective*
- *GL Functions & Specifics*
- *Characterization - How GL Differs From Other 3D Graphics APIs*
- *Large Base Of Existing Applications In Several Areas*
- *Drawbacks?!*

GL Implementation Specifics

- *IBM's Port Of SGI's GL 3.3*
  - GL, a 3D graphics programming interface
  - Licensed in 1988
- *DeFacto Standard*
  - Large existing base of applications is available (over 1500 applications)
  - SGI's Leadership + IBM's quality and floating point performance
- *Part Of AIXwindows 3D*
- *Available On SGI And IBM (only)*
- *Outlook*
  - OpenGL will make GL generally available

GL From A Technica Perspective

- *AIXwindows Client*
  - Based ontop of X-server
  - Uses Motif window manager
  - Is supported as GL widget
- *Supported Programming Interfaces*
  - C
  - Fortran
- *Subroutine Names Are Self-explaining*
- *No Clearly Stated Programming Design Principc*
  - Makes it very flexible
- *Immediate Mode Library*
  - Hierarchical programming possible with "GL-objects"

GL Functions & Specifics

- *Graphics Primitives*
  - 2D/3D user defined coordinates
  - Points and vectors
  - Polygons
  - Curves
  - NURBS
  - Characters

- *Graphics Processing Pipeline - Makes Use Of Hardware Support*
  - Matrix transformations
  - Projections
  - Clipping
  - Scaling
  - Rotating

GL Functions & Specifics...

- *Display Process*
  - Shading
  - Fill interior (flat shading)
  - Depth Cueing
  - Hidden Line Hidden Surface Removal
  - Double buffering
  - Bitplane control (overlay/underlay buffer)
  - Bit-mapped character support
  - Solid rendering
  - Up to eight light sources
- *User Interaction*
  - Pop-up menus in overlay planes
  - Support for various input devices
  - Support for 3 colour cursor

GL Characterization

- *"C-Oriented"*
- *Like A Low-Level Programming Language*
  - Framebuffer control at bitplane level
  - User defined
  - Flexible
  - Powerful
- *For Successful Programming It Is Necessary To Understand*
- *Easy To Use - Simple Programming For Simple Tasks - Fast Learning Curve*
- *"Everything Is Possible" - But The Programmer Has To Do It, e.g.:*
  - Doublebuffering and swapbuffers()
- *"Everything Is Possible" - But The Hardware Support Must Be Given*

GL Application Areas

- *CAD*
- *Structural Engineering*
- *Oil Exploration*
- *Meteorology*
- *Military Sector*
- *Scientific Visualization*
- *Fluid Dynamics*
- *Medical Imaging*
- *Molecular Modeling*
- *Animations*

GL: Drawbacks?!

- *No Common Subroutine Prefix*
  - Advantage of easy to remember subroutine names
- *No Clearly Stated Design Principal*
  - Requires programmers discipline for complex tasks
- *No Networking*
  - Might change with OpenGL
- *Hardware Dependent - Needs 3D Card*
- *Local Lighting Model Only - No Raytracing*
  - This is true for most 3D graphics APIs
  - Additional software can solve this problem
- *RGB Model Only - No CMY Or Other Colour Model*
  - Conversion algorithms are available

PHIGS Based Graphics APIs

- *The PHIGS Standard*
- *IBM graPHIGS*
- *PEX*

The PHIGS Standard

- *What PHIGS Stands For*
- *Standardization*
- *Objectives*
- *The Scope Of PHIGS and PHIGS-PLUS*

What PHIGS Stands For

- *PHIGS*
  - **P**rogrammers
  - **H**ierarchical
  - **I**nteractive
  - **G**raphics
  - **S**ystem
- *PHIGS-PLUS*
  - **P**lus (PHIGS plus...)
  - **LU**miere
  - **S**urfaces

PHIGS ANSI/ISO Standardization

- *ISO/IEC 9592-1* *PHIGS, Part 1, Functional Description*
- *ISO/IEC 9593-1* *FORTRAN Language Bindings Of PHIGS*
- *ISO/IEC 9593-4* *C Language bindings Of PHIGS*
- *X3H31-89-05* *Proposed PHIGS-PLUS Standard (07-25-89)* *(Functionality AND Language Bindings)*

PHIGS Objectives

- *3D Modelling And Viewing*
- *Graphics Hierarchies*
- *Interactive Device Input*
- *Graphics Object Identification And Editing*
- *Subroutine Bindings*
- *Separate Workstation And Application Resources*
- *Device Independence*
- *Programmer Independence*

The Scope Of PHIGS

- *PHIGS Defines - 3D Wireframe Modeling*
  - Primitives
    - *Lines*
    - *Text*
    - *Polygons*
  - Colour tables
  - Views and workstations
  - Transformation pipeline
  - Device and state inquiry
  - Resource definition
  - Finite state definition

The Scope Of PHIGS-PLUS

- *PHIGS-PLUS Adds Functions To Represent Surfaces And Lighting*
  - Lighting
  - Surface representation including NURBS
  - Shading
  - Direct colour
  - Depth cueing
  - Hidden Line Hidden Surface Removal

IBM's graPHIGS

- *What It Is*
  - IBM's PHIGS Implementation
  - Includes PHIGS+
  - ISO/PHIGS and graPHIGS bindings differences
- *Technical Specifications*
- *Design Principles And Characterization*
- *Implementation Specifics*
  - AIXwindows Client
  - Motif window manager
  - graPHIGS widget available
- *Drawbacks?!*

graPHIGS - What It Is

- ***IBM's PHIGS***
  - **Based On ANSI/ISO Standard for PHIGS FUNCTIONAL specifications**
  - **Available as**
    - *GDDM graPHIGS*
    - *Personal graPHIGS (AIX PS/2, RISC System/6000)*
    - *graPHIGS as part of AIXwindows_3D (RISC Systems/6000 and AIX 3.2.3)*
- ***Includes PHIGS+ Specifications***

graPHIGS - What It Is...

- ***Differences Between ISO/PHIGS Bindings And graPHIGS Bindings***
  - **Subroutine names differ: Prefix "GP" versus prefix "P"**
  - **Numerical values associated with enumerated items might differ by one**
  - **Storage order of transformation matrix differs (transpose matrix)**
  - **Parameter packaging for subroutine calls might differ**
  - **Coordinate lists:**
    - *PHIGS: separate arrays for x, y and z coordinates*
    - *graPHIGS: one single matrix*
- ***BUT:***
  - **PHIGS bindings are understood and accepted by compiler**
  - **grapHIGS and ISO/PHIGS calls may be interchanged**
    - *PHIGS program with PHIGS+ enhancements*

graPHIGS - Technical Specifications

- *2D/3D Graphics Library*
- *Retained Mode Library*
- *Explicit Traversal Control Available*
- *Callable From Various Programming Languages*
  - FORTRAN
  - C
  - PL/1
  - Pascal
  - Assembler
- *Language Bindings*
  - Same names for all different languages
  - All subroutine calls start with "GP"
  - Mnemonic abbreviations
    - *e.g.: GPOPWS0*

graPHIGS - Technical Specifications...

- *Device Independent Programming Possible*
  - Runs On All RISC System/6000 Graphics Hardware - at least via Xlib
  - Hardware inquiry functions available
- *CGM, GDF Output*
- *Own Terminology - Sometimes Different From Other Graphics APIs*
- *Supports Different Colour Models*
- *Networking Capabilities*
  - Using the X-client-server protocol
  - Using the Nucleus-Shell protocol
- *Contains GKS-Compatibility Option*

graPHIGS Design Principles

- *Clearly Stated Hierarchical Design Principal*
  - Well "organized"
  - Structure Stores, structures and structure traversal
  - Workstation type definitions / Workstations description tables
- *Like "High Level Programming Language":*
  - Complex
    - *Long learning curve*
    - *Overhead in case of easy tasks*
  - "Automatic" rather than "user defined"
  - FORTRAN history
- *Functional Specifics:*
  - Primitive Calls Plus Attribute Calls
  - Polygons with subareas definable
  - Hardware inquiry functions available

graPHIGS AIX Implementation Specifics

- *AIXwindows Client*
  - Xlib versus XDWA
  - 2D through Xlib possible
  - Networking:
    - *X-client-server protocol*
    - *Nucleus-Shell protocol*
- *Motif Window Manager*
- *gP Widget Available*

graPHIGS Drawbacks?!

- *Naming Conventions - Difficult To Remember?!*
- *Forces Programmer Into "Structured Design"*
  - Can Cause Overhead - program source as well as executable
  - Inflexible
  - FORTRAN history recognizable, even in C
- *"Automatic" Rather Than "User defined"*
  - Couldn't do it any different, even if you wanted to
  - You don't necessarily know what's going on "behind the scenes"
- *Has Its Own Terminology - Differs From "Common" Graphics Terminology*
- *Long Learning Curve Required*
- *Networking is either:*
  - Not Supporting 3D - or -
  - Not standard compliant
- *Standardization*
  - Functionality: mix of industry standard and proposed standard
  - Bindings: Not standard compliant, but standard bindings are usable

PEX

- *What Is PEX*
- *graPHIGS Versus PEX*
- *PEX And AIX*
  - So Far
  - With X11.5
- *PEX Drawbacks?!*

PEX - What It Is

- *PEX = PHIGS Extension to X*

- *Consists Of Two Parts:*
  - Protocol Definitions — PEX-Server
  - PHIGS API Implementation — PEX-Client

PEX Protocol & Design Requirements

- *Overcomes X-windows Shortcomings Respectively*
- *Implements PHIGS Standard In X-Windows Environment*
- *Standardized Protocol Between PEX-Server And PEX-Client*
  - Guaranteed interoperability between different hardware vendors
  - Non-Proprietary Protocol / Freeware software
  - Compatible with X-protocol
  - Could be the base for any 3D graphics API

The PEX AI I

- *So Far: MIT's Initial Release 5.0*
  - Not supported functions in release 5.0:
    - *PHIGS+ functionalities*
      - *Lighting*
      - *Shading*
      - *Solid modelling*
    - *Input device handling*
    - *Double buffering*
    - *Z-buffering*

- *Now, Since X11.5: PEX 5.1*
  - Includes now also PHIGS+ functionalities
  - MIT calles it PEX-SI
  - IBM calls it PEXlib

- *Documentation Available On The X11.5 Distribution Tape*

graPHIGS versus PEX

- ***AIX graPHIGS Fully Supports all PEX Design Requirements, except:***
  - **The Networking Protocol is either:**
    - *2D only* *or*
    - *3D, but proprietary and IBM only*

- ***The graPHIGS API is Further Developed***

PEX And AIX

- *Up To Now:*
  - PEX is part of AIXwindows Environment/6000-3D
  - PEX-Server as extension to the X-server
  - API: MIT's Initial Release
- *With X11.5:*
  - PEX protocol as X server extension: "X -x pex"
  - Two APIs are available that support the PEX protocoll:
    - *PEXlib*
    - *graPHIGS*

PEX: Drawbacks?!

- *Most Vendors Have Already Established PHIGS-based APIs Implemented*
- *API Is Still Not Completely Finished*
- *PEX Is A PHIGS Based API*

Which 3D Graphics Library

- *The Decision Should Be Based On:*
  - Given Hardware
  - Programmers skill and background
  - Design and complexity of the problem
  - Standard requirements
  - Networking required?!
  - Learning Curve and available development time
  - Both programming interfaces have PROs and CONs
  - Compare the "GL-Drawbacks" and "graPHIGS/PEX Drawbacks"

- *Outlook - Both Designs Have Future!*
  - OpenGL
  - PEX

Appendix A

"Hello World" Program With GL

- "Hello World"

- glhello.c

```
#include <gl/gl.h>
main()
{
 prefsize (200, 100);
 winopen ("Hi There");
 color (BLACK);
 clear();
 color(RED);
 cmov2 (50, 50);
 charstr ("Hello, World!");
 sleep (5);
}
```

Appendix B

"Hello World" Program With graPHIGS

- "Hello World"

- **graPHIGS_hello.c**

```
#include <afmnc.h>
#include <stdio.h>
main()
{
#define WSID 1
#define START 0
#define NUM 3
#define WHITE 1
#define STRUC 1
#define VIEW0 0
```

```
#define PRFORM 2
#define DEVID 3
#define BREAK 1
#define CSID 6
#define FONT 1
#define STROKE 3
#define ERFILE "SYSPRINT"
 #define WSTYPE "X "
#define CONNID "* " *
#define CONLEN 8
#define CONIDL 8
 #define NCID 1
#define CONN 1
#define SLEN 0
#define SPEC " "
 #define SSID 5
#define SSTYPE 1
#define SSDESC 0
 #define LMSG 44
#define LTMSG 24
#define LTEXT1 11
#define HGHT1 0.05
#define PRIOR 0.0

 int STATUS;
int CHOICE;
char MSG[45];
char TEXT1[11];
float POS1[2];
float COLORS[9];
char CONID[8];
char WST[8];
int RCONIDL;
int ERRIND;
int TOTNUM;
int CTIDAR[10];
int MAJ, CLAS, MIN;
struct {
 int ladlb ;
 int lads ;
int ERRIND;
```

```
int TOTNUM;
int CTIDAR[10];
int MAJ, CLAS, MIN;
struct {
 int ladtb ;
 int lads ;
 int code ;
 int comm ;
 int ltarg ;
 char target[8] ;
 } ADIB ;
strcpy(MSG," PRESS MOUSE BUTTON TO EXIT ");
strcpy(TEXT1,"Hello World");
 POS1[0] = 0.0;
 POS1[1] = 0.0;
COLORS[0] = 0.0;
 COLORS[1] = 0.0;
 COLORS[2] = 0.0;
 COLORS[3] = 1.0;
 COLORS[4] = 1.0;
 COLORS[5] = 1.0;
 COLORS[6] = 1.0;
 COLORS[7] = 1.0;
 COLORS[8] = 0.0;
ADIB.ladtb = 24;
 ADIB.lads = 20;
 ADIB.code = 104;
 ADIB.comm = 0;
 ADIB.ltarg = 0;

 GPOPPH( ERFILE, &ADIB);
GPCNC (NCID, CONN, SLEN, SPEC) ;
GPCRWS (WSID,NCID,CONLEN,CONNID,WSTYPE,0);
 int code ;
 int comm ;
 int ltarg ;
 char target[8] ;
 } ADIB ;
strcpy(MSG," PRESS MOUSE BUTTON TO EXIT ");
strcpy(TEXT1,"Hello World");
```

```
POS1[0] = 0.0;
POS1[1] = 0.0;
COLORS[0] = 0.0;
COLORS[1] = 0.0;
COLORS[2] = 0.0;
COLORS[3] = 1.0;
COLORS[4] = 1.0;
COLORS[5] = 1.0;
COLORS[6] = 1.0;
COLORS[7] = 1.0;
COLORS[8] = 0.0;
ADIB.ladlb = 24;
ADIB.lads = 20;
ADIB.code = 104;
ADIB.comm = 0;
ADIB.ltarg = 0;

GPOPPH( ERFILE, &ADIB);
GPCNC (NCID, CONN, SLEN, SPEC) ;
GPCRWS (WSID,NCID,CONLEN,CONNID,WSTYPE,0);
GPQRCT (WSID,CONIDL,&ERRIND,&RCONIDL,CONID,WST);
if(ERRIND != 0)
{
puts("Workstation not opened -- Exiting program");
return;
};

GPCRSS (SSID,NCID,SSTYPE,SSDESC);
GPASSW (WSID, SSID) ;
GPSSS (SSID) ;
GPQCID (WSID,1,10,&ERRIND,&TOTNUM,CTIDAR);
GPXCR (WSID,CTIDAR[0],START,NUM,COLORS);
GPACFO (WSID,CSID,FONT);
GPOPST (STRUC);
POS1[0] = 0.0;
POS1[1] = 0.0;
COLORS[0] = 0.0;
COLORS[1] = 0.0;
COLORS[2] = 0.0;
COLORS[3] = 1.0;
COLORS[4] = 1.0;
```

```
COLORS[5] = 1.0;
COLORS[6] = 1.0;
COLORS[7] = 1.0;
COLORS[8] = 0.0;
ADIB.ladlb = 24;
ADIB.lads = 20;
ADIB.code = 104;
ADIB.comn = 0;
ADIB.ltarg = 0;

GPOPPH( ERFILE, &ADIB);
GPCNC (NCID, CONN, SLEN, SPEC) ;
GPCRWS (WSID,NCID,CONLEN,CONNID,WSTYPE,0);
GPQRCT (WSID,CONIDL,&ERRIND,&RCONIDL,CONID,WST);
if(ERRIND != 0)
{
puts("Workstation not opened -- Exiting program");
return;
};

GPCRSS (SSID,NCID,SSTYPE,SSDESC);
GPASSW (WSID, SSID) ;
GPSSS (SSID) ;
GPQCID (WSID,1,10,&ERRIND,&TOTNUM,CTIDAR);
GPXCR (WSID,CTIDAR[0],START,NUM,COLORS);
GPACFO (WSID,CSID,FONT);
GPOPST (STRUC);
GPTXCI (WHITE);
GPAH(HIGHT1);
GPAN2(POS1,11,TEXT1);
GPCLST();
GPUPWS (WSID,PRFORM);
GPMSG (WSID,LMSG,MSG);
GPLCMO (WSID, 1, 3, 2);
while (1)
{
GPAWEV(0.0, &MAJ, &CLAS, &MIN);
if (CLAS == 1)
break;
}
GPDAFO (WSID,CSID,FONT); GPMSG (WSID,0,MSG);
```

```
GPDAST0:
GPCLPH0:
}
```

