Finding files and syntax notes

NX Open requires that the application programmer references files that are included with NX. Every NX installation includes a specific set of directories that are all relative to the directory that is selected for NX installation. In this document the NX installation directory selected by the system administrator referred to as the NX install directory.

Different operating systems use different syntax to specify directory paths. For instance the location of the .NET libraries are:

For Windows: NX install directory\UGII\managed\

For non-Windows: NX install directory/ugii/managed/

This document uses Windows format to define directory paths that are relative to the installation directory.

Environment Variables

Environment variables are very useful in command line scripts for defining directory locations. NX provides a set of standard environment variables. The following are two commonly used standard NX environment variables.

UGII_BASE_DIR = NX install directory

UGII_ROOT_DIR = NX install directory\UGII\

The syntax to reference environment variables is also different for different operatiing systems. For instance, the syntax to reference UGII_BASE_DIR in a command line is:

For Windows: %UGII_BASE_DIR%

For non-Windows: $UGII_BASE_DIR

So in a command line, the path to the .NET libraries is:

For Windows: %UGII_ROOT_DIR%\managed\

For non-Windows: $UGII_ROOT_DIR/managed/

This document will use Windows format to reference environment variables.

Tools to set NX environment variables

The most convenient way to set the NX environment variables is to use the NX Command Prompt. The NX Command Prompt is started as follows:

For Windows: Start → Programs → NX→ NX Tools → Command Prompt (where <NX> depends on your specific installation and NX release).

For non-Windows: run ugmenu, and select the UGOPEN-API option. Then select Non-menu activities and the shell type.

Available Toolkits

There are many software toolkits provided for NX and other Siemens PLM Software products. *NX Open* specifically refers to the procedural APIs that are provided to work directly with the NX Object Model. Each API supports a specific programming language. A set of relatively new languages all share a common object model and thus have a *Common API.* Three other APIs have existed for many years and are collectively referred to as the legacy or *Classic APIs.*

This section will discuss the NX Open APIs. Other toolkits available for NX automation are also introduced and a select few toolkits for other Siemens PLM Software products are noted. What each toolkit does and when to use it is discussed.

This Programmer's Guide focuses on the Common APIs. The legacy Programmer's Guides and Reference manuals are still available for the Classic APIs. Complete information for the Classic APIs and the other toolkits referenced in this section can be found in the user's guides that are provided for each of the tools.

Common API

The NX architecture requires NX developers to expose new features and functions to a common object model. Using this common object model it is possible to automatically generate multiple language bindings. This mean that all languages derived from the Common API have the same set of objects, object properties and methods. Furthermore, the class hierarchy is the same for all Common APIs.

There are many advantages to this architectures over the architecture used for the Classic APIs. For instance:

- All Common API languages are equal in terms of NX capabilities. This mean you have the freedom to chose an implementation language that suits your specific needs without having to worry about missing functionality.
- New features and functions are available for automation when introduced into NX, there is no longer delay between the capability being available interactively versus programmatically.
- You now have access to the same object model that is used by NX developers.

The following language bindings are available for the Common API.

NX Open for .NET - This API uses Microsoft's .NET framework. This API makes it possible to create automation programs using any of the .NET compliant languages, including Visual Basic .NET and C#. Users can take full advantage of all the benefits provided by the .NET framework including native Windows dialog development tools and all of the capabilities of the Visual Studio Integrated Development Environment (IDE). This API is ideal if Windows is your platform of choice.

NX Open for Java - This API uses Sun's Java platform. Java provides many benefits including platform independence and a huge library of existing classes. The Java Abstract Windows Toolkit (AWT) and Swing provide tools for building platform independent dialogs. The Java Remote Method Invocation (RMI) methods provide tools to build client/server based applications. Also, free development environments are available such as Eclipse. This API is ideal if a multiple platform client/server application is being developed.

NX Open for C++ - This API provides a C++ interface to NX. This new C++ library is compatible with the Open C and Open C++ APIs. This API is ideal if you have existing C/C++ applications that you need to enhance.

Journaling

Although Journaling is not a toolkit it is introduced here because it can be used to produce automation solutions or to generate code for use by larger applications.

The Journal utility is a rapid automation tool that records, edits, and replays interactive NX sessions. Built from the Common API and based on .NET, it produces a scripted file from an interactive session of NX which can be run at a later time to replay the session. These sessions can be edited and enhanced with simple programming constructs and user interface components to produce a rapidly-generated customized program (see Journals and Applications).
Although Journal replay is currently limited to Visual Basic .NET and C#, the use can choose to record in any of the Common API languages. This technique can be used to generate example code which can then be used in larger applications.

