## Implement import types

Compiler was designed with import types in mind, but it was never fully functional. Well, until now. I experimented with different approaches and settled down on the following one. Every place where import type could be seen, that is in:

* variable type
* struct/union/class field type
* function return type
* array element type
* bitfield base type
* pointer target type

type is represented by a pair of types. If during the first pass type is imported, it goes to m\_pType\_i element of the pair. During CalcLayout phase of compilation process, m\_pType is filled with the actual (resolved) type. Special case is pointer type. As you remember there are 4 typekinds of those:

* data pointer
* class pointer
* function pointer
* property pointer

If compiler sees declaration like: “import\_type\* a” it has no idea which pointer kind should be created. Therefore, it creates a special import type (CImportPtrType) which after all the imports are resolved, could be mutated into one of the above four.

## Turn classes into directly creatable types

Classes are no longer implicitly converted to class pointers. Class instances are directly allocable as variables and member fields. Hence, 'member new' is gone. ‘static new’ is also not necessary, but I didn’t artificially block it: it doesn’t require special processing and doesn’t contradict anything. Classes cannot be moved (and by extension, neither passed as arguments, returned as retvals, or be members of moveable types such as structs or arrays), although checks for those are not yet implemented.

## Turn multicasts into classes

Multicasts are now auto-generated classes (rather than separate typekinds), implementation resides in CMulticastClassType. Pointers to multicasts are therefore, class pointers (rather than data pointers as before).

## Turn autoevs into classes

Autoevs, as multicasts, are auto-generated classes, implementation resides in CAutoEvClassType. Pointers to autoevs are class pointers (special autoev-pointer type is gone). Another thing that is gone, is autoev-class. Since autoev is a class anyways, having a special kind of autoev is kind of redundant. However, now we have a dilemma: when we declare an autoev, do we want to declare a class instance, or declare a class? We solve this dilemma with storage specifier. When it’s ‘typedef’ we declare an autoev class. Otherwise, we declare an autoev-typed variable (field)

## Declaration grammar modifications

Declarators no longer have ‘multicast’ and ‘event’ prefixes (which are type modifiers now), there is only one declarator prefix and that is a pointer. This was done to address the same issue as was with class declaration before, and that is type-modifier order sensitivity. Consult CDeclTypeCalc to see how ‘multicast’ type modifier is processed.

## Cleanup compile process

Compile is now strictly formalized. CFunctionMgr no longer does all the job, instead items needed to be CalcLayout ()-ed or Compile ()-ed, are added into the corresponding lists of CModule and processed within the main compile pipeline

1. pass1 compile all modules to discover global items
2. link
3. resolve import types
4. process calclayout queue
5. allocate global variables (currently this is done during compiling of he module constructor)
6. process compile queue

Resolving of orphan functions & autoevs is done during CalcLayout ().

## Implement 'nullable' semantics for function arguments

Welcome the new pointer flag EPtrTypeFlag\_Checked. All the pointer arguments (data, class, function, property) are verified during argument passing, rather than during actual access to the object pointed to by a pointer argument. This reduces the number of checks if argument is touched more than once, and allows catching errors in the moment of invocation with incorrect argumetns, rather than deep down the call stack. If pointer argument is declared with ‘nullable’ storage specifier, checks are NOT performed during argument passing.

## Redesign unsafe pointers

Remove unsafe pointer type kind (data, class, function, pointer) and welcome the new pointer flag EPtrTypeFlag\_Unsafe. This is done to improve symmetry of the types and also to allow unsafe operations previously simply impossible (calling a class/struct/union method using an unsafe pointer)

## Redesign closures and thunks

Before the generation of closure types, values and thunks was scattered across COperatorMgr and CFunctionMgr. Now the implementation is moved to dedicated closure class types (CFunctionClosureClassType, CPropertyClosureClassType, CDataClosureClassType) and dedicated thunks (CThunkFunction, CThunkProperty)

Weak closures could be traced differently when they are part of weak function/property pointer, therefore, there is no more need in special structs for weak function/property pointers with pfStrengthen field.

## Remove special augmented property pointer types

Augmented properties are the ones with one (or both) of the following values:

* onchange
* propvalue

Before pointers to augmented properties also contained a data pointer to augmentation block. Since multicasts are now classes, data pointer is no longer an option. Instead of devising multiple fine-grained pointer kinds, I decided to get rid of special “augmented” pointers altogether. Onchange is accessible through bind accessor (part of property vtable), and propvalue is not directly accessible (only through use of get/set accessors)

## Turn type kind checks into table lookups

Before many checks were implemented as IsXxxTypeKind (EType TypeKind) functions. I decided to replace all of those with lookups into the table of or-ed ETypeKindFlag values. It might or might not be more efficient, but it definitely is less “special” functions and looks cleaner.