FreeWill+ Objects and Actions

# Creating FreeWill Device

The *FreeWill Device* is the main *FreeWill* object responsible, among other things, for creating any other *FreeWill* object. This is how the *FreeWill Device* is created (using standard WinAPI function call):

HRESULT h;

IFWDevice \*pFWDevice = NULL;

h = CoCreateInstance(CLSID\_FWDevice, NULL, CLSCTX\_INPROC\_SERVER, IID\_IFWDevice, (void\*\*)&pFWDevice);

All parameters are COM-related constants, and the m\_pFWDevice is where the pointer to a newly created object is stored. As all COM function calls, this function returns the error code which may be tested like this:

if (FAILED(h)) throw ERROR\_FREEWILL;

This is the only case in which, in FreeWill, you would use this standard Windows API function. All subsequent *FreeWill* functions must be created through the *FreeWill Device*.

# FreeWill Objects

Here is the list of some standard *FreeWill Objects*:

* ***KineBone***: a bone, or main component of the kinematic chain. Each bone has a *Transform* which defines its location and rotation in the space, and usually also have a *Mesh*, which defines its shape.
* ***Transform***: represents a matrix transformation (translation or movement, rotation, scaling, perspective and so on).
* ***Mesh***: collection of 3D vertices and faces that defines the shape of a 3D object, typically attached to a ***KineBone*** object. Often a Material is used to define the mesh colour and look.
* ***Material***: either a solid colour or a bitmal-based texture, provides the look for a mesh.
* ***Scene***: the root node of the scene structure, may contain Scene Objects, Cameras and Lighting
* ***SceneObject***: top level bone, attached directly to the scene.
* ***Camera*** is derived from the bone (*KineBone*), has its own transform which defines its location and orientation, can be moved and animated, and defines how the scene is seen in the screen.
* ***DirLight****,* ***PointLight****,* ***SpotLight***: these three categories of lighting are also derived from bones, which helps to operate the light.
* ***Bounding*** represents bounding boxes.
* ***Renderer***: the renderer objects, in current implementation based on DirectX platform.
* ***Action***: these objects are responsible for animations, or behavior of other objects.
* ***Body*** is a friendly interface to a part of kinematical hierarchy that represent human or animal body, provides easy access to each bone through a number of unified labels.

One way to create an object of any type is to use functionsavailable in *FreeWill Device* object, for example this is how a *Renderer* object is typically created:

h = m\_pFWDevice->CreateObject(L"Renderer", IID\_IRenderer, (IFWUnknown\*\*)&m\_pRenderer);

The first parameter is known as the **noun**: it's simply the object class name. The second parameter is the COM interface ID (a COM specific constant value, should be IID\_IAction for action objects, IID\_IKineNode for kinematic node objects etc.). The final parameter is where the pointer of the created object is stored. As most COM functions, it retuens the error codes (S\_OK for success).

Any object created must be released when it is not used any more:

m\_pRenderer->Release();

# How to create Actions

The function IFWDevice::CreateObject, described in the previous section, is fine for creating most of the FreeWill objects, but not for creating *Actions*. To create an action, you will almost always need to pass a long list of additional parameters. The *FreeWill* object provides special function, IFWDevice::CreateObjectEx, but its usage is a bit complex and instead we will use a facilitating function.

Here is one of the facilitating functions you will typically use to create actions. Note that, as it is a global function, you have to pass the pointer to your *FreeWill* device object as the first parameter:

IFWUnknown \*p = NULL;

p = FWCreateObj(pFWDevice, noun, verb, list-of-parameters ...);

// use the created object here, then release it!

...

p->Release();

Working with actions, calling the *Release* each time you create a new action is a burden. This special form of the function does not require calling the *Release* and is therefore quite commonly used*:*

IFWUnknown \*p = NULL;

p = FWCreateObjWeakPtr(pFWDevice, noun, verb, list-of-parameters ...);

// you must not call p->Release() now! It's already done!

Here is the explanation of the function call parameters:

* pFWDevice – pointer to your *FreeWill* object, mandatory.
* noun – the object class name.
* verb – a more specific description of what the object is for, mandatory for actions. Examples of action verbs include L"Move", L"Rotate", L"RotateTo", L"Step" and L"Walk".
* list-of-parameters – an open list, dependent on the object noun and verb, optional.

Action objects' verbs and parameters are explained in detail in the next two sections.