The ability to record and playback Journals is included with every NX seat. See Journals for an introduction to working with Journals. Complete information on recording, editing and replaying Journals can be found in: Getting Started → Working with Parts → Common Tools → Journal.

Classic APIs

Before the Common API was adopted by NX three APIs were developed. These API are still maintained but they are no longer actively enhanced.

Open C - The Open C API is a direct programming interface to NX that allows users to create custom applications using the popular C programming language. It has been used by NX developers, customers, and partners to produce unique applications to augment NX or to act as completely separate utilities. Open C also provides a fully extensible data model, allowing customers to define new types of objects that can be treated just like standard NX objects and stored persistently in NX part files.

The Open C API has grown over many years and consist of over 5,000 functions. The functions are collectively know as User Functions. The applications developed with this API are often called UFUNC or UF programs. The Open C functions typically have the naming convention of: UF_<application area>_<function>. For instance, UF_MODL_create_plane().

Given the history of this API, it provides a wide range of coverage. To ensure that new applications have access to this coverage .NET and Java wrappers have been provided (see Wrappers). Note that wrappers are not required for programs written using NX Open C++ because the Open C functions and Open C++ methods may be called directly from NX Open C++ programs.
Open C++ - This API provided the first object oriented interface to NX. Written in C++, this API takes full advantage of object oriented features including inheritance, encapsulation and polymorphism. Open C++ provides complete access to its class hierarchy, allowing customers to override methods, derive their own classes, and create entirely new, persistent objects in NX. Open C++ is fully compatible with the Open C API.

NX Open GRIP - GRIP (Graphics Interactive Programming) is an intermediate scripting language for automating CAD/CAM/CAE tasks. Users can create applications to automate Numerical Control (NC) operations, create geometric and drafting objects, control system parameters, perform file management functions and modify existing geometry.

Knowledge Driven Automation

Knowledge Fusion (KF) - This API is an interpreted, object-oriented, language that is embedded in NX. KF allows you to add engineering knowledge to a task by creating rules which are the basic building blocks of the language. The language is declarative, rather than procedural, which means that the rules are executed when needed, regardless of the order. The Knowledge Fusion rule engine determines the correct rule firing sequence driven by the dependencies between the rules. Additionally, the language has the capability to access external knowledge bases such as databases or spreadsheets and to interface to other applications such as analysis and optimization packages. This API is ideal for applications that require associative, persistent objects that participate in model update. For more information see Knowledge Fusion and Knowledge Fusion Help and Best Practices.

Other NX Toolkits

In addition to the NX Open API toolkits provided above, Siemens PLM Software provides the following automation tools for NX. This document will provide introductions for NX Open working with these toolkits. Complete information for each toolkit can be found in their respective user's guides.

Block Styler (UI Styler) - The Block Styler is a visual user interface builder that makes it possible to interactively design portable NX style dialogs. It is used internally by NX developers and externally by users and third party developers. Block Styler provides a dialog builder that runs within NX. The dialog definition files produced by the builder are automatically loaded by NX and provide the necessary event callbacks for programs to handle user interactions during a running NX session. For more information see Block Styler and the Block Styler user's guide. The Block Styler is ideal if your application needs to run on multiple platforms and would benefit from an NX look and feel.

MenuScript - This tool allows end users and third party developers to create and edit NX menus. MenuScript is a text language that can be used to define custom NX menu items used to launch applications from an interactive session of NX. Menu files support custom tailoring of the main menu bar and the Quick View Popup menu. The standard NX menus can be customized to meet the requirements for a specific workflow or new menu items may be added to launch dialogs created with the Block Styler. MenuScript is available with all NX seats. For more information see Menu Items and the MenuScript user's guide.

Open User Interface Styler (UI Styler) - The UI Styler is a visual user interface builder that is used to maintain dialogs that were created before NX adopted Block based dialogs. New dialogs should be defined with the Block Styler. For more information see UI Styler and the Open User Interface Styler user's guide.

Other Siemens PLM Software Toolkits

Siemens PLM Software offers many other automation and system integration toolkits. Two toolkits are mentioned here due to the frequency they are used in conjunction with NX. The usage of these toolkits are out of scope for this manual. For more information see their respective user's guides, which are not part of the NX Help Library.

