Click Here

Regardless what your use case is, be it hatching newly created entity in block, or whatever, the actual issue is that if you want to create a hatch, you CAN ONLY append HtachLoops with one or more entities (typically curves) that is database-residing, as ObjectIdCollection. If you use Editor.TraceBoundary() method to detect a boundary, this method, if successful, return a one or more DBObjects (Entities) in DBObjectCollection. They ARE NOT db-residing objects, thus their ObjectId is ObjectId.Null. It is up to the calling process to decide what to do with the returned non-db-residing objects. You can add then into the database first, and then use them to create HatchLopp. However, if you have some entities created already (a circle, in your case), but for some very odd reasons you know know it is there created by your code, and you know you want to use it to hatch it, but you have no way to get its ObjectId, therefore you have to use TraceBoundary() to find out the possible loop as boundary. If you do have a known point and want to find if it is inside a single closed curve (so you can use that closed curve to hatch), it might be much easier to test each curve in the drawing to see if the point is inside. If the curve is circle, it is even simple: you simply loop through all circles to see if the distance form the point to circle's center is equal or less than the radius. But for whatever reasons if you must or can only use TraceBoundary() to find the boundary, but only want to do hatch with existing entity, then you need to compare the returned non-db-residing entities with existing entities in drawing geometrically to find the exact corresponding existing entities for creating hatch (and you will dispose all the DBObjects returned by TraceBoundary() after the comparison is done). IMO, it only makes sense to use TraceBoundary() to get outmost loop for hatch WHEN the area to be hatched is formed by multiple entities. In your particular case, if the test point used in TraceBoundary() is inside a circle, the returned DBCollection would only have one DBObject, which should be a circle with exact same center/radius as the existing circle. So, you can use the non-db-residing circle's center/radius to search the drawing for the "real circle", and then do the hatch. The real issue here is, you must know that 1). TraceBoundary() returns a set of non-database-residing objects, and you decide how to use them (adding to DB, or using their geometric info then disposing them); 2). curve(s) for hatchloop must be db-residing objects. Norman YuanDrive CAD With Code and this on AutoCAD 2024 where the resulting boundary is a closed polylineTIA