**Examples:**

// generic action to start after 5 seconds from start of the system, for 2 seconds

IFWUnknown \*p = NULL;

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Generic", m\_pActionTick, 5000, 2000, L"");

// another generic action, to start immediately after, for 1 second

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Generic", m\_pActionTick, p, 1000, L"");

# Actions – Common parameters and data types

Again, the three first parameters of the FWCreateObj / FWCreateObjWeakPtr function call are:

* pointer to the *FreeWill* object
* noun: L"Action"
* verb: one of the verbs as specified in the next section

The following three or four parameters are common for all actions, whatever their verb is. These are:

* The tick source, or the higher level action – usually the "tick" action. It may be NULL in time of creation – such action needs further initialization (or: subscription) at a later stage.
* Start time – It may be the number of milliseconds from the beginning of the scene. It may also be a pointer to another (previous) action – in that case the new action will begin immediately after the previous one terminated
* Planned duration, in milliseconds (not all actions will strictly obey)
* Style (optional!) – a short string containing additional hints regarding the style of the action; the meaning of this parameter depends on the action

**Any further parameters** will be different, depending on the **Verb** you specified, but they will always be one of the following:

* **uint** (unsigned integer),
* **float**
* **string** (a UNICODE like L"this one"),
* **vector**, to provide 3D coordinates. There are two optional ways to specify a vector:
  + as a FWVECTOR object,
  + as a sequence of three **float** parameters.
* **FreeWill object**, a pointer to any FreeWill object – FWPUNKNOWN,
* **Body**, a pointer to the IBody *FreeWill* object,
* **Bone**, an object of the *KineBone* class. There are three optional ways to specify a bone:
  + a direct pointer to a *KineBone* object,
  + a pointer to any other object and a string containing the label of its child object
  + a pointer to a *Body* object followed by a bone index,
  + just a bone index – only applicable if the *Body* object appeared anywhere earlier in the list.

See the *Move* action for example of vector definition, and *Rotate* for examples of the bone definitions.

# Actions

Here is the list of action verbs. The common parameters, as described in the previous section, are not shown.

## Generic

This is a “do nothing” action, may be used to insert idle time into a sequence of actions.

Parameters:

* none

Example:

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Generic", m\_pActionTick, p, 1000, L"");

## Move

Animates motion of the object. Destination vector is relative to the current position, in object local coordinates.

Parameters:

* Bone: object to be moved
* Vector: the target position

Examples below present two different ways of specifying a vector:

FWVECTOR vector = { 0, 0, 12 };

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Move", m\_pActionTick, p, 1000, pBone, vector);

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Move", m\_pActionTick, p, 1000, pBone, 0, 0, 12);

## MoveTo

Animates motion of an object. Unlike the previous one, the destination vector is either in the scene global coordinates, or the coordinates of the reference bone if specified.

Parameters:

* Bone: object to be moved
* Vector: the target position
* Reference bone (optional): specifies the coordinate system for the target position, global if omitted.

## Rotate

Animates rotation of an object. Target rotation is relative to the current orientation of the object, in object local coordinates. Usually used to move limbs and other body parts.

Parameters:

* Bone: object to be rotated
* Transform: specifies the rotation

The example below shows how to create and initialize a transform object, pass it to the action and then release:

ITransform \*pT;

pScene->CreateCompatibleTransform(&pT); // pScene or any other Bone derived object may be used

pT->FromRotationZ(M\_PI / 2); // creates a rotation around Z axis by π/2 radians

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Rotate", m\_pActionTick, p, 1000, pBone, pT);

pT->Release();

The examples below show various ways of specifying a bone for rotation:

// use this if you have a direct pointer to the bone object, pBone:

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Rotate", m\_pActionTick, p, 1000, pBone, pT);

// to rotate a child object of the pScene bone, identified as "Ball01.Ball01"

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Rotate", m\_pActionTick, p, 1000, pScene, L"Ball01.Ball01", pT);

// to rotate the left leg of the body object provided:

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Rotate", m\_pActionTick, p, 1000, pBody, BODY\_LEG+BODY\_LEFT, pT);

## RotateTo

Animates rotation of an object using absolute rotation transform and the coordinate system specified by the reference bone.

Parameters:

* Bone: object to be rotated
* Transform: specifies the rotation
* Reference bone (optional): specifies the coordinate system to which the rotation transform refers, global if omitted.

Implementation remark: currently, the reference bone must be provided.