Parasolid - Parasolid is the world's leading production proven geometric modeling software, enabling users to model the industry's most complex parts and assemblies. Used as the geometry engine in hundreds of different computer aided design, manufacturing and engineering (CAD/CAM/CAE) applications, Parasolid has established an industry standard in global product

design. Parasolid is the solid modeling kernel used by NX. The Parasolid API is ideal for programs that provide interfaces to third party applications that produce Parasolid models.

Teamcenter Engineering Integration Tool Kit (ITK) - This API provides the functions and utilities used to customize Teamcenter to support your organization's specific data management needs. Teamcenter is a client-server architecture based system. Customization can be made both to the server and to the client portions. This API is ideal for applications that must directly interact with Teamcenter to automate the process of saving or retrieving product data produced by NX and other third party applications.

Source Notes: 1. NX Open General Programmer's Guide → About NX Open → Available Toolkits 2. Other Siemens PLM Software Toolkits - new content

Journals and Applications

In the Available Toolkits section the ability to record and playback a Journal was discussed. This ability provides an easy way to create automation solutions in NX without having to program. It does not however provide the ability to create complete applications without programming. For instance, a Journal will simply replay the steps that were recorded, it will not produce a user interface that lets the user change options. A good example is selection. If the selection of an object is recorded in a Journal then that same object (or one with the same name) is required to successfully replay the steps of the Journal.

This document uses the term "application" to refer to automation solutions that provide general solutions to a given problem domain. An application lets a user create objects based on general inputs from a user. The inputs may be in the form of data files or an interactive user interface. The objects created may be reports, data files, geometric models, NC programs, CAE models or just about anything imaginable.

It is possible to start with a Journal and modify the program to add general user inputs and output to produce a complete application. This topic is discussed in Turning Journals Into Applications.

It is also possible to use a Journal to create example code. This technique is very useful for using interactive NX to learn how to program using the common object model. Since Journals are recorded in the Common API language of your choice, the code produced by Journals can be copied into complete automation systems which are compiled and linked using all of the capabilities of high end application development tools.

Journals

This section provides an introduction to Journaling. A complete reference for creating, editing and replaying Journals is found in the NX Help Library: Getting Started → Working with Parts → Common Tools → Journal

Recording, Replaying and Editing a Journal

To record a Journal, choose Tools → Journal → Record. You will be prompted to specify the output file to store the Journal. Then each NX command that supports Journaling will record the Common API calls that are required to implement the NX command. The language that is recorded is a preference that is set with the menu command: Preferences → User Interface. Open the Journal tab on the **Preferences** dialog to find the language selections.

To playback a Journal, choose Tools → Journal → Play. The **Journal Manager** dialog is displayed. You can browse to and select the desired Journal. After selecting the Journal, click the **Run** button to execute the Journal. You may also define menu items and toolbar buttons to execute Journals. For more information on these topics see the sections in this manual on Executing NX Open Automation (Menu Items and Toolbars). Currently Journal playback is limited to Visual Basic .NET and C# .NET on the Windows platform.

To edit a Journal you can use any text editor or IDE. You can also use the Journal Manager to select a Journal and then click the **Edit** button to access a built in editor.

Journal Indicators

When a Journal is being recorded each NX command can display a marker that indicates if the command supports Journaling. The following table shows the Journal indicators and their meaning. Partial Journal support means that not all options of the NX command are recorded.

Menu Item Indicators	
	Full Journal Support
	Partial Journal Support
Toolbar Indicators	
	Fully Journal Support
	Partial Journal Support

If the Journal indicators are not shown when recording a Journal, they can be turned on by defining the following environment variable: UGII_JOURNAL_INDICATOR.

Journal Fundamentals

Here are just a few things to keep in mind when using Journals for automation.

1. Although you can pick the language to record in, playback is currently limited to Visual Basic .NET and C# .NET. If you need to develop with some other language then you can only use Journals to produce code to be compiled and linked with your application.
2. A Journal may not make calls to methods in other Journals. If your problem is too large and complex for the code to be managed in a single file, then you need to compile and link your Journals into an application.

3. The first time a Journal runs the .NET libraries have to load. So the first execution will take longer.
4. NX is not linked with all .NET libraries. If your Journal tries to make a call to a method not in the libraries linked with NX then you will get an error.