PHIGS Extension to X
PEX

Agenda

- *What Is PEX*
- *How To...*
- *PEX Demos*

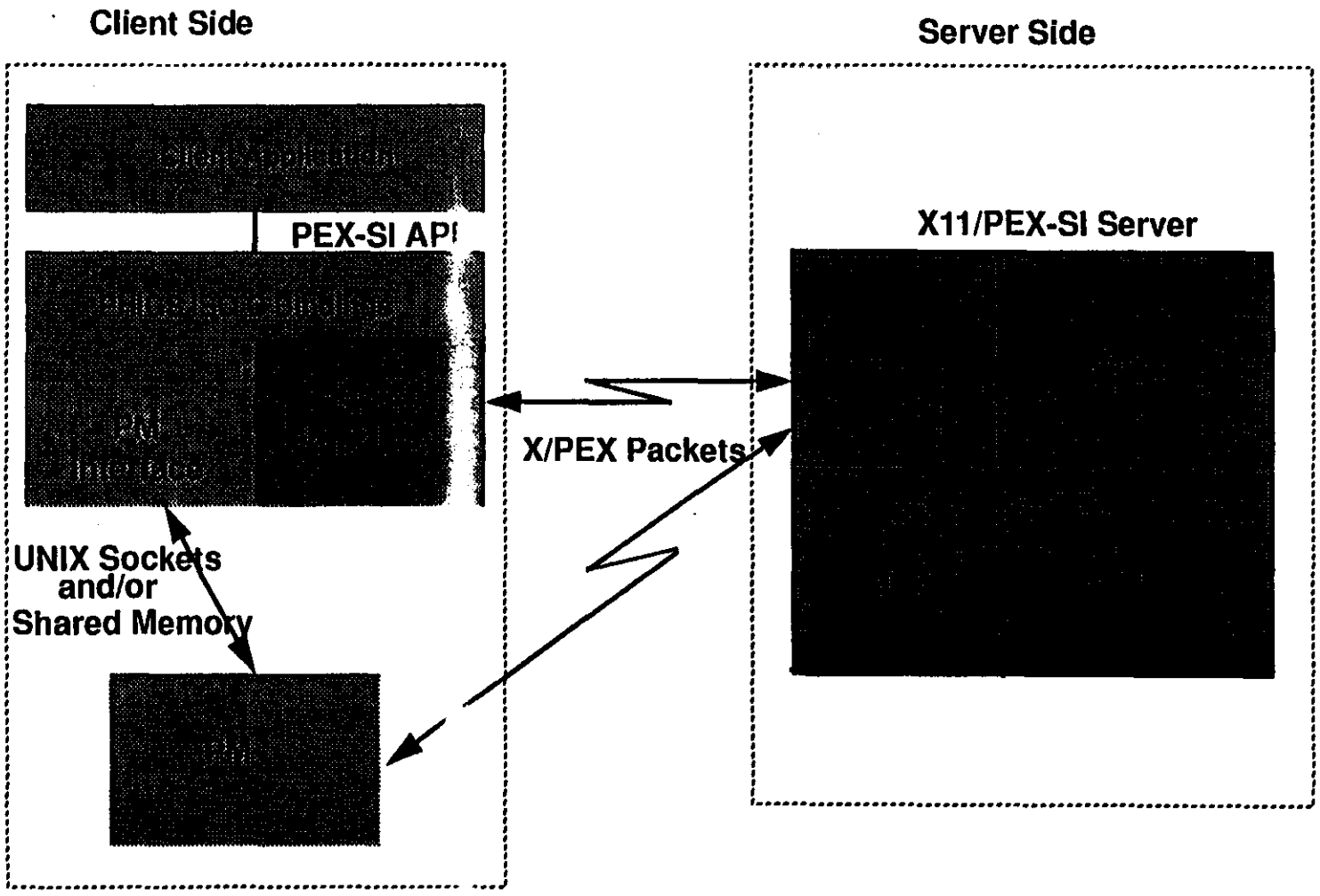
What Is PEX

- *3D PHIGS Extension To X*
- *MIT's Release 5.1*
- *Technical Details:*
  - The PEX Architecture
  - The PEX Protocol
  - The PEXlib API
  - A List Of PEX Modules
  - PEX Documentation

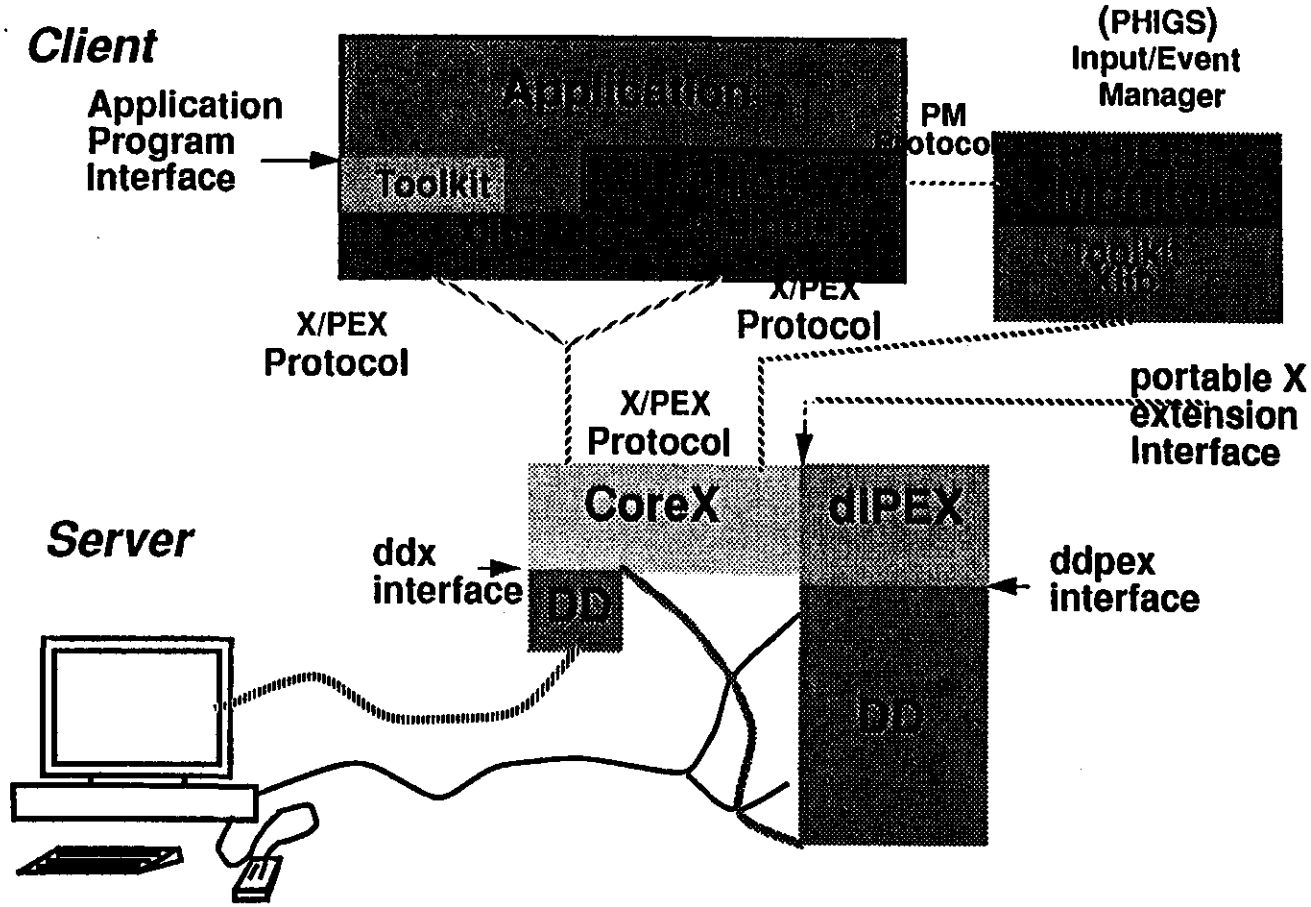
PEX Architecture

- *High Level Diagram*
- *Two Event Models*
  - Tools Model
  - Drawable Model (PHIGS Monitor)
- *Three Structure Store Scenarios*

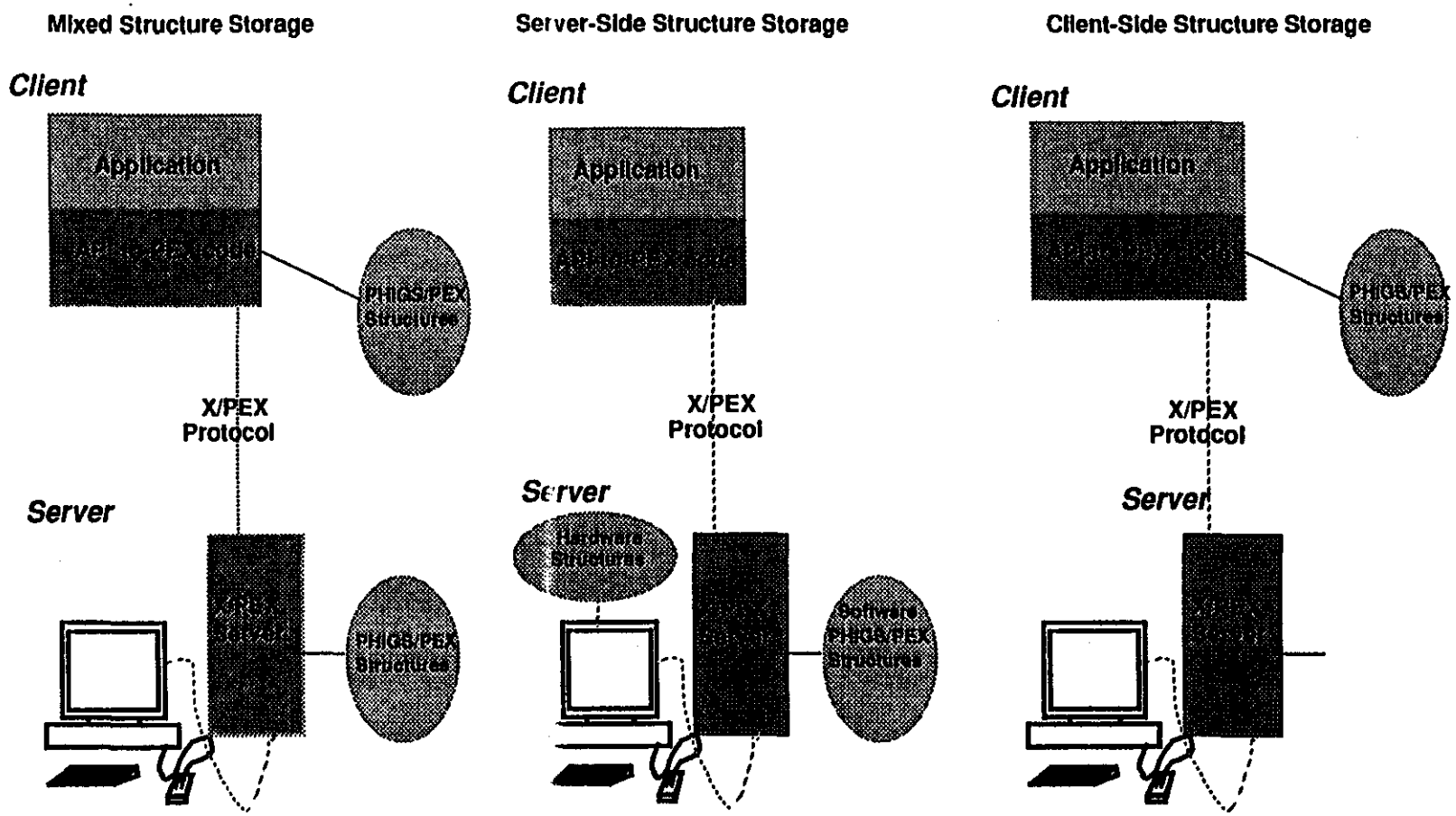
High Level Diagram



Two Event Handling Models



Three Structure Store Scenarios



The PEX Protocol

- *Standard X Extension: PEX-ME*
- *PEX Protocol Specifications*
- *PEX Resources*

PEX Protocol Specifications

- *Protocol Format*
- *Syntax*
- *Naming Conventions*
- *Errors*
- *Events*
- *Padding*
- *Extension Info*

The PEX Resources

- *Output Commands*
- *Look-up Table*
- *Pipeline Context*
- *Rendering*
- *Structures*
- *Name Sets*
- *Search Context*
- *PHIGS Workstations*
- *Picking*
- *PEX Fonts*
- *PHIS/PHIGS PLUS Specifications*

The PEXlib API

- *PEXlib, PEX-SI, PEX Toolkit (PEX 5.1)*
- *Scope Of PEXlib*
  - PHIGS
  - PHIGS+
- *Naming Conventions For Function Calls*
- *Include Files*
  - #include phigs/phigs.h
- *Compile Options*
  - Libraries: -l phigs -l X11 -lm
  - CFLAGS: -D AIXV3 -D PEX
- *PEXlib Will Become Finalized November '92*

A List Of PEX Modules

- *PEX-ME*
- *PEX-SI, PEXlib*
- *PEX-PM*
- *PEX-IM*
- *InsPEX*
- *plbpex - PEX GPC Benchmark*
- *pexscope*

PEX Documentation

■ *Hardcopies*

- ○ PEX Users Guide SH23-0020
- ○ PEX Graphics Language, Programmers Reference Guide SH-0021

■ *MIT Tape Contains Softcopies For*

- ○ PEX Protocol Specifications
- ○ PEX API Users Guide
- ○ PEX API Architecture Guide
- ○ PEX API Porting Guide and Implementation Details

How To...