Examples:

This code will rotate the body right hand so that, after the action is completed, it is rotated by π/4 radians around the X axis, in relation to the right arm:

ITransform \*pT = NULL;

m\_pScene->CreateCompatibleTransform(&pT);

pT->FromRotationX(M\_PI / 4);

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"RotateTo", m\_pActionTick, 0, 1500,

pBody, BODY\_HAND + BODY\_RIGHT, pT, pBody, BODY\_ARM + BODY\_RIGHT);

pT->Release();

If using the body object and body part index, the second pointer to the body object may be omitted:

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"RotateTo", m\_pActionTick, 0, 1500,

pBody, BODY\_HAND + BODY\_RIGHT, pT, BODY\_ARM + BODY\_RIGHT);

## RotateInv

Rotation using an inverted kinematic chain.

Parameters:

* Bone: object to be rotated (terminates the inverted kinematic chain)
* Bone: Root of the inverted kinematic chain
* Transform: specifies the rotation
* Optional: additional rotation transform to be performed.

Additional bones and transforms may be added to this list.

This example is a part of the walking action, in which the pelvis is affected by an inverted kinematic chain in which the foot (in contact with the ground) is the new (temporary) root, and the leg is also rotating:

FWCreateObjWeakPtr(FWDevice(), L"Action", L"RotateInv", pAction, nStartTime, nPeriod,

m\_pBody, BODY\_PELVIS, BODY\_FOOT, pT1, pT2, BODY\_LEG, pT3);

## MultiRotate

## MultiRotateTo

These two actions are very similar to *Rotate* and *RotateTo* but may operate on several objects in the same time. This is not only for simplification – these rotations are also significantly optimized.

Parameters:

* Number of objects affected
* Same as either *Rotate* or *RotateTo*, repeated appropriate number of times.

Example is taken from *Grasp* action internal implementation and show how to operate with 4 fingers at a time:

FWCreateObjWeakPtr(FWDevice(), L"Action", L"MultiRotateTo", pAction, m\_nStartTime, m\_nPeriod, pStyle,

12,

pBody,

BODY\_FINGER + BODY\_1 + 0, pT45, BODY\_FINGER + BODY\_1 + 1, pT15, BODY\_FINGER + BODY\_1 + 2, pT30,

BODY\_FINGER + BODY\_2 + 0, pT45, BODY\_FINGER + BODY\_2 + 1, pT15, BODY\_FINGER + BODY\_2 + 2, pT30,

BODY\_FINGER + BODY\_3 + 0, pT45, BODY\_FINGER + BODY\_3 + 1, pT15, BODY\_FINGER + BODY\_3 + 2, pT30,

BODY\_FINGER + BODY\_4 + 0, pT45, BODY\_FINGER + BODY\_4 + 1, pT15, BODY\_FINGER + BODY\_4 + 2, pT30);

## Reach

Performs the reaching gesture towards a specified location or object. This location or object is called **target**. The spot on the character's body (or nearby) with which it is touching the target is called a **terminator**.   
Most often, character's left or right hand is used as a terminator.

Parameters:

* Body object
* Bone: terminator, usually hand.
* Vector (optional), added to the terminator bone position.
* Bone (optional): target object. May be NULL if the target is specified as a vector in the global coordinate system.
* Vector (optional): added to the target object position, or used as global coordinates if no target object is specified.
* Float: elbow correction angle, used to fix the elbow position when the action is performed.

Styles:

* Either L"left" or L"right" to specify which hand is used. Right hand if none specified.

Alternatively, the 1st parameter may be replaced with the following two:

* Bone: first section of the limb involved, usually arm.
* Bone: second section of the limb involved, usually forearm.

With the second form of parameters, the style is ignored.

Example:

FWCreateObjWeakPtr(FWDevice(), L"Action", L"Reach", pAction, p, 1000, L"right",

pBody, BODY\_FINGER + BODY\_MIDDLE, pTarget, 0.9f, 0.0f, 1.8f);

## Grasp

Performs the grasp gesture with all fingers of a single hand.

* Body object

Styles:

* Either L"left" or L"right" to specify which hand is used. Right hand if none specified.

Example (grasp with the left hand):

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Grasp", m\_pActionTick, p, 1000, L"left", pBody);

## Take

Combination of *Reach* and *Grasp* actions. An easy-to-use action which always use predefined body parts (arm, forearm, hand, fingers).

Parameters:

* Body object
* Target object

Styles:

* Either L"left" or L"right" to specify which hand is used. Right hand if none specified.

Example:

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Take", m\_pActionTick, p, 1000, L"left",

pBody, pScene, L"Sphere02.Sphere02");

## Point

Use the arm, forearm, hand and fingers to point to a specified object or location.