The following .NET libraries are supported by journaling:

- mscorlib.dll
- System.dll
- System.Windows.Forms.dll
- System.Drawing.dll

Any .NET functionality not supported in one of these libraries will not replay from a Journal. If your application requires .NET functionality not found in these libraries then you will need to compile and link your application referencing the required library. For instance, if your application needs to implement a client/server architecture you may need to link to System.Runtime.Remoting.dll.

Compiling and Linking Applications

Application development using NX Open works just like developing any other application. You simply need to compile and link your application's source code while including the required compile time and link time resource files. This section identifies all of the compile time and link time resource files provided for each language supported by the Common API. To compile and link classic APIs refer to the manuals supplied for those APIs.

The required libraries for an application will depend on how that application is intended to be used and the type of user interface that is required. There are three modes of executing an NX Open program: Interactive, Batch and Remote. The execution modes are defined in Execution Modes. How to execute in these modes are covered in detail in Executing NX Open Automation. The required libraries are also driven by the user interface requirements. If the application needs to interact with the NX user using NX provided user interface tools, such as selection, NX dialogs, NX Menus or toolbars, then the libraries for those tools will need to be referenced by the application. Applications that need to interact with the NX user using NX provided tools must execute in interactive mode.

Compiling and Linking - Language Specific Details

NX Open for C++

This section defines compile and linking of NX Open C++ applications. Classic Open C and Open C++ applications should reference the Programmer's Guides for those API.

All NX Open C++ libraries included with NX are found in: **<NX install directory>\UGOPEN**

All NX Open C++ include files (.hxx) are found in: **<NX install directory>\UGOPEN\NXOpen**

Compiling and linking tools providing by NX use <NX install directory>\UGOPEN\ as the base directory for all include files. To reference the NX Open C++ include files (.hxx), the NXOpen subdirectory should be referenced in the source. For example:

```
#include <NXOpen/Session.hxx>
```

The following table show the C++ library names and provides a description for when to use the library.

.NET Library Name	Purpose
libnxopencpp.lib	Contains base functionality that can be used in batch or interactive applications.
libnxopenuicpp.lib	Contains User Interface functionality that can only be used in interactive applications.

Note:

NX Open C++ may reference the classic Open C and Open C++ API directly. See the Programmer's Guides for those API to learn more about the their libraries.

The following table shows the types of executables that are used by .NET applications.

Executable Type	File Extension	When Used	Operating System
Dynamically Loadable Library	.dll	Interactive Applications	Windows
Library	.so/.sl	Interactive Applications	Non-Windows
Executable	.exe	Batch Applications	Windows and Non-Windows

Non WindowsC++ Compiling and Linking

NX Open C++ applications can be compiled on Non-Windows/LINUX using NX supplied tools like ufmenu or using system "make" command in conjuction with a makefile.

Using ufmenu

NX provides tools to compile and link applications on Non-Windows systems. These tools hide the details of various compile time and link time switches. They also include the correct search paths for compile time include files and link time libraries. The tools are:

Tool Name	Purpose
ufmenu	General script providing access to edit, compile and link scripts. Also, provides access to to execute NX Open applications.
ufcomp	Called by ufmenu or used directly for compiling NX Open C++ applications.
uflink	Called by ufmenu or used directly to link NX Open C++ applications.

For specific instruction on how to use each tool see: ufmenu, ufcomp and uflink.

Using -makefile

On non-Windows systems NX also provides the following example makefile.

<NX install directory>/ugopen/ufun_make_template.ksh

The template makefile can copied and customized to compile and link NX Open applications. Instructions on how to use the template file are fully explained in the commented section of the file.

The template file should first be copied to the application project folders and renamed to either "Makefile" or "makefile", which are the standard names for non-Windows makefiles. Alternatively, you could use the "-f" switch with the make command to specify the makefile name.

The template makefile was designed to be used with the make command supplied by the platform's vendor. For further information on make and file dependencies use "man make".

Windows C++ Compiling and Linking

NX Open C++ applications can be compiled and linked using NX supplied tools or by using visual studio

Command Line

To compile and link NX Open C++ application on a NX command prompt, use uflink

Using Visual Studio

Creating a NX Open C++ application is exactly same as any other application you create using Visual Studio. The following project/solutions settings ensure that correct runtime libraries are used by the NX Open application.

