Code Review

The code is pretty readable and understandable.

But there are hidden risks in its functionality. The main disadvantage is the mixed usage between raw pointer and shared_ptr.
In this transition class PlmDatabaseId a raw pointer _pExtPlmId is as class member defined. To cooperate with the utility functions which use shared_ptr as argument, a local shared_ptr is created in function QueryExternalPlmId and QueryPlmId.

```
11      {
12          std::shared_ptr<ExternalPlmId> extPlmId;
13          if (PlmDBUtils::QueryDBId(iPlmId.plmType, iPlmId.plmName, iPlmId.plmRevision, extPlmId) == E_FAIL)
14          {
15              return E_FAIL;
16          }
29      {
30          return PlmDBUtils::QueryDBProperties(std::shared_ptr<ExternalPlmId>(ipExtPlmId), oPlmId.plmType, oPlmId.plmName, oPlmId.plm
31      }
32  }
```

It causes the trouble that the pointee of the raw pointer will be deleted when the shared_ptr goes out of scope and automatically deleted. Therefore there is risk using the constructor PlmDatabaseId(ExternalPlmId* iExtPlmId), PlmDatabaseId(const PlmIdentifier iPlmId), PlmDatabaseId(std::string& iString), which I have already commented in the code.

My advice is to use shared_ptr of _pExtPlmId as a class member instead of raw pointer.
Or second solution: define the local shared_ptr directly inside the unnamed namespace instead in the including functions. Thus the shared_ptr has static lifetime and crashed in the end of the program. But it is not recommended. If the raw pointer calls delete somewhere, it can cause double delete error.

Also an addition: Better in header file add the header guard (preprocessor #ifndef, #define, #endif)