Parameters:

* Body object
* Target object
* Target vector (optional) – added to the target object location

Styles:

* Either L"left" or L"right" to specify which hand is used. Right hand if none specified.

Example:

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Point", m\_pActionTick, p, 1000, L"left",

pBody, pScene, L"Sphere02.Sphere02", , 0.9f, 0.0f, 1.8f);

## Bend

Bends the body at the height of the pelvis (makes a bow).

Parameters:

* Body object
* Float: bend angle in radians; optional, the default value is 50 degrees.
* Float: additional deflection from the vertical, performed at the feet level; optional, the default value is 0.

Example:

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Bend", m\_pActionTick, p, nDur, m\_pBody1, DEG2RAD(75));

## Squat

Performs a squat (crouch)

Parameters:

* Body object
* Float: bend angle in radians, optional, the default value is 50 degrees.
* Float: additional deflection from the vertical, performed at the feet level; optional, default is 8 degrees.
* Float: bend angle in the knees; optional, the default value is 30 degrees.

Example:

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Squat", m\_pActionTick, p, nDur, m\_pBody1);

## Swing

Bends the body, just like the *Bend* action, but additionally swings a leg backwards to keep the balance.

Parameters:

* Body object
* Float: bend angle in radians, optional, the default value is 40 degrees.
* Float: leg swing angle in radians, default is ¾ of the bend angle value.

Example:

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Squat", m\_pActionTick, p, nDur, m\_pBody1, DEG2RAD(60));

## Step

Performs a single step. Rarely used directly – most often indirectly through the Walk and Turn actions.

Parameters:

* Body object
* Float: Step angle – half of the angle between the legs when the step is completed
* Float: Heading angle – specifies the angle by which the character would turn right or left after the step is completed.

Styles:

* Either L"left" or L"right" to specify which leg would be used first to step forward. This setting will be ignored if any leg is already put forward – the other one would always step first.
* Either L"force-left" or L"force-right" to specify which leg would be used first to step forward, even if one of the legs is already put forward.

Example – perfrorms a default step:

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Step", m\_pActionTick, p, 150, m\_pBody);

## Walk

A complex action involving a series of steps.

Parameters:

* Body object
* Float: X coordinate of the target position
* Float: Y coordinate of the target position
* Float: maximum length of the step, optional, default value 0 makes the system to determine this value automatically. May be used to change from slow to fast walk.
* Float: maximum turn value (how sharply can the character turn) – default value is 80 degrees.

Styles:

* Either L"left" or L"right" to specify which leg would be used first to step forward. This setting will be ignored if any leg is already put forward – the other one would always step first.
* Either L"force-left" or L"force-right" to specify which leg would be used first to step forward, even if one of the legs is already put forward.
* L"open" if, after the final step is completed, one leg should be left put forward and the other backward to allow for another walking sequence.
* L"close" if, after the final step is completed, the character should stand normally with both legs put together.

Example: four walking sequences ending at the "stand up" position. Maximum step size is 12.

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Walk", m\_pActionTick, p, 150, L"open", m\_pBody1,

-46.0f, -57.0f, 12);

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Walk", m\_pActionTick, p, 150, L"open", m\_pBody1,

118.0f, 50.0f, 12);

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Walk", m\_pActionTick, p, 150, L"open", m\_pBody1, -

46.0f, 50.0f, 12);

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Walk", m\_pActionTick, p, 150, L"close", m\_pBody1,

118.0f, -57.0f, 12);

## Turn

A complex action involving a series of steps resulting in turning the character around.

Parameters:

* Body object
* Float: Total rotation of the character; optional, π or 180 degrees used as default
* Float: Step count; optional, the default value taken so that each steps rotates the character by 60 degrees on the average
* Float: Distance to proceed after completion the turn, 0 by default

Styles: the same as with *Walk* action.

Examples:

// standard turn: 180 degrees in three steps

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Turn", m\_pActionTick, p, 150, m\_pBody1);

// full turn around: 360 degrees in seven steps

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Turn", m\_pActionTick, p, 150, m\_pBody1, M\_PI\*2, 7);

## Wait

Waits until a specified time.

Parameters:

* Body object
* End time – allows to specify waiting time in the absolute animation time rather than simply duration of the action.

Example – makes the character wait until 15 seconds after the animation started:

p = FWCreateObjWeakPtr(m\_pFWDevice, L"Action", L"Wait", m\_pActionTick, p, nDur, m\_pBody1, 15000);

## DetColl

Planned implementation.