GUI 2 위젯의 개수가 많아지면...

핵심 프로그램 예시

거의 모든 IDL 프로그래밍 작업은 핵심 프로그램(자료처리, 계산, 가시화 등)에 치중하게 됩니다. 이번 예제에서는, 다음과 같이 TITLE 키워드를 통해 제목을 지정해 줄 수 있는 초간단 등치선도 생성 루틴이 핵심 프로그램이라고 가정하고(현실에서 그럴 리는 없겠지만) 진행하겠습니다. 이 프로그램이 널리 사용될 수 있도록 하기 위해 GUI를 입히기로 결심한 상황입니다.

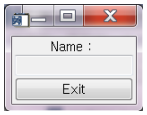
```
pro contour_some_data, title=title
  data=randomu(seed, 100, 100)
  for i=0, 9 do data=smooth(data, 9, /EDGE_TRUNCATE)
  contour, data, XSTYLE=1, YSTYLE=1, title=title
end
```

WIDGET_BASE의 COLUMN과 ROW 키워드

GUI의 배치에 대해 집중적으로 다룰 기회가 있겠지만, GUI 배치의 핵심 테크닉은 WIDGET_BASE 함수에서 COLUMN이나 ROW 키워드를 적절히 사용하는 것입니다. 말 그대로 COLUMN은 판에 몇 단(column)으로 GUI를 배치할 것인지, ROW는 몇 줄(row)로 배치할 것인지 지정하는 것입니다. 많은 경우 이 키워드에는 1이 들어갑니다. 그래서 /COLUMN이나 /ROW로 쓰이는 경우가 많지요.

```
pro contour_gui
tlb=widget_base(TITLE='GUI for Contour', /COLUMN)
                    ;1단(Column)구성의 판
nameLabel=widget_label(tlb, Value='Name : ')
nameText=widget_text(tlb, xsize=15)
button=widget_button(tlb, value='Exit')
widget_control, tlb, /REALIZE
end
```

그림과 같이 모두 TLB(하위 요소 모두 세로 1단 배치)에 소속되어 있으므로, Label, Text, Button이 모두 세로로 늘어서 있습니다. 그러나, "Name : "이라고 나오고 그 아래 줄에서 이름을 입력하는 것보다는 오른쪽에서 이름을 입력하는 것이 자연스럽겠지요. Widget_Base 하위에 Widget_Base도 올 수 있다는 점을 이용하여 이 문제를 해결합니다.

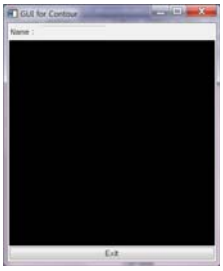


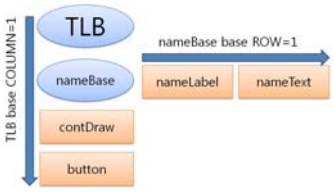

```
tlb=widget_base(TITLE='GUI for Contour', /COLUMN)
nameBase=widget_base(tlb, /ROW)
nameLabel=widget_label(nameBase, Value='Name : ')
nameText=widget_text(nameBase, xsize=15)
button=widget_button(tlb, value='Exit')
```

위 코드에서 수정된 부분을 보면, TLB 아래에 nameBase를 추가로 두어 nameLabel과 nameText는 nameBase의 하위에 소속되도록 조치한 것을 확인할 수 있습니다. 이제 그림 창(Widget_Draw)을 하나 더 추가하는 작업같은 것은 매우 간단하게 보일 것입니다.

```
nameText=widget_text(nameBase, xsize=15)
contDraw=widget_draw(tlb, xsize=400, ysize=400)
button=widget_button(tlb, value='Exit')
```

이런 식으로 GUI의 배치는 비교적 쉽게 확장이 가능합니다. 코딩을 통해 GUI를 구성하는 것이 Visual Studio나 Delphi 등의 개발 환경에 익숙한 분들에게는 어색할 수 있으나 관리의 편리성에서는 더 뛰어난 면도 있습니다.





예제 contour_gui의 위젯 배치 구조

Value와 UValue

모든 GUI 요소는 각각 Value와 UValue를 가지고 있습니다(Widget_Base는 Value가 없습니다).

Value는 GUI요소가 태생적으로 가져야 하는 값입니다. 예를 들어 Widget_Draw(그림창)의 경우에는 그림창 번호가 꼭 있어야 하고, Widget_Button(버튼)의 경우에는 버튼 위에 올라갈 문자열이나 그림 파일(그림버튼)의 경로가 반드시 있어야 합니다, Widget_Label(레이블)의 경우라면 출력될 문자열이 Value가 되겠지요. 그래서 Value는 데이터의 타입을 사용자가 마음대로 정할 수 없습니다. Widget_Draw에서 Value는 그림창의 번호가 들어갈 자리인데, 문자열이나 구조체를 넣을 수는 없는 것입니다.

이에 반해 UValue는 사용자가 어떤 값이든 자유롭게 넣을 수 있는 공간입니다(그래서 User Value입니다). 프로그램을 만들다 필요한 내용을 마음대로 저장하고 꺼내고 변경할 수 있는데, 문자열은 물론이고, **심지어 구조체도 가능**하므로 원하는 어떤 규모의 데이터라도 넣을 수 있습니다. GUI 요소에 UValue를 추가해 봅시다.

```
nameText=widget_text(nameBase, xsize=15, UVALUE='nametext')
contDraw=widget_draw(tlb, xsize=400,ysize=400, UVALUE='drawwin')
button=widget_button(tlb, value='Exit', UVALUE='exitbutton')
```

Value와 UValue는 처음에는 예제 코드와 같이

UValue= , Value= 로 설정하지만 프로그램이 진행되면서부터는 Widget_Control의 키워드 GET_VALUE=, GET_UVALUE=로 읽어오고 SET_VALUE=, SET_UVALUE=으로 변경할 수 있습니다.

