Defining Custom Tips

This version of the ArmDLL32/64 allows users to calculate the relative position offset of a custom tip from the default tip that ships with MicroScribe. These offsets along with home encoder offsets (critical for defining a custom home position) can be saved under an associated tip name and index. ArmDLL32/64 supports 4 tip definitions per MicroScribe. These tip definitions should be defined once for each custom tip. They are stored for immediate use or for use after subsequent power-up of the MicroScribe. The previously supported capability of adding offsets in the Z-direction (along the length of the stylus) through the **ArmCustomTip** call is still supported, but it is suggested that developers revise their products to take advantage of **ArmSaveTipProfile**.

A certain sequence of steps is needed to define and store the definition of a new custom tip. This sequence is outlined below. Since the custom tip definitions are stored per device, the tip definition stored for one MicroScribe will not be available for another unit. The ArmDLL32/64 will maintain the concept of a **Current Tip** for the device, which can be accessed through a call **toArmGetTipProfile**.

To Define a New Tip Using ArmSaveTipProfile:

1) Set the MicroScribe in the standard home position with the default/master tip in place. Home the device. Read the home encoder values (using **ArmGetEncoderCount**) into an integer (int) array.

2) Switch to your custom tip and set the MicroScribe back in the desired home position, but do not home the device. You will home the device later in **Step 5**. If your tip rests securely and repeatably in the standard home divot, you may designate this as your home position; or you may supply your own secure, repeatable alternate home position. Keep in mind that there can only be one custom home position per tip. Read the new encoder values in this position using **ArmGetEncoderCount** into an integer (int) array. Subtract the array of encoder values in **Step 1** from the array of encoder values in this step and record the differences. Again, do not home the device until you get to **Step 5**.

3) Collect the position and orientation of 20 or more points (using **ArmGetTipPosition, ArmGetTipOrientation** and **ArmGetTipOrientationUnitVector**) on the calibration fixture as described in the **MicroScribe Utility Software** manual. Use these three arrays as the arguments to the **ArmGenerateTipPositionOffsetEx** function to return your custom tip's offsets in X,Y,Z directions. For compatibility with older software, **ArmGenerateTipPositionOffest** is still available for which you only need the data provided by **ArmGetTipPosition** and **ArmGetTipOrientation**, but it is suggested that you upgrade to **ArmGenerateTipPositionOffsetEx** because it better handles cases when the custom probe in nearly identical in configuration to the default/master probe.

4) Call the ArmSaveTipProfile function with: an index (1 through 4), the encoder offsets as calculated in **Step 2**, the position offsets as calculated in **Step 3** and a name string to store the tip definition for future uses. This function permits partial changes to the tip definition by accepting NULL pointers for arguments you do not want to modify (e.g. it is possible to re-save only the encoder offsets, but not the name and tip position offsets, etc.).

5) Set the MicroScribe in the home position defined in **Step 2**. Home the device. Call the **ArmSetTipProfile** function with the index argument from **Step 4** to change the **Current Tip** of the device to the one you have just saved.

