Issue 850: How does name look up proceed in the parameter list of a friend function?

Add after 3.4.1 (basic.lookup.unqual) para 9:

+ Except for the names used in
+ .I template-arguments
+ of a
+ .I template-id ,
+ name look up for a name used in the function declarator for a
+ .CW friend
+ function that is a class member function
+ is first looked up in the scope of the member function's class, and if not
+ found, the look up follows the look up for unqualified names in the
+ definition of the class granting friendship.
+ .Cb
+ struct A {
+     typedef int AT;
+     void f1(AT);
+     void f2(float);
+ };
+ struct B {
+     typedef float BT;
+     friend void A::f1(AT); //\f2\& parameter type is \&\fPA::AT
+     friend void A::f2(BT); //\f2\& parameter type is \&\fPB::BT
+ };
+ .Ce
+ In the declaration of a
+ .CW friend
+ function that is a class member function,
+ the look up for a name used in the function declarator in a
+ .I template-argument
+ of a
+ .I template-id
+ follows the look up for unqualified names used in the definition of the
+ class granting friendship.

Issue 893: Lookup of conversion functions conversion-type-id and of template argument names is missing when these appear in qualified-ids

Change 3.4.3.1 (class.qual) para 1 as follows:

If the .I nested-name-specifier of a .I qualified-id
nominates a class, the name specified after the .I nested-name-specifier
is looked up in the scope of the class (_class.member.lookup_),
! except for the cases listed below.
The name shall represent
one or more members of that class or of one of its base classes
a class member can be referred to using a qualified-id at any point in its potential scope (_basic.scope.class_).

+ The exceptional cases are the following:
  + LI + a destructor name is looked up as specified in _basic.lookup.qual_;
  + LI + the + .I conversion-type-id + of an + .I operator-function-id + is looked up both in the scope of the class and + in the context in which the entire + .I postfix-expression + occurs and shall refer to the same type in both contexts;
  + LI + the + .I template-arguments + of a + .I template-id + are looked up in the context in which the entire + .I postfix-expression + occurs.

Change 3.4.3.2 (namespace.qual) para 1 as follows:

If the .I nested-name-specifier of a .I qualified-id nominates a namespace, the name specified after the .I nested-name-specifier ! is looked up in the scope of the namespace, except that + LI + the + .I conversion-type-id + of an + an + .I operator-function-id + is looked up both in the scope of the namespace and + in the context in which the entire + .I postfix-expression + occurs and shall refer to the same type in both contexts;
+ LI + the + .I template-arguments + of a + .I template-id + are looked up in the context in which the entire + .I postfix-expression + occurs.

.......................... Issue 916: conversion from pointer type to char* is not a static_cast

Change 3.8 (basic.life) para 5, third bullet as follows:

.LI the pointer is used as the operand of a
.CW static_cast
(_expr.static.cast_)
(except when the conversion is to
.CW void*,
+ and subsequently to
.CW char*,
or
.CW unsigned
.CW char ).

Change 3.8 (basic.life) para 6, third bullet as follows:

.LI
the lvalue is used as the operand of a
.CW static_cast
!(_expr.static.cast_) (except when the conversion is ultimately to
.CW char&
or
.CW unsigned
.CW char& ),
or

-------------------------------------------------------------------------------

Issue 892: ODR and string literals
==========
Add at the end of 7.1.2 (dcl.fct.spec) para 4

! A string literal in an
! .CW extern
! .CW inline
! function is the same object in different translation units.

-------------------------------------------------------------------------------

Issue 919: Can a using declaration refer to a template-id?
==========
Add after 7.3.3 (namespace.udecl) para 4:

+.P
+ A
+ .I using-declaration
+ shall not name a
+ .I template-id .
+ .E[
+ .Cb
+ class A {
+ public:
+ template <class T> void f(T);
+ template <class T> struct X { };
+ };
+ class B : public A {
+ public:
+ using A::f<double>; //\f2\& ill-formed\&\fP
+ using A::X<int>;  //\f2\& ill-formed\&\fP
+ };
+ .Ce
+ .E]

-------------------------------------------------------------------------------

Issue 902: When is 'template<class T> S(T);' used to generate a copy
==========
Add to 12.8 (class.copy) para 3:
A member function template is never instantiated to perform the copy of an
class object to an object of its class type.

struct S {
    template<typename T> S(T);
};

S f();

void g() {
    S a( f() ); // does not instantiate member template
}