활성 / 비활성

nameText는 현재 값을 입력받지 못하는 상태입니다. EDITABLE=1로 설정할 때 입력을 받을 수 있게 됩니다. 예제에서는 항상 입력 가능한 모드로 사용하면 되지만 프로그램에 따라, 일시적으로 사용자의 입력을 금지 / 허용 할 수 있는 세밀한 설정을 할 수도 있습니다.

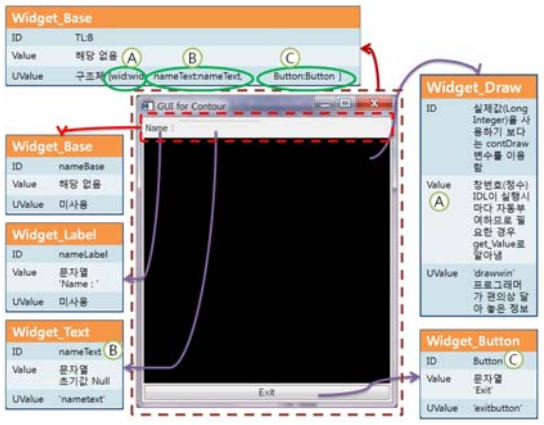
```
nameText=widget_text(nameBase, xsize=15, UVALUE='nametext', $
                     /EDITABLE)
```

최상위 판(TLB)의 UValue

이벤트 구조체는 이벤트를 발생시킨 위젯의 ID와 최상위 판의 ID가 항상 전달됩니다. 그러므로 최상위 판의 정보는 어떤 이벤트에서라도 쉽게 접근할 수 있지요. 이 때문에 최상위 판의 UValue에 프로그램에 필요한 거의 모든 정보를 모두 집어 넣어 두면 편리합니다. **여러 종류의 정보를 변수 하나에 넣어 두어야 하니 구조체가 적당할 것**입니다. GUI 선언부(메인 프로그램)는 보통 다음과 같이 완성됩니다.

```
widget_control, tlb, /REALIZE
widget_control, contDraw, GET_VALUE=wid    ;창번호 획득
state={wid : wid, nameText : nameText, button : button}
widget_control, tlb, SET_UValue=state  ;최상위판 UValue 설정
xmanager, 'contour_gui', tlb
end
```

창번호를 알아 내고, 창번호(wid)에 nameText의 ID와 button의 ID를 모두 구조체로 묶어 최상위판의 UValue로 넣었습니다. 실제 개발에서 가장 많이 사용될 UValue는 TLB의 UValue입니다.



GUI의 ID, Value, UValue

누가 이벤트를 일으킨 것일까?

event 구조체에는 이벤트를 발생한 위젯의 ID가 포함되어 있습니다. 문제는 이 ID 자체로는 어떤 위젯인지 알 수가 없다는 것인데, ID는 단순한 숫자이고, 프로그램이 실행될 때마다 IDL이 자동으로 부여하는, 프로그래밍 단계에서는 미정의 값이기 때문입니다. 그래서, 앞 절에서 이 ID들 중 필요한 것들을 최상위 판의 UValue 구조체에 기록해 두었지요. 이 값을 뽑아내어 event.ID와 비교하면, event ID가 어떤 위젯의 ID인지 알아낼 수 있습니다. 이 방법으로 이벤트 처리 루틴을 다음과 같이 만들 수 있습니다.

```
pro contour_gui_event, ev
  widget_control, ev.top, GET_UValue=state
  case ev.id of
   state.button : widget_control, ev.top, /DESTROY
   state.nameText : begin
    wset, state.wid
    widget_control, state.nameText, GET_Value=name
    contour_some_data, title=name
   end
   else :
  endcase
end
```

GUI 정의부에서 넣어 둔, 최상위판의 UValue를 뽑아내어 그림창 번호와 nameText, Button의 ID를 알 수 있으므로, 이벤트를 발생시킨 위젯이 Button일 경우와 nameText일 경우를 구분할 수 있고, 어느 창에 그림을 그릴지도 결정할 수 있게 됩니다(wset, state.wid).
예제에는, 각 위젯에 UValue도 심어 놓았으므로 이벤트 처리 루틴은 다음과 같이 만들 수도 있습니다. 작동 내용은 똑같습니다.

```
pro contour_gui_event, ev
  widget_control, ev.top, GET_UValue=state
  widget_control, ev.id, GET_UValue=uval
  case uval of
    'exitbutton' : widget_control, ev.top, /DESTROY
    'nametext' : begin
      widget_control, ev.id, GET_Value=name
      wset, state.wid
      contour_some_data, title=name
    end
  else :
endcase
end
```

포인터의 필요성

이번 예제는 프로그램이 작아서 괜찮지만, 규모가 조금 커지면 TLB의 UValue는 덩치가 꽤 커집니다. 이를 이벤트 발생 때마다 매번 꺼내고 넣으려면 프로그램의 실행이 둔해질 수 있지요. 그래서 최상위 판의 UValue는 포인터로 넣어 두는 경우가 많습니다.