- *Start The PEX Server*
- *Build PEX Demos*
- *Run PEX Demos*

Demos

- *Where To Find PEX Sample Programs*
- *Demo Programs*
  - "Original" PEX Distribution Tape Demos
  - PEX Function Call Sample Programs
- *Problems*
  - What Does NOT Run Together
  - Hardware Considerations

C++ / Fortran Interlanguage Prototype

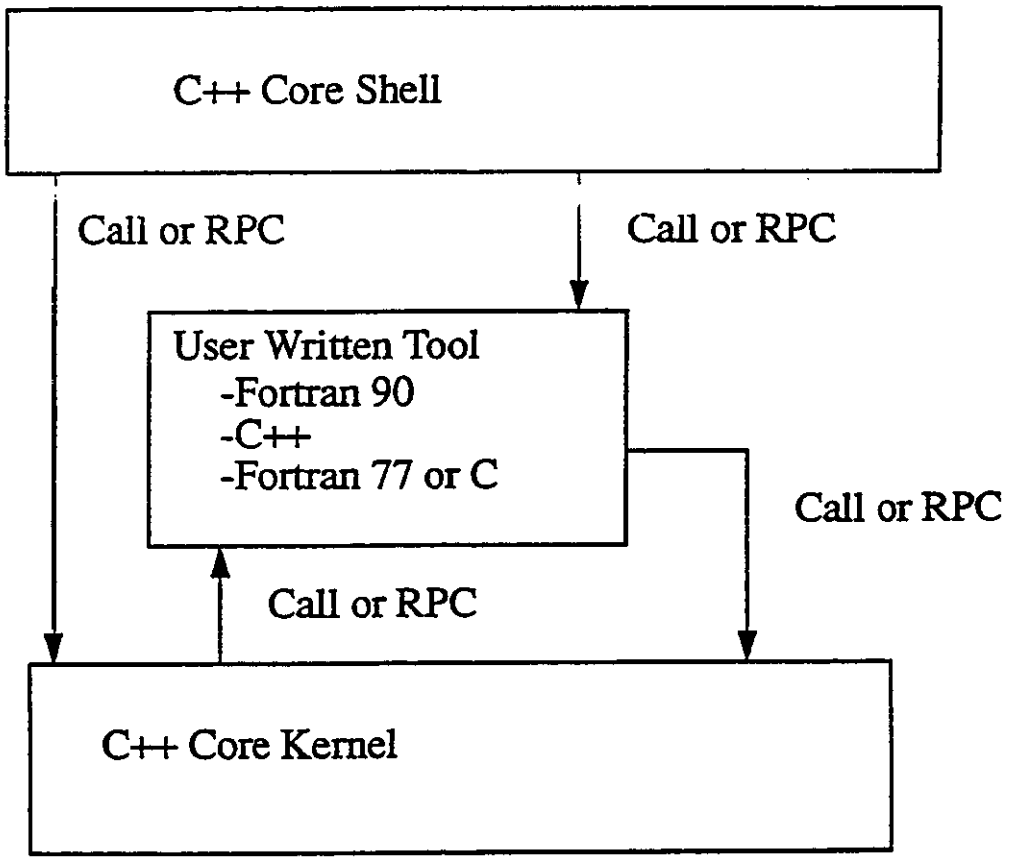
IBM FSC High Performance Solutions Integration

Glenn Kubena
Alicia Harvey
Larry Roberts

May 11, 1993

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

General Assumptions



Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

Prototype Objectives

Short Term

- Determine C++ / Fortran 77 compatibility
  - -Single Process on single vendor Unix platform (IBM RS6000)[1]
  - -Single Process ported to other Unix platforms (Sun, SGI, HP)[2]
  - -Distributed multi-process using UNIX RPC on Berkeley Sockets
  - -- On single vendor (RS6000 cluster)
  - -- Or multi vendor (RS6000, Sun, ...)
- Document C++ / Fortran 77 "interface" rules, issues and recommended solutions for SDC[3]

Long Term

- Determine C++ / native Fortran 90 compatibility (IBM beta compiler available ~July)
- Update "interface" rules, programming style and examples
- Evaluate "open" RPC options, e.g., OSF DCE and OMG
- Evaluate IDL options, e.g., Sun's RPCGEN and OSF's IDL for DCE

---

1. Complete
2. Complete for Sun, SGI and HP on PDSF
3. Complete for calls within a single process on IBM, Sun, SGI and HP (See Appendix 1)

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

Results: Single Process on RS6000

Test Case Setup

```
//C++ fortran tool prototype declarations
extern "FORTRAN" array_add (int &, double [3][2]) ;
extern "FORTRAN" ftocpp (char *, double &) ;
int i ; double test_array1 [3][2] ; char *char1 ; double f1 ;
```

```
// C++ analysis_shell.C
#include "fortran_proto.h"
#include "cpp_proto.h"

main ()
{
// call Fortran routines
array_add (i, test_array1) ;
ftocpp (char1, f1);
//ftocpp in turn calls C++
}
```

```
subroutine array_add(i,a)
integer*4 i
real*8 a
dimension a(3,2)
...
end
```

```
subroutine ftocpp (i, f)
integer*4 i
real*8 f
character*6 cpass
call core1 (cpass, f)
end
```

```
//C++ core declarations
extern "C" core1(char *,
double &, int) ;
...
```

```
// C++ core1.C definition
#include "cpp_proto.h"
core1 (char *a, double &x,
int lena)
{...
}
```


Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

C++ / Fortran 77 Interface Rules

- Fortran defaults to pass by reference ; C++ defaults to pass by value except when passing arrays and subroutine names
- Fortran stores arrays in column order; C++ stores arrays in row order
- C++ character strings are terminated by a null character ('\0'); Fortran strings are not null terminated
- Fortran subroutine names declared in C++ must be all lower case
- C++ is case sensitive; Fortran is not unless compiled with the -U option, ==> affect C++ program names called from Fortran

- IBM unique RS6000 Fortran and C++ interface rules:

  Must use extern "FORTRAN" to declare a Fortran routine used by C++

  Must use extern "C" to declare a C++ routine called by Fortran

- SGI unique interface rules:

  Must use extern "C" to declare Fortran routines callable by C++ and C++ routines callable by Fortran

  In the C++ source, must append an underscore to the name of all routines (both C++ and Fortran) that will receive interlanguage calls

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

- HP unique interface rules:

  Must use extern "C" to declare Fortran routines callable by C++ and C++ routines callable by Fortran

- Sun unique interface rules:

  Must use extern "C" to declare Fortran routines callable by C++ and C++ routines callable by Fortran

  In C++ source, must append an underscore to the name of all Fortran routines that will be called by C++

  In the Fortran source, must either use EXTERNAL and a C() pragma to declare C++ routines callable by Fortran. Or, in the C++ source, append an underscore to the subroutine name in the function declaration and definition

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

C++ / F77 Interface Recommendations for SDC[1]

Case Sensitivity

Name all C++ functions that will be called from Fortran in lower case

Rationale : Avoids encumbering Fortran programmers with interlanguage issues

Parameter Passing

When calling Fortran from C++, explicitly use pass by reference in C++ (implicit for arrays)

Rationale : Avoids encumbering Fortran programmers with interlanguage issues and allows use of existing code

Linkage Specifier

Sun, HP, IBM and SGI all use the "C" linkage specifier for Fortran to C++.

Sun, HP, and SGI all use the "C" linkage specifier for C++ to Fortran. IBM uses the "Fortran" linkage specifer for C++ to Fortran

Recommend using conditional compile (only necessary in C++ code); Therefore, Fortran programmer is not encumbered
<note: this is hard coded in our examples>

1. For a complete writeup see Appendix 1 (RPC issues not addressed)

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

Underscores in Names of Routines

IBM and HP ==> underscores not required
SUN and SGI ==> must append underscore to function name for Fortran calling C++ or C++ calling Fortran

No good solution other than conditional compile (within C++ code only); therefore Fortran programmer is not encumbered

Character Data

Code must be added in C++ to compute the length of character strings and add the length as an extra argument (both when calling Fortran or when being called by Fortran)

This approach requires no change in the Fortran code

Array Data

The C++ programmer should accommodate the differences in array indexing to make it transparent to the Fortran programmer

I/O Considerations

Since I/O within Fortran under a C++ main will not work on at least one system (Sun) this should be re-visted as part of Core Data Services (i.e. all I/O might be done only in C++ by Core Data Services)

C++/Fortran77 Type Comparison (see Appendix 1)

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

Code Examples

Refer to Appendix 2 for Version 1 of prototype code for IBM, Sun, SGI and HP

Bottom Line :

1. Fortran code is the identical on all platforms
2. C++ code is unique per platform <did not use conditional compiles>

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

Appendix 1

C++ / Fortran 77 Interlanguage Interface Issues

<Note : This Appendix is being distributed as a separate PS file>

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

C++ / Fortran 77 Interlanguage Interface Issues

Alicia E. Harvey

This document represents information gathered (at times verbatim) from IBM and Sun reference manuals and from Bjarne Stroustrup's *The C++ Programming Language* (ed. 2). Sprinkled through out are useful tidbits gained from my experiences making a small multilanguage prototype run on the RS/6000 and porting it to Sun, HP, and SGI platforms. Several recommendations are also given.

RPC issues are not addressed in this document.

1.0 Case Sensitivity

C++ is case sensitive *i.e.*, C++ differentiates between uppercase and lowercase. Fortran is not case sensitive. This disparity may be dealt with as follows:

> In the C++ source, make the name of C++ functions that will be called from Fortran all lowercase.

Depending on the particular compiler, another solution is to:

> Compile the Fortran program with the -U option, which tells Fortran to preserve existing uppercase / lowercase distinctions.

Recommendation:

> In order to maintain compatibility with existing Fortran code and avoid encumbering the Fortran programmer with interlanguage issues, name C++ functions that will be called from Fortran in all lower case.

2.0 Parameter Passing

Fortran defaults to pass-by-reference. In this style of parameter passing, the address of the parameter is passed. C++ defaults to pass-by-value, except when passing arrays and subroutine names. In this style of parameter passing, the actual value of the parameter is passed. One method for dealing with this disparity is to explicitly use pass-by-reference in C++. In this manner, the Fortran code need not be modified in order to deal with C++ interlanguage calls.

When passing parameters by reference:

> When writing a C++ function that is to be callable from Fortran, declare all the parameters as references.

When writing a C++ function that calls a routine written in Fortran, all arguments must be references.

Recommendation:

In order to maintain compatibility with existing Fortran code and avoid encumbering the Fortran programmer with interlanguage issues, explicitly use pass-by-reference in C++.

3.0 Linkage Specifiers

A linkage specifier is used in the declaration of all non-C++ functions and in the declaration of all C++ functions that may be called from other languages. The form for this is

```
extern "C" void fn();
extern "FORTRAN" void f2();
```

for a C++ function callable from non-C++ functions. This says that fn() should be linked according to the C linkage conventions. Note that the C in extern "C" names a linkage convention and *not* a language.

3.1 IBM Specific Notes

IBM uses the linkage specifier "FORTRAN" in the declaration of Fortran functions that will be called from other languages. And, "C" in the declaration of C++ functions that will be called from other languages.

3.2 Sun, HP, and SGI Specific Notes

Sun, HP, and SGI use the linkage specifier "C" in the declaration of all functions that are candidates for interlanguage calls involving C++ and Fortran.

4.0 Character Data

C++ functions expect that strings are terminated by the null character (\0). Fortran does not append the null character to the end of character literals. It is the coders responsibility to concatenate the null character to the end of any Fortran string data that is passed to C++ routines.

The only character type in Fortran is CHARACTER, which is stored as a set of contiguous bytes, one character per byte. The length of a Fortran character variable or character array element is determined at compile time and is therefore static. Character lengths are returned by the Fortran intrinsic function LEN.

When you pass a Fortran string data item as a parameter, the address of the beginning of the string is passed along with a parameter of type int that is the length of the string. This parameter is added to the end of the declared parameter list. Likewise, if there are several strings passed, a whole list of string lengths will be added to the end of the declared parameter list.

Thus, when Fortran is calling a C++ function and passing a string, the C++ function must *explicitly* provide arguments at the end of the Fortran calling sequence to accommodate these lengths. And, when C++ is calling a Fortran subroutine, this length must be *explicitly* pass this length by adding it to the end of the calling sequence for the Fortran subroutine.

Character strings in C++ are typically stored as arrays of the type char. The char data type stores one character per byte; therefore, an array of char is stored exactly like a Fortran CHARACTER variable.

It is worth noting that, because both Fortran and C++ characterize strings in the same manner using the address of the first character in the string in passing strings between Fortran and C++, the C++ type char * is the appropriate type to use for string parameters. Also, the length parameter that Fortran uses in characterizing a string is handled by value, *i.e.*, it is of C++ type int.

When an array of char is used to represent a character string in C++, the end of the string is indicated by the null character (\0). C++ functions expect that strings are null terminated. Fortran does not append the null character to the end of character literals. It is the coders responsibility to concatenate the null character to the end of any Fortran character data that is passed to C++ routines.

One final note to C++ programmers, depending on the platform, it may or may not be necessary to #include strings.h in order to perform string manipulations. Some platforms provide that capability in stdlib.h .

5.0 Underscore in Names of Routines

5.1 Sun Specific Notes

The Fortran compiler normally appends an underscore (_) to the names of subprograms, for both a subprogram and a call to a subprogram. This distinguishes it from C++ procedures or external variables with the same user assigned name. There are two common solutions to the underscore problem. When dealing with calls to C++ functions:

In the C++ function, change the name of the function by appending an underscore to the end of that name.

Or,

Use the C() pragma to tell the Fortran compiler to omit those trailing underscores.

The C() pragma directive takes the names of external functions as arguments. It specifies that these functions are written in the C++ language, so the Fortran compiler does not append an underscore to such names.

The C() directive for a particular function must appear before the first reference to that function.

It must appear in each subprogram that contains such a reference.

The conventional usage is:

```
EXTERNAL ABC, XYZ !$PRAGMA C( ABC, XYZ )
```

If you use this pragma, then in the C++ function you must not append an underscore to those names.

Use one or the other, but not both.

Because of this Sun Fortran compiler convention of appending an underscore to the names of subprograms, within C++ an underscore must be appended to the names of Fortran functions when they are called.

5.2 SGI Specific Notes

The SGI Fortran compiler appends underscores in like fashion, however, no mechanisms are provided in either C++ or Fortran to work around them. Therefore, for any C++ function that is called from Fortran, the name given that function must end with an underscore. For example:

```
ex.C: void example_()
          {...}
ex.f: ...
      CALL EXAMPLE
      ...
```

And likewise, when a Fortran subroutine is called from C++, an underscore must be appended to that function name in the call.

5.3 IBM and HP Specific Notes

Neither IBM nor HP require the use of appended underscores anywhere in either the C++ or Fortran code. The linkage specifiers are sufficient to assure compatibility.

Recommendation:

For HP and IBM, use the appropriate linkage specifier and don't worry about underscores.

For Sun and SGI, within C++, append an underscore to the names of Fortran functions when they are called. And, when naming C++ functions that may be called from Fortran, use an underscore as the last character in the name (as shown in the example in section 5.2 above).

6.0 Array Data

6.1 Array Indexing

Indexing of C++ arrays always starts at 0 (zero). By default, indexing of Fortran arrays starts at 1. There are two common ways of approaching this:

Use the Fortran default, and keep in mind that Fortran element b(2) is equivalent to C++ element b[1].

Or,

Specify that the Fortran array start at 0 rather than 1. For example:

```
INTEGER B(0:2)
```

This way the Fortran element B(1) is equivalent to the C++ element b[1].