- If working with a Debug project, make sure that Runtime Library (In C/C++ → Code Generation) is set to *Multi-Threaded DLL (/MD)*.
- Additional Include directory correctly points to the directory location of NX Open C++ header files.
- Additional dependencies lists all the NX Open library the application should link against.

Alternatively, you can use the visual studio wizard to automatically set the appropriate project/solution settings for NX Open C++ application.

Using NX Visual Studio Wizard

NX provides visual studio wizard to build NX Open applications using visual studio. The wizard sets up the necessary options in visual studio for NX Open applications. For more information on how to install the wizard see: Visual Studio Application Wizard Setup
Once the wizard is set up, use the following steps to set up a NX Open C++ project in Visual Studio:

1. Inside Visual Studio, select File → New → Project, and select the desired language, then select the related Wizard.
2. Highlight the NX5 Open Wizard

   Name: (your desired project name)

   Location: (your work area)

   OK

   Next
3. Select "An internal application...." (Or whatever matches your project intentions)

   Next>

4. Make appropriate selections for your project.
5. Select Finish

x64 Compiling and Linking (Using Visual Studio)

Visual studio defaults to Win32 project on x64 machines also. To compile and link 64-bit applications:

1. Select Build→Configuration Manager.
2. Select x64 under *Active solution platform:*
3. If x64 is not listed, select <New...> from the option menu under *Active solution platform:*
4. In the New Solution Platform Dialog, select x64 from the option menu: Type or select the new platform

Note:

If x64 is not listed under Type or select new platform, you need to install 64-bit extensions for visual studio. Refer to visual studio installation guides.

NX Open for .NET

All .NET libraries included with NX are found in: **<NX install directory>\UGII\managed**

The following table show the library names and provides a description for when to use the library.

.NET Library Name	Purpose
NXOpen.dll	Contains base functionality that can be used in batch or interactive applications.
NXOpen.Utilities.dll	Contains utility functionality that can be used in batch or interactive applications.
NXOpenUI.dll	Contains User Interface functionality that can only be used in interactive applications.
NXOpen.UF.dll	Contains .NET wrapped user function for use in batch or interactive applications. Only use this library if the application requires access to the classic API.

The following table shows the types of executables that are used by .NET applications.

Executable Type	File Extension	When Used
Dynamically Loadable Library	.dll	Interactive Applications
Executable File	.exe	Batch Applications

Other .NET Libraries

Another library in the UGII\managed directory is ManagedLoader.dll. This library is for internal use and should not be included with any NX Open application.

Other files in the UGII\managed\ directory are: NXOpen.Utilities.xml, NXOpen.xml and NXOpenUI.xml. These files are used by Visual Studio and other development tools to provide documentation. For more information see: Browsing the Class Hierarchy

Command Line

The Microsoft .NET Framework SDK includes compilers for all of the .NET languages. For instance, vbc.exe is available to compile Visual Basic and csc.exe is available to compile C#. These compilers take .NET source files as input and produces a .dll file which can be loaded and

executed interactively by NX. The .NET compilers can also produce .exe files, which are used for batch applications.

For example, assuming the directory for vbc.exe and csc.exe is included in the active PATH and that UGII_ROOT_DIR = <NX install directory>\UGII\, the following commands could be used to create .NET interactive executables (<application>.dll) on Windows:

Creating Class Library

vbc /libpath:%UGII_ROOT_DIR%\managed /t:library /r:NXOpen.dll /r:NXOpen.Utilities.dll /r:NXOpen.UF.dll <application>.vb

csc /libpath:%UGII_ROOT_DIR%\managed /t:library /r:NXOpen.dll /r:NXOpen.Utilities.dll /r:NXOpen.UF.dll <application>.cs

Creating Executable (Batch Program)

To create a batch application executable (<application>.exe) set the /t option to "exe" instead of "library". For example:

vbc /libpath:%UGII_ROOT_DIR%\managed /t:exe /r:NXOpen.dll /r:NXOpen.Utilities.dll /r:NXOpen.UF.dll <application>.vb

csc /libpath:%UGII_ROOT_DIR%\managed /t:exe /r:NXOpen.dll /r:NXOpen.Utilities.dll /r:NXOpen.UF.dll <application>.cs

Using Visual Studio for .NET Development

Visual Studio is a very popular development environment from Microsoft. The following is a check list for using Visual Studio to develop NX Open applications. For more information on Visual Studio see the Microsoft Developer Network webpage.

