This document contains steps associated with some procedures for modifying the omc code base to extend functionality.

**Add a new symbol to the Symbol hierarchy**

The following steps were documented during the addition of the new symbol DimensionSymbol to the Symbol hierarchy.

1. In the ompp solution, set the active project to omc.
2. Choose an existing symbol with similar properties, e.g. TableExpressionSymbol has some similarities to the new DimensionSymbol.
3. In Windows Explorer, create copies of the declaration (.h) and implementation files (.cpp) of the existing symbol, and rename them using the new symbol.
4. Use “Add” in the Tortoise SVN context menu to mark the two new source files (DimensionSymbol.h, DimensionSymbol.cpp) for subsequent upload to SVN (but don’t check in yet).
5. Use Solution Explorer to add the new header file to the Header Files group, and the new implementation file to the Source Files group.
6. Open the file ast.h, and add a #include statement for the new header file. Close ast.h.
7. Open the two new files for editing, and close any other files open in the editor.
8. Use global replace (Ctrl-Shift-H) on the two open files to change all uses of the original class name in the copied files (TableExpressionSymbol) to the new class name (DimensionSymbol). Global replace settings are case sensitive, whole word, all open documents. At the moment, 11 replacements are made.
9. At the beginning of the class declaration, check and change the base class if necessary, and verify that the typedef declaration for ‘super’ 3 lines below corresponds. Very important, as this is the mechanism used to implement hierarchical calling chains. Bugs here have odd symptoms and can be difficult to track down.
10. Edit doxygen comment for the new class in the header file.
11. Continue with minimal implementation related to the new .h and .cpp files, but  
    - avoid references to the new class in other code for the moment (e.g. grammar file parser.y)  
    - avoid implementing new class-specific functionality  
    Typical changes at this point include:  
    - remove irrelevant members  
    - remove irrelevant arguments for constructor  
    - remove #include for unused header files  
    - verify and edit doxygen comments for all class members in the .h file.
12. Do trial compilation of the implementation file (TableDimensionSymbol.cpp), using Ctrl-F7, and fix errors as required.
13. Build omc. There should be no errors.
14. Create code snapshot using SVN or other method (optional).
15. Continue with implementation of functionality for the new class.

**Rename a symbol in the Symbol hierarchy**

The following steps were documented during the renaming of the symbol DerivedTablePlaceholderSymbol to DerivedTableMeasureSymbol to the Symbol hierarchy.

1. Check that there are no outstanding check-ins, and that the current version is stable and tested.
2. Close the ompp solution
3. In Windows Explorer, use TortoiseSVN to rename the declaration (.h) for the symbol. TortoiseSVN will helpfully prompt to replace the corresponding implementation file (.cpp) for the symbol.
4. Open the ompp solution.
5. Use Solution Explorer in the Header Files group to Remove the old header file name, and Add > Existing Item… for the renamed new header file. Do the same for the implementation file in the Source Files group.
6. Open the renamed declaration file to give context to VS for the upcoming global replace.  
   Use global replace (Ctrl-Shift-H) with the following settings:  
   Find what: DerivedTablePlaceholderSymbol  
   Replace with: DerivedTableMeasureSymbol  
   Look in: Current Project  
   Find options > Match Case  
   Find options > Match whole word  
   Look at these file types: \*.cpp;\*.h;\*.y  
   Click Replace All
7. File > Save all [note, will also save .sln and vcxproj file]
8. Rebuild omc, perform sanity test.
9. Check in changes to repository. Message can be something like  
   omc: name change in Symbol hierarchy: DerivedTablePlaceholderSymbol ==> DerivedTableMeasureSymbol

**Add a new accumulator**

The following steps were documented during the addition of the new experimental accumulator gini\_obs\_all().

1. Add token for new accumulator to parser.  
   - adding a new token requires exact correspondence between parser.y and Symbol.cpp, as indicated in source comments.  
   - open parser.y and Symbol.cpp, side by side.  
   - search for comment ‘// body level om keywords, in alphabetic order’, in both.  
   - scroll down in each to find appropriate place to insert new keyword, in alphabetic order  
   - copy, paste & edit existing line in each:  
     
    %token <val\_token> TK\_gini\_obs\_all "gini\_obs\_all"  
    { token::TK\_gini\_obs\_all, "gini\_obs\_all" },  
     
   - save all, build
2. Add accumulator to parser  
   - parser.y, find line ‘table\_accumulator:’  
   - add new accumulator token to list of accumulators  
     
    | TK\_gini\_obs\_all
3. Add accumulator to sanity check assert in TableAccumulatorSymbol constructor, in TableAccumulatorSymbol.h  
     
    || accumulator == token::TK\_gini\_obs\_all  
     
   build.
4. Assign appropriate initial value in TableSymbol::cxx\_definition\_global
5. If the accumulator has an associated collection of observations, indicate so in the constructor body of TableAccumulatorSymbol in the assignment of has\_observation\_collection