6.2 Array Order

Fortran arrays are stored in column-major order. C++ arrays are stored in row-major order. For one-dimensional arrays, this is no problem. For two-dimensional arrays, this is only a minor problem, as long as the array is square. Sometimes it is enough to just switch subscripts. For two-dimensional arrays that are not square, it is not enough to just switch subscripts.

7.0 I/O Considerations

Mixing Fortran I/O and C I/O is not recommended. If you must mix them, it is usually safer to pick one and stick with it, rather than alternating.

The Fortran I/O library is implemented largely on top of the C standard I/O library. Every open unit in a Fortran program has an associated standard I/O file structure. For the stdin, stdout, and stderr streams, the file structure need not be explicitly referenced, so it is possible to share them.

If a Fortran main program calls C++, then before the Fortran program starts, the Fortran I/O library is initialized to connect units 0, 5, and 6 to stderr, stdin, and stdout, respectively. The C++ function must take the Fortran I/O environment into consideration to perform I/O on open file descriptors.

7.1 Fortran I/O

I used the C++ compiler to invoke the linker.

All of the platforms I worked with, except HP, required that one or more libraries be linked in explicitly in order for Fortran I/O to work.

7.1.1 IBM Specific Notes

To get the proper libraries linked, the XL Fortran library, libxlf, must be linked explicitly. For example:

```
xlC foo.o bar.o -lxlf
```

7.1.2 Sun Specific Notes

Sun Fortran 77 exhibits a library bug in dealing with interlanguage calls and I/O. I/O in a Fortran subroutine will only work if it is being called from a main that is also written in Fortran.

7.1.3 SGI Specific Notes

To get the proper libraries linked, the Fortran 77 libraries libI77 and libisam must be linked explicitly and in order. For example:

```
CC foo.o bar.o -lI77 -lisam
```

8.0 Correspondence of C++ and Fortran Data Types

C++ Data Types	Fortran Data Types
wchar_t	
char	CHARACTER
signed char	INTEGER*1
unsigned char	LOGICAL*1
short signed int	INTEGER*2
short unsigned int	LOGICAL*2
signed int long int	INTEGER*4
unsigned int	LOGICAL*4
float	REAL REAL*4
structure of two floats	COMPLEX COMPLEX*8
double	REAL*8 DOUBLE PRECISION
structure of two doubles	COMPLEX*16 DOUBLE COMPLEX
C style struct	
enumeration	INTEGER*4

C++ Data Types	Fortran Data Types
char[n]	CHARACTER*n
array pointer (*) to type	Dimensioned variable (transposed)
pointer (*) to function	Functional Parameter

Appendix 2

Version 1 of C++ / Fortran 77 prototype code

This code is available on the PDSF on

/home/dssg0/sdc/harvey/interface/6000

/home/dssg0/sdc/harvey/interface/hp

/home/dssg0/sdc/harvey/interface/sun

/home/dssg0/sdc/harvey/interface/sgi

All Fortran Code is the same for all platforms

C++ code is different per platform (cond. compile not used)

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

Fortran Code Listings

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

```
C  Name: array_add
C  Description: The fortran subroutine adds one element of two arrays
C  Author : Glenn Kubena
C  Date : 4/16/93, vers. 1
C  Notes:
      SUBROUTINE  ARRAY_ADD(I, J, A, B, C)
      INTEGER*4 I,J
      REAL*8 A,B,C
      DIMENSION A(3,2), B(3,2), C(3,2)
      C(I,J) = A(I,J) + B(I,J)
      RETURN
      END
```

```
C  Name: array2_add
C  Description: The fortran subroutine adds ~~one~~ all elements of two arrays
C  Author : Glenn Kubena
C  Date : 4/16/93, vers. 1
C  Notes:
      SUBROUTINE  ARRAY2_ADD(I, J, A, B, C)
      INTEGER*4 I,J
      REAL*8 A,B,C
      DIMENSION A(3,2), B(3,2), C(3,2)
      DO I=1,3
      DO J=1,2
      C(I,J) = A(I,J) + B(I,J)
      END DO
      END DO
      RETURN
      END
```

```
C   Name: ftocpp
C   Description: The fortran subroutine calls C++
C   Author : Glenn Kubena
C   Date : 4/19/93, vers. 1
C   Notes:
      SUBROUTINE  FTOCPP(I, CPASS, CPASS2, A, B, C, D)
      INTEGER*4 I
      CHARACTER*6 CPASS , CPASS2
      REAL*8 A,B,C, D
      DIMENSION A(3,2), B(3,2), C(3,2)
      DO I=1,3
      DO J=1,2
      C(I,J) = A(I,J) * B(I,J)
      END DO
      END DO
C   CALL C++
C   MAKE IT EASY ON THE FORTRAN PROGRAMMER
C   PASS EVERYTHING BY REFERENCE
C   I.E. DO NOT USE %VAL

C     TRY OUT SOME I/O FIRST
      WRITE (6,*) "FTOCPP: ", CPASS2

      CALL CORE1 (CPASS, CPASS2, C, D, I)
      RETURN
      END
```

IBM C++ Code Listings

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

```
//  Name: analysis_shell.C
//  Description: C++ Analysis Shell Test Code Fragment
//  Author : Glenn Kubena
//  Date : 4/15/93, Vers. 1
//  Notes :
//
//****************
//
// Modified by: Alicia E. Harvey
// Date: 4/29/93
// Modification: Corrected handling of character strings.
//

#include <iostream.h>

#include "fortran_proto.h"  //fortran function prototype definitions

main ()
{
  cout<< "C++ SHELL: Welcome to the C++ Shell Prototype" << endl;

  // *****
  cout<< "C++ SHELL: We'll default to run the Fortran routine array_add,for now"
            << endl;
  cout << "C++ SHELL: Enter 3 values for test array 1, col 1" << endl;
  cin >> test_array1 [0][0] >> test_array1 [1][0] >> test_array1 [2][0] ;
  cout<< "C++ SHELL: Enter 3 values for test array 1, col 2" << endl;
  cin >> test_array1 [0][1] >> test_array1 [1][1] >> test_array1 [2][1] ;
  //note: Fortran indexes arrays 1..n
  //      C++ indexes arrays 0..(n-1)
  // call Fortran, Note: test_array1 points to 1st element
  //                     C++ passes arrays by reference (pointer)
  int i=1, j ;
  while (i <= 3)
  {
    j=1 ;
    while (j <= 2)
    {
    array_add (i,j, test_array1, test_array2, test_array3);
    // cout << "C++ SHELL: array result for " <<  " row " << i-1 << " and col "
       //   << j-1 << " = " << test_array3[i-1][j-1] <<  endl;
    ++j ;
    }
  ++i ;
  }
  cout << "C++ SHELL: test_array3 results are: " << endl;
  cout << test_array3[0][0] << " " << test_array3[0][1] << endl;
  cout << test_array3[1][0] << " " << test_array3[1][1] << endl;
  cout << test_array3[2][0] << " " << test_array3[2][1] << endl;
  // *****

  // try a call where Fortran adds the entire array
  array2_add (i,j, test_array1, test_array2, test_array4);
  cout << "C++ SHELL: test_array4 results are: " << endl;
  cout << test_array4[0][0] << " " << test_array4[0][1] << endl;
  cout << test_array4[1][0] << " " << test_array4[1][1] << endl;
  cout << test_array4[2][0] << " " << test_array4[2][1] << endl;
  // *****

  // call Fortran which in turn calls a C++ "core" routine
  cout << "C++ SHELL : Next we'll call ftocpp, which is a fortran routine"
       << " which calls a C++ Core routine, core1 " << endl;
  cout << "C++ SHELL : Enter 6 characters of text" << endl;
```

```
    char *char2, *char1;    // Declare character strings to pass to Fortran.
    int len1, len2;         // Lengths must be passed too.

    // Allocate space for the string we want to read in.  Then read it.
    char2 = new char[10];
    cin >> char2;

    // We need the length of the character string so we can pass it along
    // to the Fortran subroutine.
    len2 = strlen(char2);

    double f1;

    // Let's make sure the string looks like we think it should.
    cout << "shell: The string is :" << char2 << ":\n" << endl;

    // Remember, we need to pass the string lengths to Fortran  explicitly
    // at the end of the parameter list.
    ftocpp (i, char1, char2, test_array1, test_array2, test_array3, f1, len1,
            len2);

}
```

```
// Name: fortran_proto.h
// Description : This file contains the Fortran subroutin prototype definitions
//               which are used by the C++  Shell test code
// Author: Glenn Kubena
// Date: 4/15/93, Vers. 1
// Notes:
//   1. Fortran defaults to call by reference
//   2. C++ defaults to call by value, with some exceptions (e.g. arrays)
//   3. IBM XLF requires that fortran function or subroutine names use
//      lower case only
//   4. C++ stores arrays in row order, Fortran stores arrays in col order
//****************
//
// Modified by: Alicia E. Harvey
// Date: 4/29/93
// Modification: Corrected handling of character strings.
//

extern "FORTRAN"  array_add (int &, int &, double [3][2],
                                    double [3][2],
                                    double [3][2] ) ;
                                    //add 2 arrays
                                    //note that integers
                                    // must be explicitly
                                    //passed by reference but not the arrays
double test_array1 [3][2] ; //test array 1 , double float
double test_array2 [3][2] ={7.0,8.0,9.0,10.0,11.0,12.0} ; //test array 2
double test_array3 [3][2] ={0.0,0.0,0.0,0.0,0.0,0.0} ; //test array 3
//****************
extern "FORTRAN"  array2_add (int &, int &, double [3][2],
                                    double [3][2],
                                    double [3][2] ) ; //array2_add

double test_array4 [3][2] ={0.0,0.0,0.0,0.0,0.0,0.0} ; //test array 4
//****************
//call the following Fortran routine to call C++ core1 routine
// Remember to pass the string lengths explicitly.
extern "FORTRAN" ftocpp (int &, char *, char *, double [3][2],
                                     double [3][2],
                                     double [3][2],
                                     double &,
                                     int, int ) ;
```

```
// Name: core1.C
// Description: Includes header files needed by the C++ function "core1".
// Author: Glenn Kubena
// Date: 4/19/93
//

#include <iostream.h>
#include "cpp_proto.h"
#include "core1.h"
```

```
// Name: cpp_proto.h
// Description : This file contains the C++ prototype definitions
//               which are used by Fortran code to call the C++ "Core"
//               routines
// Author: Glenn Kubena
// Date: 4/19/93, Vers. 1
// Notes:
//   1. Fortran defaults to call by reference
//   2. C++ defaults to call by value, with some exceptions (e.g. arrays)
//   3. IBM XLF requires that fortran function or subroutine names use
//      lower case only
//   4. C++ stores arrays in row order, Fortran stores arrays in col order
//   5. IBM XLF requires the "C" to tell the C++ linkage editor that this
//      routine is called from another language (weird but true)
//****************
//the following are C++ core routines prototypes
//

 // We need to deal explicitly with the lengths of any strings that are
 // passed from Fortran.  Thus the "int"s at the end of the parameter list.
 extern "C" void corel (char *, char *, double [3][2],
                                        double &,
                                        int &,
                                        int , int ) ;
```

```
// Name : core1.h
// Description : C++ Sample Core function definition
// Author : Glenn Kubena
// Date : 4/19/93
// Notes :
//
//************************
//
// Modified by : Alicia E. Harvey
// Date : 4/29/93
// Modification : Corrected handling of character strings.
//

#include <stdlib.h>
#include <stdio.h>

// Strings are passed as "char *" and the string lengths as "int".
void core1 (char *a, char *b, double [3][2], double &x, int &n, int lena,
            int lenb)
   {
     // We must construct a C style null terminated string form the Fortran
     // string that is passed to us.
     char *strb;

     // Make sure we know where we are and that the string looks like we
     // think it should.
     cout << "inside core1" << endl ;

     cout << "String b is :" << b << ":" << endl;
     cout << "The length of the string is: " << lenb << endl;

     // Allocate space for the C style string we're going to create.
     strb = new char[lenb + 1];
     // Copy the Fortran string into the C style string.
     strncpy(strb, b, lenb);
     // Add the null terminator.
     strb[lenb] = '\0';

     // Check that our C style string is correct.
     cout << "core1: The string is :" << strb << ":" << endl;

   }
```

