Inicialización, creación de nodos y lectura de datos.

```
XnStatus nRetVal = XN_STATUS_OK;

xn::Context context;

nRetVal = context.Init();

xn::DepthGenerator depth;
nRetVal = depth.Create(context);

nRetVal = context.StartGeneratingAll();

bool bShouldRun = true;
while (bShouldRun)
{
    nRetVal = context.WaitOneUpdateAll(depth);
    if (nRetVal != XN_STATUS_OK)
    {
        printf("Failed          updating          data:          %s\n",
xnGetStatusString(nRetVal));
        continue;
    }

    const XnDepthPixel* pDepthMap = depth.GetDepthMap();

}

context.Shutdown();
```

Cadenas de producción

```
xn::Query query;

nRetVal = query.SetVendor("MyVendor");

query.AddSupportedCapability(XN_CAPABILITY_SKELETON);

xn::NodeInfoList possibleChains;
nRetVal = context.EnumerateProductionTrees(XN_NODE_TYPE_USER, &query,
possibleChains, NULL);

xn::NodeInfo selected = *possibleChains.Begin();

nRetVal = context.CreateProductionTree(selected);

xn::UserGenerator userGen;
nRetVal = selected.GetInstance(userGen);
```

Gestión de mapas/imágenes.

```
XnStatus nRetVal = XN_STATUS_OK;

Context context;
nRetVal = context.Init();

DepthGenerator depth;
nRetVal = depth.Create(context);

XnMapOutputMode mapMode;
mapMode.nXRes = XN_VGA_X_RES;
mapMode.nYRes = XN_VGA_Y_RES;
mapMode.nFPS = 30;
nRetVal = depth.SetMapOutputMode(mapMode);

nRetVal = context.StartGeneratingAll();

XnUInt32 nMiddleIndex =
      XN_VGA_X_RES * XN_VGA_Y_RES/2 + XN_VGA_X_RES/2;

bShouldRun = true;
while (bShouldRun)
{
      nRetVal = context.WaitOneUpdateAll(depth);

      const XnDepthPixel* pDepthMap = depth.GetDepthMap();

      printf("Middle pixel is %u millimeters away\n",
            pDepthMap[nMiddleIndex]);
}

context.Shutdown();
```

Acceso a los datos de las coordenadas de la mano (Hand Point)

```
#define GESTURE_TO_USE "Click"

xn::GestureGenerator g_GestureGenerator;
xn::HandsGenerator g_HandsGenerator;

void XN_CALLBACK_TYPE
Gesture_Recognized(xn::GestureGenerator& generator,
                   const XnChar* strGesture,
                   const XnPoint3D* pIDPosition,
                   const XnPoint3D* pEndPosition, void* pCookie)
{
        printf("Gesture recognized: %s\n", strGesture);
        g_GestureGenerator.RemoveGesture(strGesture);
        g_HandsGenerator.StartTracking(*pEndPosition);
}

void XN_CALLBACK_TYPE
Gesture_Process(xn::GestureGenerator& generator,
                const XnChar* strGesture,
                const XnPoint3D* pPosition,
                XnFloat fProgress,
                void* pCookie)
{}

void XN_CALLBACK_TYPE
Hand_Create(xn::HandsGenerator& generator,
            XnUserID nId, const XnPoint3D* pPosition,
            XnFloat fTime, void* pCookie)
{
  printf("New Hand: %d @ (%f,%f,%f)\n", nId,
         pPosition->X, pPosition->Y, pPosition->Z);
}

void XN_CALLBACK_TYPE
Hand_Update(xn::HandsGenerator& generator,
            XnUserID nId, const XnPoint3D* pPosition,
            XnFloat fTime, void* pCookie)
{
}

void XN_CALLBACK_TYPE
Hand_Destroy(xn::HandsGenerator& generator,
             XnUserID nId, XnFloat fTime,
             void* pCookie)
{
  printf("Lost Hand: %d\n", nId);
  g_GestureGenerator.AddGesture(GESTURE_TO_USE, NULL);
}



void main()
{

  XnStatus nRetVal = XN_STATUS_OK;

  Context context;
  nRetVal = context.Init();
```

```
  nRetVal = g_GestureGenerator.Create(context);
  nRetVal = g_HandsGenerator.Create(context);

  XnCallbackHandle h1, h2;
  g_GestureGenerator.RegisterGestureCallbacks(Gesture_Recognized,
                                              Gesture_Process,
                                              NULL, h1);
  g_HandsGenerator.RegisterHandCallbacks(Hand_Create, Hand_Update,
                                         Hand_Destroy, NULL, h2);

  nRetVal = context.StartGeneratingAll();

  nRetVal = g_GestureGenerator.AddGesture(GESTURE_TO_USE);

  while (TRUE)
  {
    nRetVal = context.WaitAndUpdateAll();

  }

  context.Shutdown();
}
```

Acceso a los datos del generador de audio

```
Context context;
nRetVal = context.Init();

AudioGenerator audio;
nRetVal = audio.Create(context);

XnWaveOutputMode waveMode;
waveMode.nSampleRate = 44100;
waveMode.nChannels = 2;
waveMode.nBitsPerSample = 16;
nRetVal = audio.SetWaveOutputMode(waveMode);

while (true)
{
	nRetVal = context.WaitOneUpdateAll(audio);


	const XnUChar* pAudioBuf = audio.GetAudioBuffer();
	XnUInt32 nBufSize = audio.GetDataSize();

}

// Clean up
context.Shutdown();
```