- Make sure that the Microsoft .NET Framework SDK supported by your version of Visual Studio is compatible with that required by the target NX release. You can find the .NET Framework requirements for NX in NX Open System Information.
- For interactive applications (.dll executable) use the Class Library or Windows Application project templates when creating the project.
- For batch applications (.exe executable) use the Console Application template when creating a the project.
- To add the .NET libraries found in UGII\managed (described above) select the project in the Solution Explorer. Using the right mouse button select the Add References... menu item. Using the Browse tab locate the NX Open .NET libraries and add the libraries required by the application.
- By default Visual Studio will copy referenced libraries to the project directories. If copies of the NX Open libraries are not wanted then select each library from the list of project references and in the Properties window set the Copy Local property to False.
- For batch applications make sure the entry point is set to the desired method. The Entry Point property is found under project properties → Linker → Advanced.

Using NX Open Visual Studio Wizard

NX provides visual studio wizard to build NX Open applications using visual studio. The wizard sets up the necessary options in visual studio for NX Open applications. For more information on how to install the wizard see: Visual Studio Application Wizard Setup

NX Open for Java

All Java libraries included with NX are found in: *<NX install directory>\UGII*

The following table show the library names and provides a description for when to use the library.

Java Library Name	Purpose
NXOpen.jar	Contains base functionality that can be used in batch or interactive applications.
NXOpenUI.jar	Contains User Interface functionality that can only be used in interactive applications.
NXOpenUF.jar	Contains Java wrapped user function for use in batch or interactive applications. Only use this library if the application requires access to the classic API.

The following table shows the types of executables that are used by Java applications.

Executable Type	File Extension	When Used
Binary Class File	.class	Interactive or Batch Applications (during development cycle)
Java Archive	.jar	Interactive or Batch Applications (for signature and release)

Other Java Libraries

Also in the UGII directory are a set of Java libraries for remote applications: NXOpenRemote.jar, NXOpenUIRemote.jar and NXOpenUFRemote.jar. These libraries are used when starting remote client and server applications (see Executing Remote Processes).
Also in the UGII directory are a set of Java libraries that are used internally by NX that should not be included in NX Open applications: NXOpenRun.jar, NXOpenUIRun.jar and NXOpenRun.jar.

Command Line

The Java Developer Kit (JDK) includes a Java compiler *javac*. The Java compiler takes .java source files as input and produces a .class file which can be loaded and executed interactively by NX or in batch mode.

Non-Windows JAVA Compiling and Linking

Use the following command to compile a Java program on non-Windows systems:

```
javac -classpath
".;$UGII_ROOT_DIR/NXOpen.jar;$UGII_ROOT_DIR/NXOpenUF.jar;$UGII_ROOT_DIR/NXOpen
UI.jar" <application>.java
```

Windows JAVA Compiling and Linking

Assuming the directory for javac.exe is included in the active PATH and that UGII_ROOT_DIR = <NX install directory>\UGII\, the following command would be used to create the Java executable (<application>.class) on Windows:

```
javac -classpath
".;%UGII_ROOT_DIR%\NXOpen.jar;%UGII_ROOT_DIR%\NXOpenUF.jar;%UGII_ROOT_DIR%\
NXOpenUI.jar" <application>.java
```

JAR Files

The javac command will create a .class file. The class file can be used during the development cycle for testing. When the application is ready to release a .jar file will need to be signed (see Signing Process). To create a .jar file use another utility included with the JDK -- *jar*. The syntax for using jar is the same on non—Windows and Windows, for example:
jar cf <application>.jar <application>.class

See Sun's Java documentation on JAR for further information on all options available with JAR utility. NX will load and execute both .class files and .jar files. Both types of executable can also be executed in batch mode.

"For documentation on how to build a jar the following pages could be of interest.

Sun's page on Jar files.

http://java.sun.com/javase/6/docs/technotes/guides/jar/index.html

Suns page on using the Jar tool on Windows — http://java.sun.com/javase/6/docs/technotes/tools/windows/jar.html

Suns page on using the Jar tool on SUN (non-Windows) — http://java.sun.com/javase/6/docs/technotes/tools/solaris/jar.html

Sun's JAR tutorial — http://java.sun.com/docs/books/tutorial/deployment/jar/"

Using Eclipse for Java Development

Eclipse is a free development environment for Java. The following is a check list for using Eclipse to develop NX Open applications. For more information on Eclipse see www.eclipse.org.