HP C++ Code Listings

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

```
//  Name: analysis_shell.C
//  Description: C++ Analysis Shell Test Code Fragment
//  Author : Glenn Kubena
//  Date : 4/15/93, Vers. 1
//  Notes :
//
//************************
//
//  Modified by: Alicia E. Harvey
//  Date: 4/29/93
//  Modification: Corrected handling of character strings.
//

#include <iostream.h>

#include "fortran_proto.h"  //fortran function prototype definitions

main ()
{
  cout<< "C++ SHELL: Welcome to the C++ Shell Prototype" << endl;

  // *****
  cout<< "C++ SHELL: We'll default to run the Fortran routine array_add,for now"
            << endl;
  cout<< "C++ SHELL: Enter 3 values for test array 1, col 1" << endl;
  cin >> test_array1 [0][0] >> test_array1 [1][0] >> test_array1 [2][0] ;
  cout<< "C++ SHELL: Enter 3 values for test array 1, col 2" << endl;
  cin >> test_array1 [0][1] >> test_array1 [1][1] >> test_array1 [2][1] ;
  //note: Fortran indexes arrays 1..n
  //      C++ indexes arrays 0..(n-1)
  // call Fortran, Note: test_array1 points to 1st element
  //                     C++ passes arrays by reference (pointer)
  int i=1, j ;
  while (i <= 3)
  {
    j=1 ;
    while (j <= 2)
    {
    array_add (i,j, test_array1, test_array2, test_array3);
    // cout << "C++ SHELL: array result for " <<  " row " << i-1 << " and col "
       //   << j-1 << " = " << test_array3[i-1][j-1] <<  endl;
    ++j ;
    }
  ++i ;
  }
  cout << "C++ SHELL: test_array3 results are: " << endl;
  cout << test_array3[0][0] << " " << test_array3[0][1] << endl;
  cout << test_array3[1][0] << " " << test_array3[1][1] << endl;
  cout << test_array3[2][0] << " " << test_array3[2][1] << endl;
  // *****

  // try a call where Fortran adds the entire array
  array2_add (i,j, test_array1, test_array2, test_array4);
  cout << "C++ SHELL: test_array4 results are: " << endl;
  cout << test_array4[0][0] << " " << test_array4[0][1] << endl;
  cout << test_array4[1][0] << " " << test_array4[1][1] << endl;
  cout << test_array4[2][0] << " " << test_array4[2][1] << endl;
  // *****

  // call Fortran which in turn calls a C++ "core" routine
  cout << "C++ SHELL : Next we'll call ftocpp, which is a fortran routine"
       << " which calls a C++ Core routine, core1 " << endl;
  cout << "C++ SHELL : Enter 6 characters of text" << endl;
```

```
  char *char2, *char1;  // Declare character strings to pass to Fortran.
  int len1, len2;       // Lengths must be passed too.

  // Allocate space for the string we want to read in.  Then read it.
  char2 = new char[10];
  cin >> char2;

  // We need the length of the character string so we can pass it along
  // to the Fortran subroutine.
  len2 = strlen(char2);

  double f1;

  // Let's make sure the string looks like we think it should.
  cout << "shell: The string is :" << char2 << ":\n" << endl;

  // Remember, we need to pass the string lengths to Fortran  explicitly
  // at the end of the parameter list.
  ftocpp (i, char1, char2, test_array1, test_array2, test_array3, f1, len1,
          len2);

}
```

```
// Name : core1.h
// Description : C++ Sample Core function definition
// Author : Glenn Kubena
// Date : 4/19/93
// Notes :
//
//**************************
//
// Modified by: Alicia E. Harvey
// Date: 4/29/93
// Modification: Corrected handling of character strings.
//

#include <string.h>

// Strings are passed as "char *" and the string lengths as "int".
void core1_ (char *a, char *b, double [3][2], double &x, int &n, int lena,
             int lenb)
   {
     // We must construct a C style null terminated string form the Fortran
     // string that is passed to us.
     char *strb;

     // Make sure we know where we are and that the string looks like we
     // think it should.
     cout << "inside core1" << endl ;

     cout << "String b is :" << b << ":" << endl;
     cout << "The length of the string is: " << lenb << endl;

     // Allocate space for the C style string we're going to create.
     strb = new char[lenb + 1];
     // Copy the Fortran string into the C style string.
     strncpy(strb, b, lenb);
     // Add the null terminator.
     strb[lenb] = '\0';

     // Check that our C style string is correct.
     cout << "core1: The string is :" << strb << ":" << endl;

   }
```

```
// Name: cpp_proto.h
// Description : This file contains the C++ prototype definitions
//               which are used by Fortran code to call the C++ "Core"
//               routines
// Author: Glenn Kubena
// Date: 4/19/93, Vers. 1
// Notes:
//   1. Fortran defaults to call by reference
//   2. C++ defaults to call by value, with some exceptions (e.g. arrays)
//   3. IBM XLF requires that fortran function or subroutine names use
//      lower case only
//   4. C++ stores arrays in row order, Fortran stores arrays in col order
//   5. IBM XLF requires the "C" to tell the C++ linkage editor that this
//      routine is called from another language (weird but true)
//****************
//the following are C++ core routines prototypes
//

 // We need to deal explicitly with the lengths of any strings that are
 // passed from Fortran.  Thus the "int"s at the end of the parameter list.
 extern "C" void corel_ (char *, char *, double [3][2],
                                         double &,
                                         int &,
                                         int , int ) ;
```

```
// Name: core1.C
// Description: Includes header files needed by the C++ function "core1".
// Author: Glenn Kubena
// Date: 4/19/93
//

#include <iostream.h>
#include "cpp_proto.h"
#include "core1.h"
```

```
// Name: fortran_proto.h
// Description : This file contains the Fortran subroutin prototype definitions
//               which are used by the C++  Shell test code
// Author: Glenn Kubena
// Date: 4/15/93, Vers. 1
// Notes:
//   1. Fortran defaults to call by reference
//   2. C++ defaults to call by value, with some exceptions (e.g. arrays)
//   3. IBM XLF requires that fortran function or subroutine names use
//      lower case only
//   4. C++ stores arrays in row order, Fortran stores arrays in col order
//****************
//
// Modified by: Alicia E. Harvey
// Date: 4/29/93
// Modification: Corrected handling of character strings.
//

extern "C"  array_add_ (int &, int &, double [3][2],
                                      double [3][2],
                                      double [3][2] ) ;
                                      //add 2 arrays
                                      //note that integers
                                      // must be explicitly
                                      //passed by reference but not the arrays
double test_array1 [3][2] ; //test array 1 , double float
double test_array2 [3][2] ={7.0,8.0,9.0,10.0,11.0,12.0} ; //test array 2
double test_array3 [3][2] ={0.0,0.0,0.0,0.0,0.0,0.0} ; //test array 3
//****************
extern "C"  array2_add_ (int &, int &, double [3][2],
                                      double [3][2],
                                      double [3][2] ) ; //array2_add

double test_array4 [3][2] ={0.0,0.0,0.0,0.0,0.0,0.0} ; //test array 4
//****************
//call the following Fortran routine to call C++ core1 routine
// Remember to pass the string lengths explicitly.
extern "C" ftocpp_ (int &, char *, char *, double [3][2],
                                       double [3][2],
                                       double [3][2],
                                       double &,
                                       int, int ) ;
```

```
    char *char2, *char1;     // Declare character strings to pass to Fortran.
    int len1, len2;          // Lengths must be passed too.

    // Allocate space for the string we want to read in.  Then read it.
    char2 = new char[10];
    cin >> char2;

    // We need the length of the character string so we can pass it along
    // to the Fortran subroutine.
    len2 = strlen(char2);

    double f1;

    // Let's make sure the string looks like we think it should.
    cout << "shell: The string is :" << char2 << ":\n" << endl;

    // Remember, we need to pass the string lengths to Fortran  explicitly
    // at the end of the parameter list:
    ftocpp_ (i, char1, char2, test_array1, test_array2, test_array3, f1, len1,
            len2);

}
```

```
//  Name: analysis_shell.C
//  Description: C++ Analysis Shell Test Code Fragment
//  Author : Glenn Kubena
//  Date : 4/15/93, Vers. 1
//  Notes :
//
//*****************************
//
//  Modified by: Alicia E. Harvey
//  Date: 4/29/93
//  Modification: Corrected handling of character strings.
//

#include <iostream.h>
#include <string.h>

#include "fortran_proto.h"  //fortran function prototype definitions

main ()
{
  cout<< "C++ SHELL: Welcome to the C++ Shell Prototype" << endl;

  // *****
  cout<< "C++ SHELL: We'll default to run the Fortran routine array_add,for now"
            << endl;
  cout<< "C++ SHELL: Enter 3 values for test array 1, col 1" << endl;
  cin >> test_array1 [0][0] >> test_array1 [1][0] >> test_array1 [2][0] ;
  cout<< "C++ SHELL: Enter 3 values for test array 1, col 2" << endl;
  cin >> test_array1 [0][1] >> test_array1 [1][1] >> test_array1 [2][1] ;
  //note: Fortran indexes arrays 1..n
  //      C++ indexes arrays 0..(n-1)
  // call Fortran, Note: test_array1 points to 1st element
  //                     C++ passes arrays by reference (pointer)
  int i=1, j ;
  while (i <= 3)
  {
    j=1 ;
    while (j <= 2)
    {
    array_add_ (i,j, test_array1, test_array2, test_array3);
    // cout << "C++ SHELL: array result for " <<  " row " << i-1 << " and col "
       //   << j-1 << " = " << test_array3[i-1][j-1] <<  endl;
    ++j ;
    }
  ++i ;
  }
  cout << "C++ SHELL: test_array3 results are: " << endl;
  cout << test_array3[0][0] << " " << test_array3[0][1] << endl;
  cout << test_array3[1][0] << " " << test_array3[1][1] << endl;
  cout << test_array3[2][0] << " " << test_array3[2][1] << endl;
  // *****

  // try a call where Fortran adds the entire array
  array2_add_ (i,j, test_array1, test_array2, test_array4);
  cout << "C++ SHELL: test_array4 results are: " << endl;
  cout << test_array4[0][0] << " " << test_array4[0][1] << endl;
  cout << test_array4[1][0] << " " << test_array4[1][1] << endl;
  cout << test_array4[2][0] << " " << test_array4[2][1] << endl;
  // *****

  // call Fortran which in turn calls a C++ "core" routine
  cout << "C++ SHELL : Next we'll call ftocpp, which is a fortran routine"
       << " which calls a C++ Core routine, core1 " << endl;
  cout << "C++ SHELL : Enter 6 characters of text" << endl;
```

SGI C++ Code Listings

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

```
// Name : corel.h
// Description : C++ Sample Core function definition
// Author : Glenn Kubena
// Date : 4/19/93
// Notes :
//
//*******************************
//
// Modified by : Alicia E. Harvey
// Date : 4/29/93
// Modification : Corrected handling of character strings.
//

#include <stdlib.h>
#include <stdio.h>

// Strings are passed as "char *" and the string lengths as "int".
void corel (char *a, char *b, double [3][2], double &x, int &n, int lena,
            int lenb)
   {
     // We must construct a C style null terminated string form the Fortran
     // string that is passed to us.
     char *strb;

     // Make sure we know where we are and that the string looks like we
     // think it should.
     cout << "inside corel" << endl ;

     cout << "String b is :" << b << ":" << endl;
     cout << "The length of the string is: " << lenb << endl;

     // Allocate space for the C style string we're going to create.
     strb = new char[lenb + 1];
     // Copy the Fortran string into the C style string.
     strncpy(strb, b, lenb);
     // Add the null terminator.
     strb[lenb] = '\0';

     // Check that our C style string is correct.
     cout << "corel: The string is :" << strb << ":" << endl;

   }
```

```
// Name: cpp_proto.h
// Description : This file contains the C++ prototype definitions
//               which are used by Fortran code to call the C++ "Core"
//               routines
// Author: Glenn Kubena
// Date: 4/19/93, Vers. 1
// Notes:
//   1. Fortran defaults to call by reference
//   2. C++ defaults to call by value, with some exceptions (e.g. arrays)
//   3. IBM XLF requires that fortran function or subroutine names use
//      lower case only
//   4. C++ stores arrays in row order, Fortran stores arrays in col order
//   5. IBM XLF requires the "C" to tell the C++ linkage editor that this
//      routine is called from another language (weird but true)
//****************
//the following are C++ core routines prototypes
//

 // We need to deal explicitly with the lengths of any strings that are
 // passed from Fortran.  Thus the "int"s at the end of the parameter list.
 extern "C" void corel (char *, char *, double [3][2],
                                        double &,
                                        int &,
                                        int , int ) ;
```

```
// Description: Includes header files needed by the C++ function "corel".
// Author: Glenn Kubena
// Date: 4/19/93
//

#include <iostream.h>
#include "cpp_proto.h"
#include "corel.h"
```

```
// Name: fortran_proto.h
// Description : This file contains the Fortran subroutin prototype definitions
//               which are used by the C++  Shell test code
// Author: Glenn Kubena
// Date: 4/15/93, Vers. 1
// Notes:
//   1. Fortran defaults to call by reference
//   2. C++ defaults to call by value, with some exceptions (e.g. arrays)
//   3. IBM XLF requires that fortran function or subroutine names use
//      lower case only
//   4. C++ stores arrays in row order, Fortran stores arrays in col order
//****************
//
// Modified by: Alicia E. Harvey
// Date: 4/29/93
// Modification: Corrected handling of character strings.
//

extern "C"  array_add (int &, int &, double [3][2],
                                     double [3][2],
                                     double [3][2] ) ;
                                     //add 2 arrays
                                     //note that integers
                                     // must be explicitly
                                     //passed by reference but not the arrays
double test_array1 [3][2] ; //test array 1 , double float
double test_array2 [3][2] ={7.0,8.0,9.0,10.0,11.0,12.0} ; //test array 2
double test_array3 [3][2] ={0.0,0.0,0.0,0.0,0.0,0.0} ; //test array 3
//****************
extern "C"  array2_add (int &, int &, double [3][2],
                                     double [3][2],
                                     double [3][2] ) ; //array2_add

double test_array4 [3][2] ={0.0,0.0,0.0,0.0,0.0,0.0} ; //test array 4
//****************
//call the following Fortran routine to call C++ core1 routine
// Remember to pass the string lengths explicitly.
extern "C" ftocpp (int &, char *, char *, double [3][2],
                                      double [3][2],
                                      double [3][2],
                                      double &,
                                      int, int ) ;
```

Sun C++ Code Listings

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

```
//  Name: analysis_shell.C
//  Description: C++ Analysis Shell Test Code Fragment
//  Author : Glenn Kubena
//  Date : 4/15/93, Vers. 1
//  Notes :
//
//**********************************
//
//  Modified by: Alicia E. Harvey
//  Date: 4/29/93
//  Modification: Corrected handling of character strings.
//

#include <iostream.h>
#include <string.h>

#include "fortran_proto.h"  //fortran function prototype definitions

main ()
{

  cout<< "C++ SHELL: Welcome to the C++ Shell Prototype" << endl;

  // *****
  cout<< "C++ SHELL: We'll default to run the Fortran routine array_add,for now"
            << endl;
  cout<< "C++ SHELL: Enter 3 values for test array 1, col 1" << endl;
  cin >> test_array1 [0][0] >> test_array1 [1][0] >> test_array1 [2][0] ;
  cout<< "C++ SHELL: Enter 3 values for test array 1, col 2" << endl;
  cin >> test_array1 [0][1] >> test_array1 [1][1] >> test_array1 [2][1] ;
  //note: Fortran indexes arrays 1..n
  //      C++ indexes arrays 0..(n-1)
  // call Fortran, Note: test_array1 points to 1st element
  //                     C++ passes arrays by reference (pointer)
  int i=1, j ;
  while (i <= 3)
  {
    j=1 ;
    while (j <= 2)
    {
    array_add_ (i,j, test_array1, test_array2, test_array3);
    // cout << "C++ SHELL: array result for " <<  " row " << i-1 << " and col "
       //   << j-1 << " = " << test_array3[i-1][j-1] <<  endl;
    ++j ;
    }
  ++i ;
  }
  cout << "C++ SHELL: test_array3 results are: " << endl;
  cout << test_array3[0][0] << " " << test_array3[0][1] << endl;
  cout << test_array3[1][0] << " " << test_array3[1][1] << endl;
  cout << test_array3[2][0] << " " << test_array3[2][1] << endl;
  // *****

  // try a call where Fortran adds the entire array
  array2_add_ (i,j, test_array1, test_array2, test_array4);
  cout << "C++ SHELL: test_array4 results are: " << endl;
  cout << test_array4[0][0] << " " << test_array4[0][1] << endl;
  cout << test_array4[1][0] << " " << test_array4[1][1] << endl;
  cout << test_array4[2][0] << " " << test_array4[2][1] << endl;
  // *****

  // call Fortran which in turn calls a C++ "core" routine
  cout << "C++ SHELL : Next we'll call ftocpp, which is a fortran routine"
       << " which calls a C++ Core routine, core1 " << endl;
```

```
  cout << "C++ SHELL : Enter 6 characters of text" << endl;

  char *char2, *char1;    // Declare character strings to pass to Fortran.
  int len1, len2;         // Lengths must be passed too.

  // Allocate space for the string we want to read in.  Then read it.
  char2 = new char[10];
  cin >> char2;

  // We need the length of the character string so we can pass it along
  // to the Fortran subroutine.
  len2 = strlen(char2);

  double f1;

  // Let's make sure the string looks like we think it should.
  cout << "shell: The string is :" << char2 << ":" << endl;

  // Remember, we need to pass the string lengths to Fortran  explicitly
  // at the end of the parameter list.
  ftocpp_ (i, char1, char2, test_array1, test_array2, test_array3, f1, len1,
           len2);

}
```

```
// Name: fortran_proto.h
// Description : This file contains the Fortran subroutin prototype definitions
//               which are used by the C++  Shell test code
// Author: Glenn Kubena
// Date: 4/15/93, Vers. 1
// Notes:
//   1. Fortran defaults to call by reference
//   2. C++ defaults to call by value, with some exceptions (e.g. arrays)
//   3. IBM XLF requires that fortran function or subroutine names use
//      lower case only
//   4. C++ stores arrays in row order, Fortran stores arrays in col order
//****************
//
// Modified by: Alicia E. Harvey
// Date: 4/29/93
// Modification: Corrected handling of character strings.
//

extern "C" array_add_ (int &, int &, double [3][2],
                                     double [3][2],
                                     double [3][2] ) ;
                                     //add 2 arrays
                                     //note that integers
                                     // must be explicitly
                                     //passed by reference but not the arrays
double test_array1 [3][2] ; //test array 1 , double float
double test_array2 [3][2] ={7.0,8.0,9.0,10.0,11.0,12.0} ; //test array 2
double test_array3 [3][2] ={0.0,0.0,0.0,0.0,0.0,0.0} ; //test array 3
//****************
extern "C" array2_add_ (int &, int &, double [3][2],
                                     double [3][2],
                                     double [3][2] ) ; //array2_add

double test_array4 [3][2] ={0.0,0.0,0.0,0.0,0.0,0.0} ; //test array 4
//****************
//call the following Fortran routine to call C++ core1 routine
// Remember to pass the string lengths explicitly.
extern "C" ftocpp_ (int &, char *, char *, double [3][2],
                                      double [3][2],
                                      double [3][2],
                                      double &,
                                      int, int);
```

```
// Name: core1.C
// Description: Includes header files needed by the C++ function "core1".
// Author: Glenn Kubena
// Date: 4/19/93
//

#include <iostream.h>
#include "cpp_proto.h"
#include "core1.h"
```

```
// Name: cpp_proto.h
// Description : This file contains the C++ prototype definitions
//               which are used by Fortran code to call the C++ "Core"
//               routines
// Author: Glenn Kubena
// Date: 4/19/93, Vers. 1
// Notes:
//   1. Fortran defaults to call by reference
//   2. C++ defaults to call by value, with some exceptions (e.g. arrays)
//   3. IBM XLF requires that fortran function or subroutine names use
//      lower case only
//   4. C++ stores arrays in row order, Fortran stores arrays in col order
//   5. IBM XLF requires the "C" to tell the C++ linkage editor that this
//      routine is called from another language (weird but true)
//****************
//the following are C++ core routines prototypes
//

 // We need to deal explicitly with the lengths of any strings that are
 // passed from Fortran.  Thus the "int"s at the end of the parameter list.
 extern "C" void core1 (char *, char *, double [3][2],
                                        double &,
                                        int &,
                                        int, int);
```

```
// Name : core1.h
// Description : C++ Sample Core function definition
// Author : Glenn Kubena
// Date : 4/19/93
// Notes :
//
//**************************
//
// Modified by: Alicia E. Harvey
// Date: 4/29/93
// Modification: Corrected handling of character strings.
//

#include <stdlib.h>
#include <string.h>

// Strings are passed as "char *" and the string lengths as "int".
void core1 (char *a, char *b, double [3][2], double &x, int &n, int lena,
            int lenb)
   {
     // We must construct a C style null terminated string form the Fortran
     // string that is passed to us.
     char *strb;

     // Make sure we know where we are and that the string looks like we
     // think it should.
     cout << "inside core1" << endl ;

     cout << "String b is :" << b << ":" << endl;
     cout << "The length of the string is: " << lenb << endl;

     // Allocate space for the C style string we're going to create.
     strb = new char[lenb + 1];
     // Copy the Fortran string into the C style string.
     strncpy(strb, b, lenb);
     // Add the null terminator.
     strb[lenb] = '\0';

     // Check that our C style string is correct.
     cout << "core1: The string is :" << strb << ":" << endl;

   }
```

