Acceso a los datos del Esqueleto

```
// Definimos la pose de calibración. "Psi" es una pose por defecto que
// se refiere a colocar los brazos en ángulo recto, codos a la altura
// del pecho aproximadamente y abiertos, y manos hacia arriba, de
// forma que el usuario, visto desde la cámara, simula la forma de la
// letra griega Psi.
#define POSE_TO_USE "Psi"

// Declaramos una variable o nodo de producción global de tipo
generador de usuario, que será el que utilicemos para detectar el
esqueleto del usuario.
xn::UserGenerator g_UserGenerator;

// Declaramos una función callback, que será llamada cuando se detecte
// un nuevo usuario en la escena
void XN_CALLBACK_TYPE
User_NewUser(xn::UserGenerator& generator,
             XnUserID nId, void* pCookie)
{
  // Cada vez que se detecta un nuevo usuario, se inicia un proceso de
  // detección de la pose de calibración.
  printf("New User: %d\n", nId);

g_UserGenerator.GetPoseDetectionCap().StartPoseDetection(POSE_TO_USE,
                                                         nId);
}

// Igualmente, se declara una función callback para el caso en que un
// usuario previamente detectado ha desaparecido o no ha sido posible
// continuar con el seguimiento/identificación del mismo (esta
// función está vacía porque no deseamos en nuestra aplicación
// realizar acción alguna cuando perdamos un usuario, pero es
// necesaria declararla en el interfaz de OpenNI).
void XN_CALLBACK_TYPE
User_LostUser(xn::UserGenerator& generator, XnUserID nId,
              void* pCookie)
{}

// También definimos una función callback que será llamada cada vez
// que se detecte un usuario con la pose correcta.
void XN_CALLBACK_TYPE
Pose_Detected(xn::PoseDetectionCapability&    pose,    const    XnChar*
strPose,
              XnUserID nId, void* pCookie)
{
  // Una vez hemos detectado una pose sobre un usuario que estaba
  // previamente identificado, dejamos de buscar dicha pose (obvio) y
  // procedemos a calibrar el esqueleto sabiendo que la persona ha
  // adoptado una pose en particular (en nuestro caso, la pose Psi)
  printf("Pose %s for user %d\n", strPose, nId);
  g_UserGenerator.GetPoseDetectionCap().StopPoseDetection(nId);
  g_UserGenerator.GetSkeletonCap().RequestCalibration(nId, TRUE);
}

// Esta callback implementa la funcionalidad que seguiría
// inmediatamente a la función anterior. Concretamente, esta función
// será llamada mientras dure el proceso de calibración del esqueleto,
// y su objetivo es meramente informativo: indicará al usuario de la
// aplicación que su pose ha sido detectada y está en proceso de
```

```
// calibración.
void XN_CALLBACK_TYPE
Calibration_Start(xn::SkeletonCapability& capability, XnUserID nId,
                  void* pCookie)
{
  printf("Starting calibration for user %d\n", nId);
}

// Por último, esta función callback se ejecuta en el momento en que
// se ha completado el proceso de calibración del esqueleto.
void XN_CALLBACK_TYPE
Calibration_End(xn::SkeletonCapability& capability, XnUserID nId,
                XnBool bSuccess, void* pCookie)
{
  if (bSuccess)
  {
    // Si la calibración tuvo éxito, comenzamos a realizar el
    // seguimiento de las coordenadas del esqueleto, que ya ha sido
    // ajustado a la forma del usuario detectado y calibrado.
    printf("User calibrated\n");
    g_UserGenerator.GetSkeletonCap().StartTracking(nId);
  }
  else
  {
    // Si la calibración ha fallado, reanudamos nuevamente el proceso
    // de detección de pose del generador de usuario
    printf("Failed to calibrate user %d\n", nId);
   g_UserGenerator.GetPoseDetectionCap().StartPoseDetection(

POSE_TO_USE,
                                                            nId);
  }
}

void main()
{
  // Inicializamos la aplicación.
  XnStatus nRetVal = XN_STATUS_OK;

  xn::Context context;
  nRetVal = context.Init();

  // Inicializamos la variable global que declaramos previamente.
  nRetVal = g_UserGenerator.Create(context);

  // Ahora llega el momento de registrar las funciones callback
  // definidas anteriormente para que OpenNI las ejecute cuando se
  // produzcan las condiciones asociadas a cada una de ellas
  // (detección de nuevo usuario, pérdida de usuario, detección de
  // pose, inicio de calibración, etc.)
  // Para ello, el generador de usuario permite registrar callbacks
  // asociados a los tres bloques de casos previamente descritos:
  // callbacks de usuario (detección de nuevo usuario y pérdida de
  // usuario), callbacks de pose, y callbacks de calibración (inicio y
  // fin)
  XnCallbackHandle h1, h2, h3;
  g_UserGenerator.RegisterUserCallbacks(User_NewUser, User_LostUser,
                                        NULL, h1);
  g_UserGenerator.GetPoseDetectionCap().RegisterToPoseCallbacks(
      Pose_Detected, NULL, NULL, h2);
  g_UserGenerator.GetSkeletonCap().RegisterCalibrationCallbacks(
```

```
        Calibration_Start, Calibration_End, NULL, h3);
    // Es interesante notar en este caso que la detección de pose y la
    // detección del esqueleto no son funciones intrínsecas al generador
    // de usuario, sino que derivan de capacidades o capabilities
    // asociadas al mismo. Por tanto, no todos los generadores de datos
    // de usuario serán capaces de detectar poses y/o de calcular
    // esqueletos.

     // Con esta función, le indicamos al User Generator que realice el
     // seguimiento del esqueleto completo (de cabeza a piés).
     g_UserGenerator.GetSkeletonCap().SetSkeletonProfile(
        XN_SKEL_PROFILE_ALL);

    // Indicamos al contexto que ponga a todos los nodos definidos a
    // generar datos.
    nRetVal = context.StartGeneratingAll();

    // Y procedemos a trabajar con el bucle principal.
    while (TRUE)
    {
      // Esperamos a que todos los nodos de producción tengan datos
      // nuevos, y entonces actualizamos los datos mostrados por los
      // mismos.
      nRetVal = context.WaitAndUpdateAll();

      // En esta aplicación, vamos a extraer las coordenadas de la
      // cabeza del usuario una vez se ha calculado su esqueleto.

      // Damos soporte para un total de 16 usuarios.
      XnUserID aUsers[15];
      XnUInt16 nUsers = 15;

      // Preguntamos al generador de usuario cuantos usuarios ha
      // detectado.
      g_UserGenerator.GetUsers(aUsers, nUsers);

      // Para cada usuario
      for (int i = 0; i < nUsers; ++i)
      {
        // Preguntamos al generador de usuario si se está realizando
        // actualmente el seguimiento del esqueleto del usuario i-ésimo.
        if (g_UserGenerator.GetSkeletonCap().IsTracking(aUsers[i]))
        {
          // Definimos una variable "posición de articulación" para
          // almacenar los datos de la posición de la cabeza.
          XnSkeletonJointPosition Head;

          // Obtenemos los datos de la articulación del esqueleto
          // correspondiente a la cabeza (indexada por la constante
          // XN_SKEL_HEAD).
          g_UserGenerator.GetSkeletonCap().GetSkeletonJointPosition(
             aUsers[i], XN_SKEL_HEAD, Head);
          printf("%d: (%f,%f,%f) [%f]\n", aUsers[i],
                 Head.position.X, Head.position.Y, Head.position.Z,
                 Head.fConfidence);
        }
      }
    }

    context.Shutdown();
}
```