- Remember to set the compiler preference to produce code for the Java Runtime Environment (JRE) that is compatible with the JRE shipped with the target version of NX. You can find the JRE version shipped with NX in NX Open System Information. To set the Eclipse compiler preference open the project's properties dialog, select Java Build Path and then the Libraries tab. Select the JRE System Library and then use the Edit button to select the desired JRE version.
- Add the .jar files found in the UGII directory as described above. To add libraries open the project's properties dialogs, select Java Build Path and then the Libraries tab. The Add External JARs... button will bring up a file selection dialog. Browse the .jar files in the UGII directory that are needed by your NX Open application and add them to the project.
- When exporting the .jar file for your project make sure the "Export generated class files and resources" option is set on the JAR Export dialog.

Compiling Open C API Programs
Linking Open C API Programs

Turning Journals Into Applications

Journals can be used for automation but as discussed in Journals and Applications recorded Journals do not provide a user interface. When the Journal is played back it will repeat exactly

what was recorded using the exact same named objects. This section will discuss the steps required to modify a Journal to provide a user interface which will permit the Journal to implement a general solution for a given problem. It will also discuss reasons for possibly migrating the Journal into a fully compiled and linked solution

The following table shows differences in the capability between an as recorded Journal, one that has been turned into an application and a fully compiled and linked system:

Journal (as recorded)	**Journal Application (adding UI)**	**Fully Compiled and Linked Application**
Single Source File	Single Source File	Any number of source files
Operates on named objects that match those selected during record, using parameters enter by the author	Operates on user selected objects, using parameters entered by the user at runtime	Operates on user selected objects, using parameters entered by the user at runtime
Limited to NX commands that support Journaling	May use all Common API classes supported by the .NET libraries link with NX (see Journal Fundamentals in Journals)	May use all Common API classes and any desired .NET class
Provides no ability to initialize events during NX startup	Provides no ability to initialize events during NX startup	During NX startup the application may be automatically loaded, a startup method can be defined to register event handlers to support dialogs, User Defined Objects and many other runtime options
Feature based license checking	Feature based license checking	Author license required during development cycle, signature required before release to the user base

QuickExtrude Example

The example in this section can be found in:

<NX install directory>\UGOPEN\SampleNXOpenApplications\
.NET\\QuickExtrude\QuickExtrude.vb

The QuickExtrude example starts with a Journal as recorded. The Journal selects an existing sketch and then creates a solid by extruded the sketch a set distance. The source code has been modified to ask the user for the distance and to let the user select the target sketch. The source code shows the originally Journaled commands, which have been commented out. The added user interface commands are highlighted. This example show exactly what to remove and what to add to a recorded to turn it into an application.

The readme file in the example folder explains how to determine what code was added and removed from the original journal. The code modifications are explained in more details below.

Adding a User Interface

In the QuickExtrude example two objects (extend1 and extend2) are used to set the starting and ending offset expressions from the sketch for the extrude command. The recorded code sets the starting and ending offset expressions to the values given to the interactive extrude command by the author (0.0 and 1.0). The Visual Basic commands recorded to set these values are:

extend1.SetValue("0.0")

extend2.SetValue("1.0")

To add a simple user interface these commands can be replaced by commands that display a dialog box to obtain a value from the user. For example:

Dim inputBox As NXInputBox = New NXInputBox

extend1.SetValue(inputBox.GetInputString("Set the Start Limit: ", "Extrude Starting Offset", "0.0"))

extend2.SetValue(inputBox.GetInputString("Set the End Limit: ", "Extrude Ending Offset", "1.0"))

These added command use the GetInputString method of the NXInputBox class. A dialog is displayed for each value asking the user to key in the desired offset values. To access this user interface class the NXOpenUI name space must be included.

Imports NXOpenUI

Removing Selection Stickiness

Journals record the exact events, including the selection events, that a user performs during the recording process. What gets recorded in a journal, is not the "intent" of a selection or the series of UI events the user performed to produce the final result. Instead, the journal records the actual name of the selected object and specific methods invoked for those object. When replayed a Journal will therefore only operate on the same named object. This behavior is referred to as Selection Stickiness.

For example, if a user is recording the blanking of all the datum planes in a view, what actually gets recorded is the exact name of each datum plane and the final method to blank the datum planes. Replaying this journal causes the execution of the exact action on the specific objects chosen. Selection Stickiness can sometimes cause a replay failure if the journal is executed out of context of its original recorded events. For example, if the Journal to blank datum planes is replayed in a different part file an error would result if the part file did not contain datum planes with the same names.