SDC Computing Workshop, Day 3.

1. Utopia Batch System / A.Fry.

* General introduction to Utopia.

* Experience with the system on PDSF and the SDC machine.

* Happy with Utopia, proposed long term solution.

2. SDC Workgroup Cluster / A.Fry.

* Idea: Move all SDC machines into a cluster. In progress.

* Goals: Move day-to-day work from PDSF to a local cluster.

  More control over SDC account

  Local tests without PDSF support

  Better use of resources.

  Resources: 5 HP's, 30 Sun's (with old PDSF hardware).

* Question: Does it need lab resources?

The setup looks very similar to the proprosed GEM workgroup cluster.

3. Common UNIX User Environment / I. Gaines.

* We need a more uniform UNIX user environment.

* There is a HEPIX document on this.but it is CERN minded.

* US version discussed at a meeting at CEBAF. The US moves slower than the Europeans. Three options:

    1) Work with them?
    2) Ignore them?
    3) Follow them?

* What is the SSCL policy?

* First step: Pull work done sofar together.

* Next step: **Start thinking** about what we really want.

4. An object oriented analysis framework/ F. Rouse.

Seminar at the SSCL of a few weeks ago.

5. Application servers / Mark Selover.

Setup of disks, file-servers etc. Refer to the minutes of the GEM local computing meeting of a few weeks ago.

6. Process control prototype / Glenn Kubena.

- Short term objectives:

  * Experiment with a C++ implementation of process control.

  * Evolve concept in close interaction with potential user.

  * Integrate with other SDC prototyping efforts.

- Long term objectives:

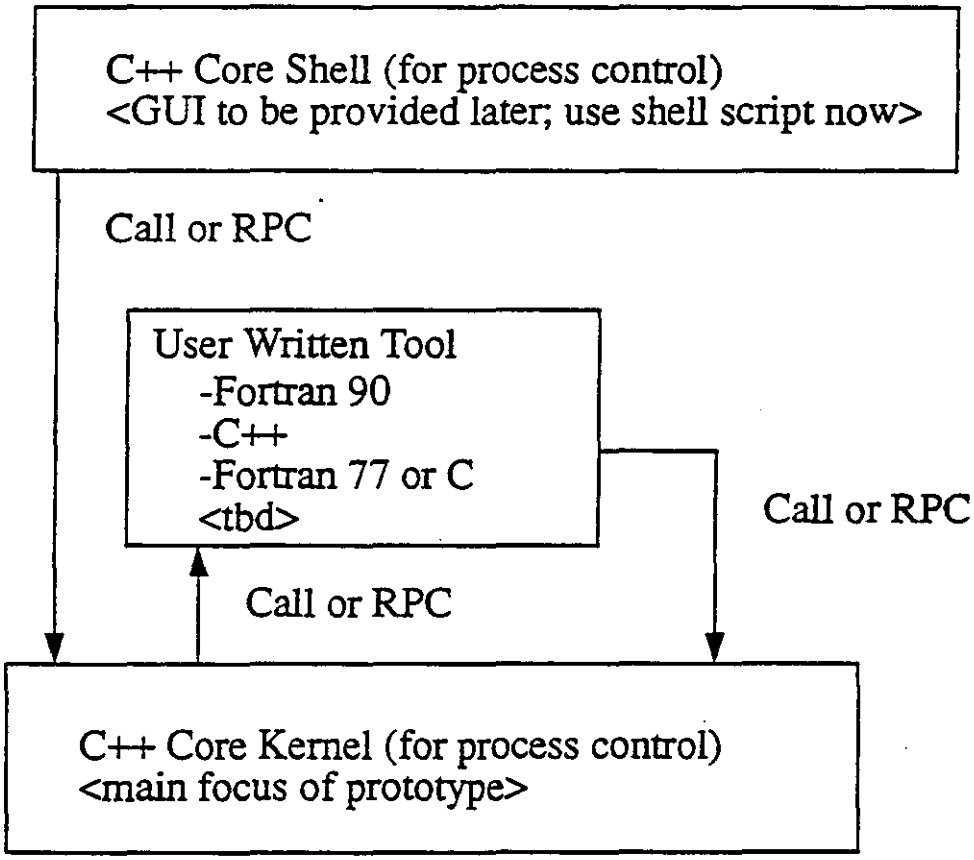
  * Start using RPC's.

  * Develop programming style and examples, "style guide for the core s/w".

  * Add GUI.

- Transparancies showed a first prototype, with pull-down menu's to set thing up, run them etc. Implements control lines, queueing of messages, etc.

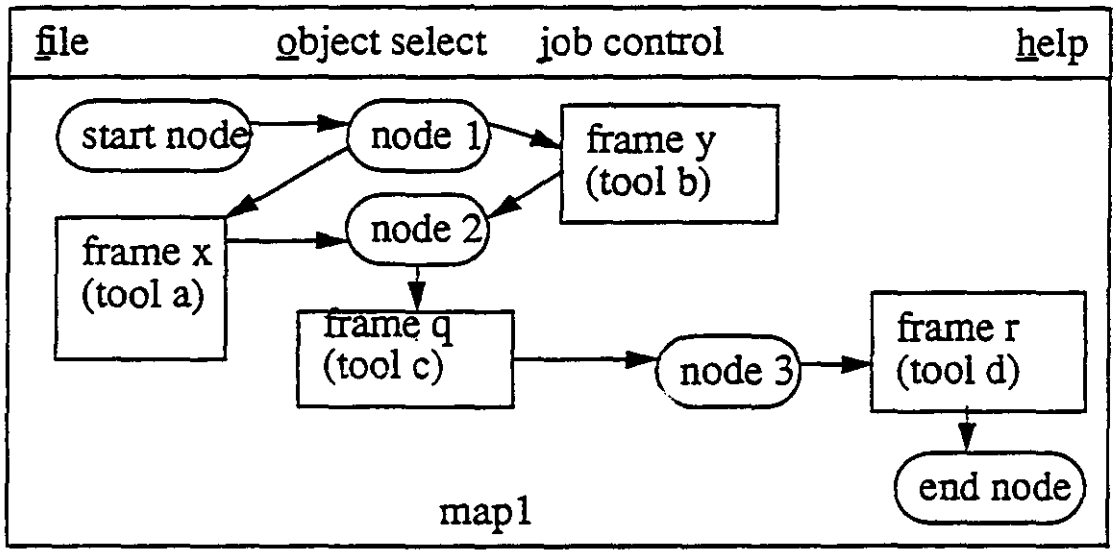
Process Control Prototype General Assumptions



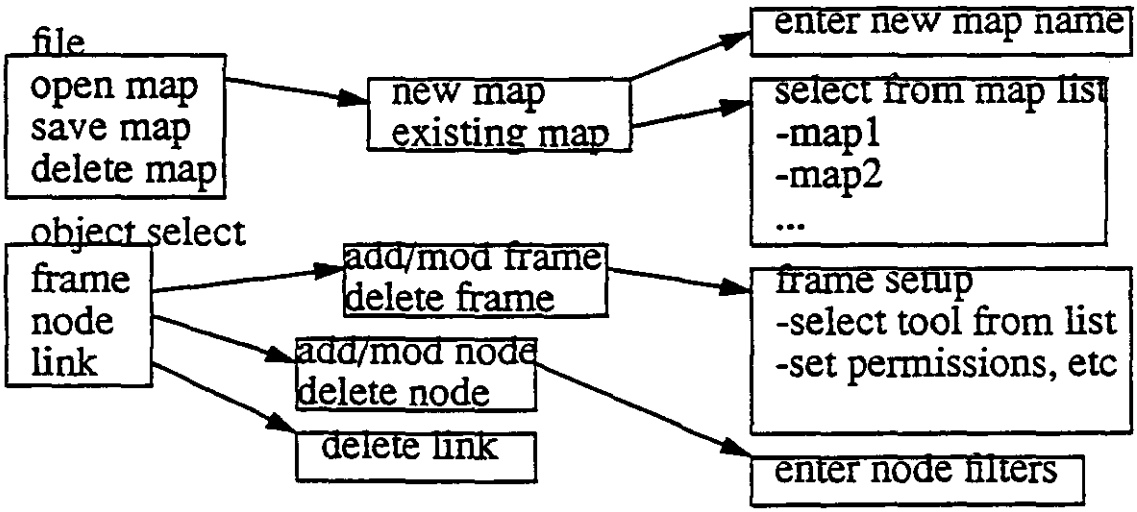
Glenn Kubena (713) 282-7535, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSD High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

Candidate Process Control GUI

Map Setup & Job Control Window



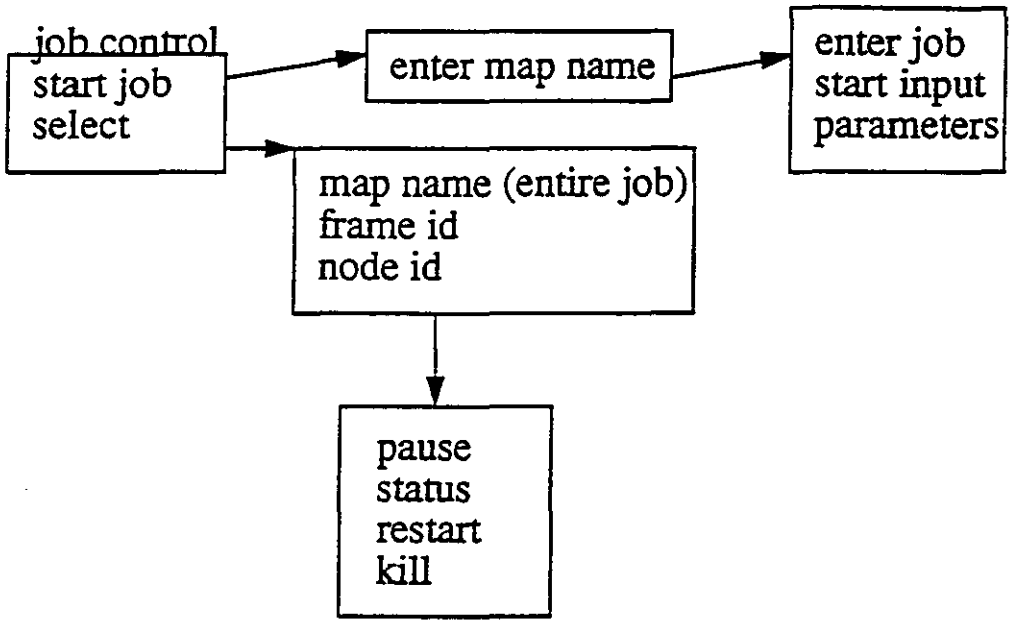
Pull Downs (etc.)



Use mouse for linking frames and nodes; and for selecting objects

Glenn Kubena (713) 282-7835, INTERNET kubena@houvmsc.vnet.ibm.com, FAX (713) 282-7409
IBM FSD High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

Pull Downs (continued)



Note: Map can be modified after job is started; e.g. may want to Pause a node and add a link to a new frame(tool) which does additional analysis

Glenn Kubena (713) 282-7835, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
EM PSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

7. Core Software / Soren Frederiksen *et al.*

*** HENCE/PVM from Oak Ridge.**

- Test bed for ideas, not a prototype or demo of the real system.
- This study looked into data transfer, message passing, client/server models, parallel processing.
- Parallel processing: shared memory or message passing. The latter was used here.
- Process control through passing of event id's in the network. A data server provides the data to the processing units.

*** Demo of HENCE/PVM, SDC version.**

- System control by a master control process.
- The user draws a analysis programme on a workstation, then runs it by pressing a start button on a GUI. The programme consists of ``system nodes" (start, stop, spooler, replicator....) and ``user nodes" (Fortran or C code for analysis) and channels.
- **The demo looked very impressive.** Development will continue, the first step is a better data-server. For a demo at the SSCL: contact Ivan Chow.