At this point you have calibrated a custom tip and made it the currently active tip The following is sample code implementing these steps, with comments indicating where the process flow would await user interactions, is shown below:

```
extern "C"
{
	#include "ArmDll32.h"
}
#include <vector>
bool CalibrateProbe(int defaultProbeId, int probeId, const char* probeName)
{
	bool result = false;
	// Pre-acquire homing differences before gathering calibration points
	int
		homeEncoder[NUM_ENCODERS],
		currEncoder[NUM_ENCODERS],
		encoderDelta[NUM_ENCODERS];
	// Ask user to place in home position with default tip.....then do this...
	ArmSetSoftHome();
	ArmGetEncoderCount(homeEncoder);

	// Now ask user to place in home position with custom tip.....then do this...
	ArmGetEncoderCount(currEncoder);
	// ignore differences in base encoder
	encoderDelta[0] = 0;
	for (int i=1; i< NUM_ENCODERS; ++i)
	{
		encoderDelta[i] = currEncoder[i] - homeEncoder[i];
	}
	// DO NOT CALL ArmSetSoftHome() here.

	// Gather 20 points from MicroScribe using ArmDLL API.
	// Include possible error conditions we should handle, being unfamiliar with
	// the ways ArmDLL may report errors.
	std::vector<length_3D> pts;
	std::vector<angle_3D> orient;
	std::vector<angle_3D> unit_orient;
	int minNumPts = 20;
	bool gatheringPoints = true;
	while (pts.size() <= minNumPts)
	{
		// after a point is passed in from the hardware (same as for collecting points
		// from user during general use).....do this...
		length_3D new_pt;
		angle_3D new_angle;
		angle_3D new_unit_orientation;

		ArmGetTipPosition(&new_pt);
		ArmGetTipOrientation(&new_angle);
		ArmGetTipOrientationUnitVector(&new_unit_orientation);
		pts.push_back(new_pt);
		orient.push_back(new_angle);
		unit_orient.push_back(new_unit_orientation);
	}
	// Compute the new probe offset using ArmDLL API.
	float lengthOffset[3] = {0,0,0};
	length_3D posOffset;
```

```
	int result = ArmGenerateTipPositionOffsetEx(&pts.front(), &orient.front(), minNumPts,
				posOffset, &unit_orient.front());
	if (result == PTS_TOO_CLOSE)
	{
		int userChoice = DisplayMessage("Calibration points or angles too close to one another.");

		if (userChoice == IDOK)
		{
			lengthOffset[0] = 0.0;
			lengthOffset[1] = 0.0;
			lengthOffset[2] = 0.0;
		}
		else
		{
			ArmSetTipProfile(defaultProbeId);
			ArmCustomTip(0.0);
			DisplayMessage("Place the device in home position with the Default tip \nand press the Home button before
					acquiring data");
		}
	}
	else if (result != ARM_SUCCESS)
	{
		DisplayMessage("Calibration incomplete. Please collect 20 additional points");
		pts.clear();
		orient.clear();
		unit_orient.clear();
	}
	else
	{
		lengthOffset[0] = posOffset.x;
		lengthOffset[1] = posOffset.y;
		lengthOffset[2] = posOffset.z;
	}
	// Save probe with specified probeID and probeName using ArmDLL API.
	if (result)
	{
		if (ArmSaveTipProfile(probeId, encoderDelta, lengthOffset, probeName) != ARM_SUCCESS)
		{
			DisplayMessage("New tip has not been saved properly");
			result = false;
		}
		else
		{
			DisplayMessage("New tip profile has been saved");
		}
	}
	return result;
}
```

Notes:

1. On subsequent power-ups, MicroScribe will automatically assume the home encoder offsets and tip position offsets for the **Current Tip**. Therefore, be sure to use the proper physical tip in proper home position on power-up. (The last tip used, 0-4 with 0 representing the default tip, is saved by ArmDLL32/64 in the registry of the computer for subsequent use.)

   This also means that if you last sat and used a custom tip you do not have to switch to the default/master tip when starting up your MicroScribe for the first time in a subsequent session. This holds true if you last tip used was the default/master tip or a custom tip. If you last used an offset tip, you must start and home your MicroScribe with the default/master tip first and then mount the alternate tip and call

ArmCustomTip with the correct offset length before reusing that tip in subsequent sessions.

2. If you want to switch to a different custom tip that you have previously saved, call the **ArmSetTipProfile** function with the appropriate index 1-4. This function should be called while the MicroScribe is in the home position defined for the new tip and the device needs to be homed after the function call. Thus, it is advised that the users physically switch the tip before making the **ArmSetTipProfile** function call.

3. The **ArmSetTipProfile** function internally calls the **ArmSetHomeEncoderOffset** function and the **ArmSetTipPositionOffset** function. Therefore, it will void any offsets previously set by these two functions. Similarly, calling these two functions after calling **ArmSetTipProfile** will void the offsets you are currently using for your custom tip. (Although it will not alter the stored values for the tip definition).

4. If you follow these steps to define a new custom tip, you do not need to follow the instructions in the **Custom Home Position** section to define a new custom home. You just need to make sure that the home position you use in **Step 2** is your desired custom home position.

5. **ArmCustomTip** should only be used when the Current Tip is 0 (the default). The most recent call to **ArmSetTipProfile** or **ArmCustomTip** will override the offsets set by the previous call to either one.