UIM/X 2.9 -> 3.0 Migration Notes

These notes are provided as a general guide for the person who needs to migrate their UIM/X project from UIM/X 2.9 (or earlier) to UIM/X 3.0. Some of this information is available in the UIM/X 3.0 Installation Guide, Chapter 2.

Removing the Main Program and Makefile from your Project

Before attempting to load your project into UIM/X 3.0, you need to remove your project's main program and makefile since the templates for these files have changed from UIM/X 2.9 to 3.0. To do this:

- Load the .prj file into your favorite text editor.
- Find the specifications for the makefile. If your project uses the Ux Convenience Library, the makefile specifications look something like this:

```
*uxmake.class: PJuxmake
*uxmake.PjName: uxmake
*uxmake.PjParent: Untitled
*uxmake.PjFileName: Untitled.mk
*uxmake.PjDate: 0
*uxmake.PjMode: -1
*uxmake.PjBody: body of makefile…
*uxmake.PjEdited: 1
```

- Delete the specifications for PjBody and PjEdited.
- Find the specifications for the main program. If your project uses the Ux Convenience Library, the main program specifications look something like this:

```
*uxmain.class: PJuxmain
*uxmain.PjName: uxmain
*uxmain.PjParent: Untitled
*uxmain.PjFileName: Untitled.c
*uxmain.PjDate: 0
*uxmain.PjMode: -1
*uxmain.PjBody: body of main program…
*uxmain.PjExplicitEventLoop: body of event loop…
*uxmain.PjEdited: 1
*uxmain.PjHasImplicitLoop: 1
*uxmain.PjHasExplicitLoop: 0
```

- Delete the specifications for PjBody, PjExplicitEventLoop, PjEdited, PjHasImplicitLoop, and PjHasExplicitLoop.
- Save the project file.

Updating Resource Files and Option Files

Because UIM/X PTE has a different application class name than pre-3.0 releases of UIM/X, you will have to update any resource files (such as your .Xdefaults) where you set UIM/X resources. Also, if you made changes to the UIM/X resource file in /usr/lib/X11/app-defaults/, you will want to make the same changes to /usr/lib/X11/app-defaults/Uimx3_0 (/usr/openwin/lib/app-defaults/Uimx3_0 on Solaris).

Option files saved by UIM/X (.op and .uxrc files) use the application class name, so you may want to update these files as well.

Eliminating Protected Code

In UIM/X 2.9 (and earlier), C++ code entered by the user had to be protected by wrapping the code with the preprocessor directive:

```
#ifdef (__cplusplus)
#endif
```

Since UIM/X 3.0 now contains a C and C++ interpreter, the symbol "__cplusplus" is now defined by the interpreter. Therefore, C++ code which was previously protected will now be parsed by the UIM/X C++ interpreter. Certain C++ constructs and syntax are not supported by the UIM/X C++ interpreter, so must continue to be wrapped or changed, including:

- Templates
- C++ Method Calling using Implicit "this" pointer – UIM/X requires that methods added in, and generated by, UIM/X be called using the syntax provided in the UIM/X 3.0 Developer's Guide. The syntax must adhere to this format:

  A method is called by its method macro, which when Corba 2.0 support (the default) is chosen, takes the form:

  InterfaceName_MethodName(swidget, other_args, Environment *)

  where:

  InterfaceName_MethodName
  : The name of the method as assigned in the Method Editor

  swidget
  : The first argument, always the top-level swidget of the interface to which the method is applied.

  other_args
  : The user-defined arguments.

  Environment *
  : A pointer to an Environment structure (requirement for CORBA compliance). The type and location of the Environment structure depends on the CORBA support.

To most easily facilitate these changes, execute the following steps:

- Backup your project to a temporary directory.
- Edit each interface (.i) file using your favorite editor, changing references of:

  ```
  #ifdef __cplusplus
  ```

  to:

  ```
  #ifndef DESIGN_TIME
  ```

- Load your project into UIM/X 3.0.
- For each reference of "#ifndef DESIGN_TIME":
  - Make any necessary changes to the code within the #ifndef (method calling syntax, etc.)
  - Delete the reference to "#ifndef DESIGN_TIME" or move it to enclose only code which still needs to be protected.

- Delete the matching reference to "#endif" if the "#ifndef DESIGN_TIME" was deleted.
- Apply the change in the editor you just changed.
- Save your project.

Using Global Swidgets

In UIM/X, by default a C++ wrapper class is defined for each of the Motif widgets. Motif objects within an interface are declared as objects of these classes, rather than as swidgets, providing C++ bindings for these objects. This may cause problems with external swidget declarations from projects built with previous releases of UIM/X, because the compiler will expect a class declaration, rather than a swidget declaration. To solve this problem, do the following:

- Load UIM/X without C++ bindings enabled:

  ```
  uimx -xrm "Uimx3_0.UxNoUxBindings.set: true"
  ```

- Load the project.
- Replace declarations of external swidgets with a block of code that correctly describes how the swidget should be declared. For example, suppose your project includes form1 and form2, where form2 is declared global, and you want to act on form2 from form1. In the declarations for form1, locate the following line of code:

  ```
  extern swidget form2;
  ```

- ... and replace it with the following:

  ```
  #if defined(__cplusplus) && !defined(XT_CODE) && !defined(NO_UXBINDINGS)
  extern UxForm form2;
  #else
  extern swidget form2;
  #endif
  ```

- Save the project – the project can now be loaded into UIM/X with C++ bindings enabled.

Platform-Specific Migrating Issues

There are two platform-specific migration issues, both relating to migrating from CPT. The first relates to migrating from an HP platform. The second relates to migrating from Solaris.

On HP platforms, when you link a C++ library with a C program, the HP linker requires that the _main() function be called in your main(). When generating a C application, make sure your main() calls _main(), because some CPT libraries are written in C++. The provided main template will do this automatically.

Under Solaris, in CPT 2.03 we used the +d flag to turn off all in-line methods. This was a work-around for a SPARCWorks 3.0 bug. This bug has been fixed in SPARCWorks 4.1, and the work-around is no longer necessary. We have removed the +d flag in the Solaris makefile template. You can remove it from your project makefiles.