8. Next meeting: June 30 - July 2 @ SSCL.

SDC Process Control GUI Prototype

IBM FSC High Performance Solutions Integration

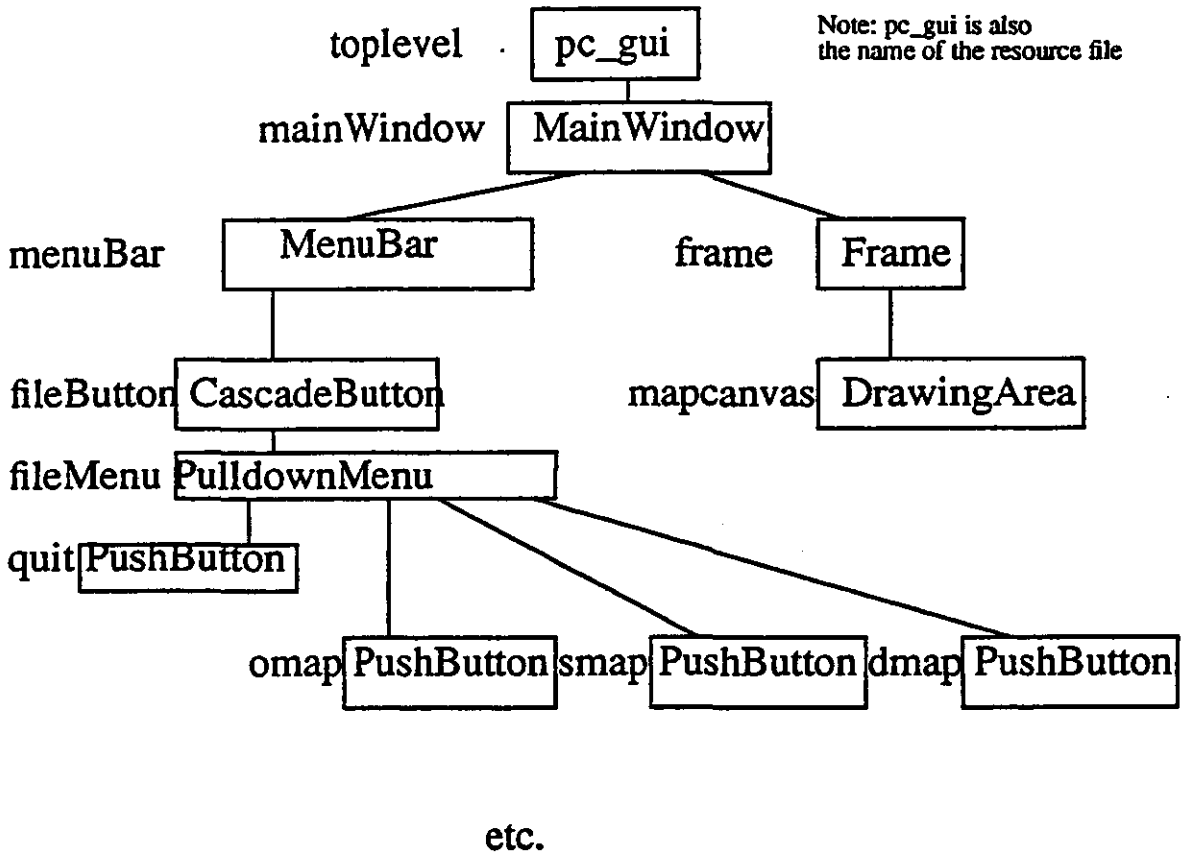
Glenn Kubena
Alicia Harvey
Larry Roberts
In Collaboration with David Adams, Rice University

May 17, 1993

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

Process Control GUI Motif Widget Instance Hierarchy

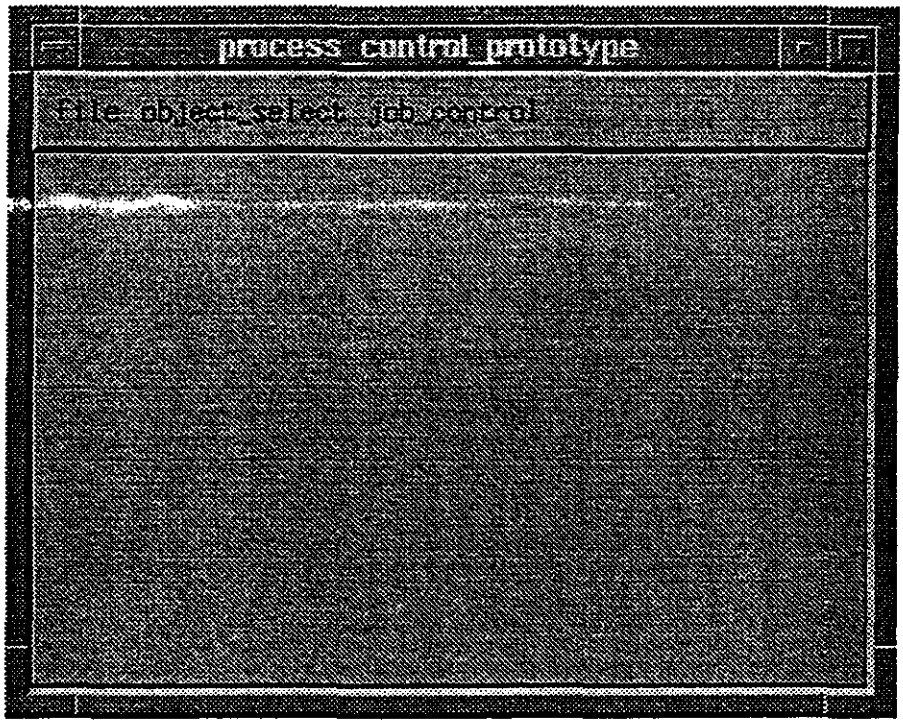
Load Module Name for X Client is process_control_prototype
Source Code for X Client is pc_xclient.c



Note: Widget Class Names are in boxes, Widget internal identifiers are next to box

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

Process Control GUI X/Motif Window



Glenn Kubena (713) 282-7635, INTERNET kubena@hcuvm:scc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

pc_xclient.c

```
/* Name: pc_xclient.c */
/* Description: Process Control user interface prototype */
/* Date: May 14, 1993 */
/* Vers 1 */


#include <X11/Intrinsic.h> /* intrinsics definitions*/
#include <X11/StringDefs.h> /* X string definitions*/

#include <Xm/Xm.h> /* Motif definitions */
/* #include <Xm/RepType.h> */ /* Motif definitions */

#include <Xm/MainW.h> /*Motif main window*/
#include <Xm/BulletinB.h> /*Motif Bulletin Board widget */
#include <Xm/TextF.h> /*Motif Text Widget*/
#include <Xm/Frame.h> /*Motif Frame widget *
#include <Xm/Label.h> /*Motif Label widget */
#include <Xm/PushB.h> /*Motif Push Button widget */
#include <Xm/CascadeB.h> /*Motif Cascade Button Widget */
#include <Xm/RowColumn.h> /*Motif Row Col Widget */
#include <Xm/DrawingA.h> /*Motif Drawing Area Widget */
#include <Xm/MessageB.h> /*Motif Message Box Widget (for help)*/
```


Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

```
main (argc, argv)
 int argc; /*command line arg */
 char *argv[]; /*command line arg */


{
/* 1st create the root of the widget tree or the "shell" widget*/
/* This function calls other intrinsics functions to establish the
connection to the X Server and init the intrinsics' data strucs */
XtAppContext app_context;
/*widget names*/
Widget toplevel, mainWindow, menuBar, frame;
Widget fileMenu, fileButton, omap, smap, dmap, quit ;
Widget jcButton, oselectButton;
Widget mapcanvas ;
/* XtSetLanguageProc(NULL, (XtLanguageProc)NULL, NULL); */
toplevel = XtAppInitialize(&app_context,
"pc_gui", /*application class name*/
/*also name of resource file*/
NULL,0, /*command line option list*/
&argc,argv, /*command line args*/
NULL, /*app defaults*/
NULL,0);

/* Create a main window widget */
mainWindow = XtVaCreateManagedWidget
```


Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

```
("hello", xmMainWindowWidgetClass, toplevel, NULL );

/* register converter for setting tearoff menus from resource files*/
/* XmRepTypeInstallTearOffModelConverter(); */

/* create a menubar along top inside of main window */
menuBar = XmCreateMenuBar(
mainWindow, /*parent widget*/
"menuBar", /*widget name*/
NULL, 0 ) ;
XtManageChild (menuBar);
frame = XtVaCreateManagedWidget(
"frame", xmFrameWidgetClass, mainWindow, NULL);
/* Set MainWindow areas */
XmMainWindowSetAreas (mainWindow, menuBar, NULL, NULL, NULL,
frame);


/* Create the File button in the menu bar*/
fileButton = XtVaCreateManagedWidget
("file",
xmCascadeButtonWidgetClass,
menuBar,
NULL) ;

/* Create menu (really a Shell widget and a RowColumn widget combo */
fileMenu = XmCreatePulldownMenu
```


Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

```
(menuBar, /*parent widget*/
"fileMenu", /*widget name*/
NULL, 0);
/* Notice that fileMenu is intentionally NOT managed here */

/* Create Button that opens a map */
omap = XtVaCreateManagedWidget (
"open_map", /*widget name*/
xmPushButtonWidgetClass, /*widget class*/
fileMenu, /*parent widget*/
NULL);

/* Create Button that saves a map */
smap = XtVaCreateManagedWidget (
"save_map", /*widget name*/
xmPushButtonWidgetClass, /*widget class*/
fileMenu, /*parent widget*/
NULL);

/* Create Button that deletes a map */
dmap = XtVaCreateManagedWidget (
"delete_map", /*widget name*/
xmPushButtonWidgetClass, /*widget class*/
fileMenu, /*parent widget*/
NULL);
```


Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

```
/* Create Button that exits application */
quit = XtVaCreateManagedWidget (
"quit", /*widget name*/
xmPushButtonWidgetClass, /*widget class*/
fileMenu, /*parent widget*/
NULL);
/* Specify which menu the fileButton will pop up */
XtVaSetValues(fileButton, XmNsubMenuId, fileMenu, NULL);
/* Arrange for quit button to call function that exits */
/* XtAddCallback (quit, XmNactivateCallback,Quit,0); */


/* Create the object select button in the menu bar*/
oselectButton = XtVaCreateManagedWidget
("object_select",
xmCascadeButtonWidgetClass,
menuBar,
NULL) ;
/* Create the job control button in the menu bar*/
jcButton = XtVaCreateManagedWidget
("job_control",
xmCascadeButtonWidgetClass,
menuBar,
NULL) ;
```


Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

```
/* Create a Canvas Motif widget, as a child of frame */
 mapcanvas = XtVaCreateManagedWidget
 ("mapcanvas", xmDrawingAreaWidgetClass, frame,
 /* XtNwidth, 600, XtNheight, 400, */
 XmNx, 10, XmNy, 50, NULL) ; /*set xy offsets*



/* Create a CascadeButton as attachment point for pulldown */
/* cascade1 = XtVaCreateManagedWidget
/* ("file", xmCascadeButtonWidgetClass, rc1,
/* XmNx, 20, XmNy, 20, NULL ) ; /*set xy offsets*/

/* Open the toplevel window and display it via realize */
XtRealizeWidget (toplevel); /*note:this realizes all widgets*/

/* Wait for a user event to occur */
XtAppMainLoop (app_context);
}
```


Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

pc_gui (Resource File) in users home directory

```
! Name: pc_gui
! Description : Resource File for Process Control GUI
! Author : Glenn Kubena
! Date : May 15, 1993
! Set initial size of application window
*XmMainWindow.width: 400
! Frame may be bigger than application, thus add scrollbars
*XmFrame.width: 800
*XmFrame.height: 800
! Make MainWindow display scrollbars and use them to pan around in the frame
*XmMainWindow.scrollingPolicy: XmAUTOMATIC
! Set size of canvas drawing area for the PC Map
*XmDrawingArea.height: 1000
*XmDrawingArea.width: 1000
```


Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

SDC Process Control Prototype

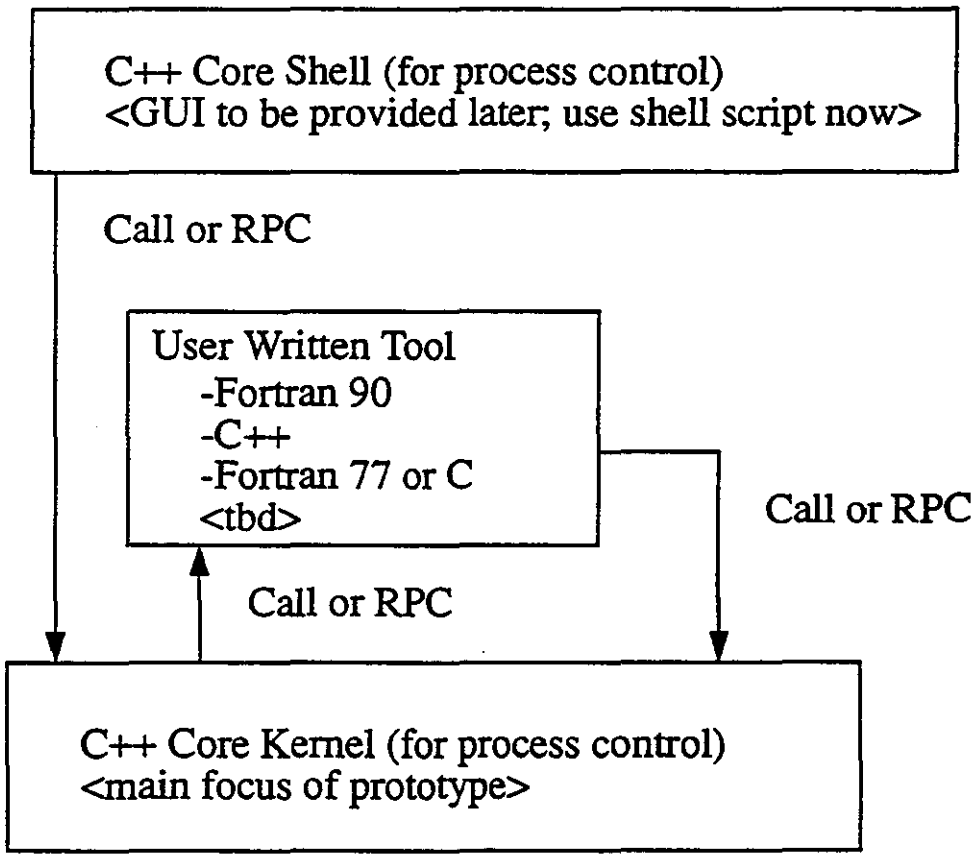
IBM FSC High Performance Solutions Integration

Glenn Kubena
Alicia Harvey
Larry Roberts
In Collaboration with David Adams, Rice University

May 17, 1993

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

Process Control Prototype General Assumptions



Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

Process Control Prototype Objectives

Short Term

- Experiment with one C++ implementation of Process Control
- Document detailed Process Control design[1] and issues as prototype evolves
- Evolve Process Control operational concept and GUI with Users
- Coordinate and integrate Process Control prototype with other SDC prototyping efforts (e.g. Communications, User Interface, ...)

Long Term

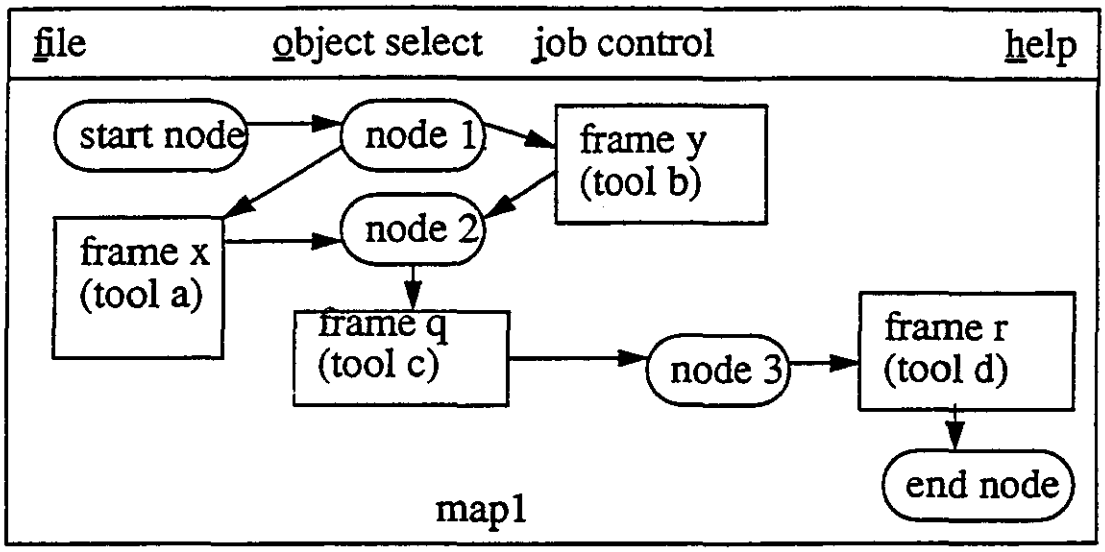
- Replace calls with RPCs, where appropriate
- Develop programming style and examples ==> potential style guide for other components of the core
- Add GUI

1. Rumbaugh notation on new release of ParadigmPlus (May) and on STP (Fall) with output to Framemaker ==> need common design documentation approach for all of Core

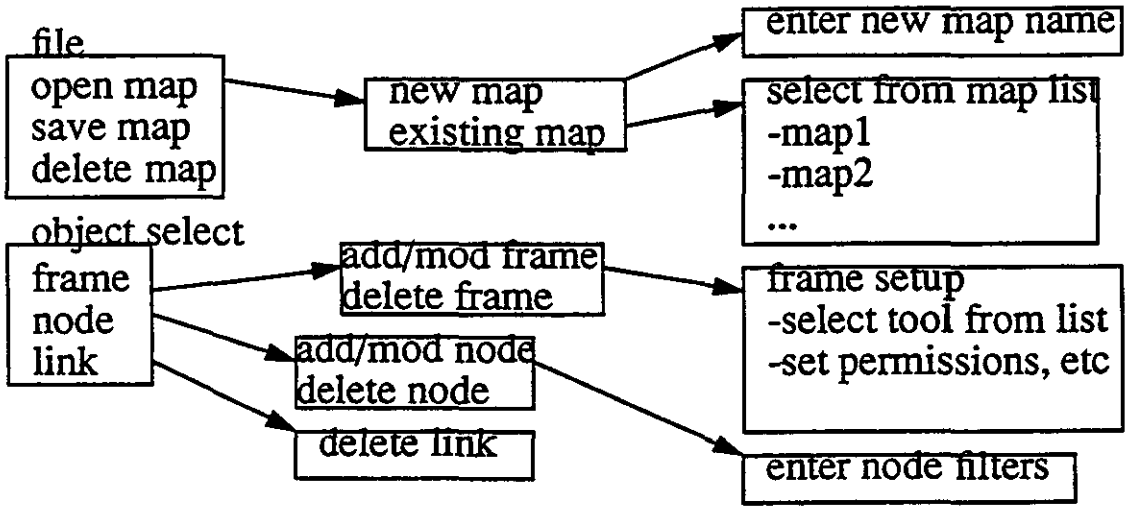
Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

Candidate Process Control GUI

Map Setup & Job Control Window



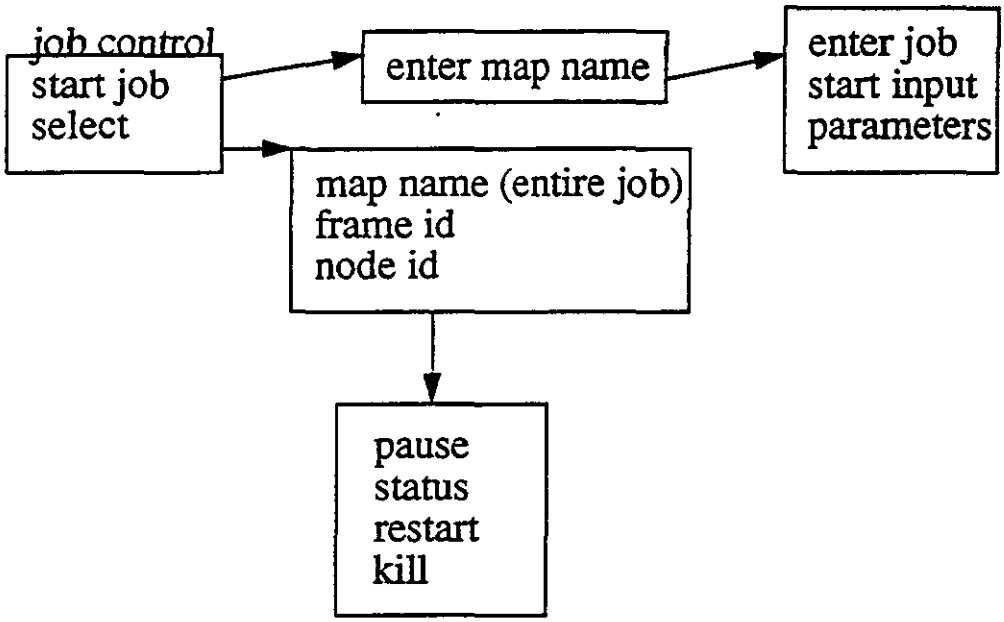
Pull Downs (etc.)



Use mouse for linking frames and nodes; and for selecting objects

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

Pull Downs (continued)



Note: Map can be modified after job is started; e.g. may want to Pause a node and add a link to a new frame(tool) which does additional analysis

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

Basic Operational Scenario

At "Setup Time"

1. Pull down file ==>Open a map (new or existing)
2. Pull down object select ==>

   Select a frame type and assign a tool to the frame from tool list (based on permissions which may be overridden based on user)

   Select a node type and set node filters
3. Using the mouse, link frames to nodes and nodes to links <PC returns an error for invalid links>
4. Pull down file ==> Save map

At "Job Initialization & Start Time"

1. Pull down job control ==> Select start job, enter map name and any job input parameters
2. <PC creates the Unix processes for all the frames, nodes and links, preloads selected tools (based on tool descriptor) and displays job map to GUI>

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

While the Job is Running

1. <PC updates the Job Map on the GUI as job progresses (e.g. using colors)
2. User may pull down job control ==> to pause, status, restart or kill a frame, node or the entire job
3. User may reconfigure the map while the job is running, for example,

   by pulling down job control ==> pausing a node
   pulling down object select ==> adding a new frame/tool and linking the paused node to that new frame/tool
   job control ==> restarting the node

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

A Sample Script

PC Shell : Welcome to the Process Control Shell

PC Shell : Select a Pull Down Menu

file

PC Shell : Enter open, save or delete

open

PC Shell : Select new or existing map

new

PC Shell : Enter new map name

my_analysis_map

PC Kernel : Opening map my_analysis_map

etc.

****This is ugly to use and requires code to emulate the GUI***

****Therefore avoid if possible ****

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

Operational Assumptions & Design Decisions

1. Assume one and only one tool per frame
2. Frame functions include:

   Handles permissions for tools (which tool can run, what files it can access) <==based on frame type

   Keeps tool status
   -running
   -# events handled
   -etc.

   Handles all communications to the tool

   Forwards output messages to downstream queue (link)

   Handles errors
3. Each frame and node is a separate Unix process and can run distributed (to allow for max job throughput and flexibility to run on different hardware architectures; also to allow for flexible job control, e.g., pause/status of individual frames and nodes)
4. Assume one and only one input queue per frame to support a single entry point to a tool.
5. A node can have multiple input queues(links) and send messages to multiple output queues(links).
6. A node looks at filter bits in the "control signal" and outputs errors and status based on the bits. Filter bits are set by upstream frame.
7. A "control signal" consists of:
   -Event ID
   -Filter bits (success/failure, ...)

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

Asynchronous Control and Queueing on Link

Basic Assumption : PC will use Core Comm. Services (e.g. RPCs) and basic Unix OS Services (e.g. process scheduling, IPC message queueing, ...) where appropriate

Also, the messages queued are of type, control_signal
And, the message queues are FIFO (first in/first out)

Message queued for input to a frame or node

1. A node or frame is in a wait state until the input queue (link) is posted with a message (by a separate "link" process)
2. If frame ==> the frame then wakes up and calls the tool. While the tool is executing additional messages can be added to the queue asynchronously (by the separate "link" process). When the tool finishes executing it returns to the frame, which again waits for the next message on the queue to be posted (or continues immediately if a message is already posted to the queue)
3. If node ==> the queueing works the same except that, instead of calling a tool the node checks the filter bits. Also, since the node can have multiple input queues and may have logic to coordinate these (e.g. by event number) for subsequent output
4. Special Case : Queue Full ==> Rather that simply purging the newest message, the "link" process can send a message to the upstream node or frame to wait for N time units and resend

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

Message sent to a queue:

1. A node or frame simply sends a control signal message to the proper queue (based on queue ID) ==> This is most likely done by RPC since the receiving link/node or link/frame may be distributed
2. The "link" process receives the message and adds it to the queue (or if the queue is full, notified the upstream sender)

Note: Since the "link" process and node or frame process are so tightly coupled it probably makes no since to distribute them and simply use Unix IPC message queueing facility

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

Process Control Detailed Design: Proposed Classes

pc_job_map

- map_list
- map_name
- tool_list
- tool_name
- frame_id
- node_no
- job_start_parms
- status

- pc_maintain_map_list
- pc_maintain_tool_list
- pc_setup_map
- pc_start_job
- pc_monitor_job

pc_node

- node_id
- node_type
- input_link_ids
- output_link_ids
- node_status
- node_filters

- pc_setup_node
- pc_initialize_node
- pc_dequeue_input_signal
- pc_filter_messages
- pc_enqueue_output_signal
- pc_supply_node_status

pc_frame

- frame_id
- frame_type
- input_link_id
- output_link_id
- control_signal
- frame_status
- tool_name

- pc_setup_frame
- pc_initialize_frame_tool
- pc_dequeue_input_signal
- pc_enqueue_output_signal
- pc_supply_status

Glenn Kubena (713) 282-7835, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

Persistent Data Classes Used by Process Control

map_list
map_names

tool_list
tool_names

tool_descriptor
tool_name tool_type data_base_requests

map_descriptor
map_name frame_id_list node_id_list

frame_descriptor
map_name.frame_id frame_type input_link_id output_link_id tool_name

node_descriptor
map_name.node_id node_type input_link_id output_link_id

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

Programming Style for Prototype

- Process control classes are defined in a separate header file
- Process control methods (member functions) are defined in a separate .C file for separate compilation
- Process control shell is a separate file which instantiates objects of PC classes and calls PC member functions (later replaced or replicated in part by GUI)
- Later, PC shell and PC core may be made separate remote processes

```
// process_cntl_classes.h
class pc_job_map {
public
pc_do_map_list (char* x);
}
class pc_frame { ... }
class pc_node { ... }
```

```
// process_control_shell.C

#include "process_cntl_classes.h"

main ( )
{
// shell code with process control
// classes instantiations and calls
  pc_job_map mapx(100,500,16);
  mapx.pc_do_map_list(map);
  }
```

```
// process_cntl_methods.C
#include "process_cntl_classes.h"

// pc classes member functions definitions
pc_job_map::pc_do_map_list ( ... ) { ... }
// etc.
```


Glenn Kubena (713) 282-7535, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

Appendix 1 - Code Listings

Note : Ver. 1 of the code may be found on the PSDF at _____.

Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

process_cntl_classes.h

```
// Name: process_cntl_classes.h
// Description : Header file for process control class definitons
// Author : Glenn Kubena
// Date : 4/22/93, Vers.1
//
//*****************job_map class*****************
class pc_job_map
{
 public: //public declarations, visable to other classes

 //member function prototype declarations follow

 //get list of existing map names
 char* pc_maintain_map_list (char* map_list) ;
 //note : char* is to pass an error message back to the GUI
 //open or save a map
 char* pc_maintain_map_list (char* map_name, int opt) ;

 //get list of existing tools
 char* pc_maitain_tool_list (char* tool_list) ;
 //select a tool
 char* pc_maintain_tool_list (char* tool_list, int opt) ;
```


Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

```
//select a frame for the tool
char* pc_setup_map (char* frame_id, char* tool_name);
//select a node
char* pc_setup_map (int node);
//link or unlink a frame to a node
char* pc_setup_map (char* map_name, char* tool_name, int node_no, int opt);
//opt 1 = link frame to node, opt 2 = link node to frame, opt 3 = unlink

//start job
struct jsp { //rough idea of job start parameters
char file[16] ; //e.g. input file for events
int option ;
} ;
char* pc_start_job ( char* map_name, jsp job_start_parms ) ;

//monitor & control job
//select a node or frame
void pc_monitor_job ( char* map_name, int node_no, char* tool_name);
//pause, kill, status or restart a node or frame
struct status { //rough idea of job, frame or node status
char mode[16] ; //waiting, running, ...
int cpu; //cpu utilization
int mem; //memory utilization
int enque; //number of messages waiting in queue
int eventno; //event number being processed};
void pc_monitor_job ( char* map_name, int node_no, char* tool_name, int opt,
double stats[] );
```


Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration 3700 Bay Area Blvd., Houston, Texas 77058

```
//pause, kill, status or restart entire job
void pc_monitor_job ( char* map_name, int opt) ;
//*****end of public declarations

private: //visible only to member functions
int maxmaps ; // max number of maps
int maxtools ; // max number of tools
int maxname ; // max name size
char* pri_map_list;
char* pri_tool_list;
//*****end of private declarations

public: // more public declarations
//constructor for this class; allocates space
pc_job_map (int i, int j, int k)
{ //inline definition for less overhead
maxmaps = i; maxtools = k; maxname = j; // user can set these
pri_map_list = new char [i*j] ;
pri_tool_list = new char [k*j] ;
}
pc_job_map ( ) //overloaded constructor for defaults
{
maxmaps = 100; maxtools = 500 ; maxname = 16;
pri_map_list = new char[maxmaps*maxname] ;
pri_tool_list = new char[maxtools*maxname] ;
}
```


Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

```
//destructor for this class; frees space
~pc_job_map () { delete [] pri_map_list; delete [] pri_tool_list; }

}; //************end of map class definition***
```


Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

process_cntl_methods.C

```
//Name : process_cntl_methods.C
//Description : Member function (method) definitions for process control
//Author : Glenn Kubena
//Date : 5/10/93, Vers. 1
//
#include <iostream.h>
#include "process_cntl_classes.h"


// Definition of pc_maintain_map_list
char* pc_job_map::pc_maintain_map_list (char* mn, int x)
 {
 cout << "PC Kernel : Opening map " << mn << endl;
 // rest of code to be supplied


 return "PC Kernel : Successful Completion" ;
 }
```


Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

process_control_shell.C

```
// Name: process_control_shell.C
// Description: Handles process control script (and later GUI)
// Author: Glenn Kubena
// Date: May 7, 1993
//
# include <iostream.h>
# include "process_cntl_classes.h"
char pdname[16];
char osd[8];
char noe[8];
char map[16];

main ()
{
 cout << "PC Shell : Welcome to the Process Control Shell" << endl;
 // define an instance mapx, of class pc_job_map
 pc_job_map mapx(100,500,16); //and set the defaults
 cout << "PC Shell : Select a Pull Down Menu" << endl;
 cin >> pdname ;
 if (pdname[0] == 'f' ) //file
 {
 cout << "PC Shell : Enter open, save or delete" << endl;
 cin >> osd ;
 }
```


Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058

```
if (osd[0] == 'o') //open
{cout << "PC Shell : Select new or existing map" << endl;
cin >> noe ;}
if (noe[0] == 'n' ) //new
{cout << "PC Shell : Enter new map name" << endl;
cin >> map;
mapx.pc_maintain_map_list(map, 1); //option 1 to open map
}

} //end of main
```


Glenn Kubena (713) 282-7635, INTERNET kubena@houvmscc.vnet.ibm.com, FAX (713) 282-7439
IBM FSC High Performance Solutions Integration, 3700 Bay Area Blvd., Houston, Texas 77058