In the QuickExtrude example Selection Stickiness was removed by first adding a sketch selection method to the Journal file. The following is an example of code to select a sketch. This selection function can be inserted into the end of the original Journal file just before the End Module command.

To set up selection programmatically, the following steps are needed:

1. Define the selection scope
2. Define the selection mask
3. Use the *SelectObject* method on the *SelectionManager* class to select a specific object. *SelectObject* method invokes a simple dialog to allow you to select the sketch.

To access this user interface class and the selection mask constants the following name spaces must be included.

```
Imports NXOpen.UF
Imports NXOpenUI
```

Now, set up interactive selection (steps 1,2 and 3)

```
Public Function SelectSketch() As Sketch
```

```
    Dim ui As UI = ui.GetUI
    Dim message As String = "Select sketch"
    Dim title As String = "Selection"

    Dim scope As Selection.SelectionScope =
Selection.SelectionScope.WorkPart
    Dim keepHighlighted As Boolean = False
    Dim includeFeatures As Boolean = True

    Dim selectionAction As Selection.SelectionAction =
Selection.SelectionAction.ClearAndEnableSpecific
    Dim selectionMask_array(1) As Selection.MaskTriple

    With selectionMask_array(0)
            .Type = UFConstants.UF_sketch_type
            .Subtype = 0
            .SolidBodySubtype = 0
    End With

    Dim selectedObject As NXObject = Nothing
    Dim cursor As Point3d

            ui.SelectionManager.SelectObject(message, title,scope, _
                                                    selectionAction,
includeFeatures, _

    keepHighlighted,selectionMask_array,_
                                                    selectedObject,
cursor)
    Dim sketch As Sketch = CType(selectedObject, Sketch)
             If sketch Is Nothing Then
                   Return Nothing
             End If

             Return sketch

End Function
```

A call to the sketch selection method may now be added to the beginning of the Journal.

```
Dim sketch1 As Sketch = SelectSketch()

    If sketch1 Is Nothing Then
           Return
    End If
```

Replacing FindObject() calls

The body of the Journal will contain the original code used to reference a specifically named object. In this case the object selected during Journaling was "SKETCH(4)". The following lines of code are used to find the named object and cast the object to the appropriate feature type.

```
' **** Removed Code ****
```

```
'Dim sketchFeature1 As Features.SketchFeature =
CType(workPart.Features.FindObject("SKETCH(4)"),
Features.SketchFeature)
'features1(0) = sketchFeature1
```

Since we use interactive sketch selection, SelectSketch function will provide the user selected sketch object. To get the actual feature:

```
features1(0) = sketch1.Feature
```

Now, we need to add the actual curves to the extrude section. In the recorded journal, this is done using the specifically named object:

```
Dim arc1 As Arc = CType(sketch1.FindObject("Curve Arc1"), Arc
```

The selection intent during journal creation was Feature Curves, i.e. select all curves belonging to the selected feature. Thus, we need to find one curve of the sketch feature and replace the FindObject() call with the curve in the interactively selected sketch.

```
Dim geoms() As NXObject = sketch1.GetAllGeometry() 'Gets all the
geometry objects controlled by the sketch feature
Dim nXObject1 As NXObject = geoms(0) 'We just need the first curve as
selection intent will find all the other curves
```

Replace reference to *arc1* with the *nXObject1* in the *AddSection* method

```
' **** Removed Code ****
'section1.AddToSection(rules1, arc1, nullNXObject, nullNXObject,
helpPoint1, Section.Mode.Create)
'^^^^^^^^^^^^^^^^^^^^^^

' **** Added Code ****
section1.AddToSection(rules1, nXObject1, nullNXObject, nullNXObject,
helpPoint1, Section.Mode.Create)
```

Journal Replay

After making all the above changes, replay the journal. On replay, the selection dialog will ask you to select a sketch. On successful sketch selection, enter values start and end extend values in "Set Start Limit" and "Set End Limit" input boxes.

At the end of journal replay, an extrude feature gets created using the sketch you selected and the extrude extends you entered.

Deciding When to Compile and Link

The table given at the beginning of this section shows the major differences between Journal Applications and applications that are compiled and linked.

Compiled and linked applications provide full access to the Common API and the .NET framework. Compiled applications provide full access to NX events, NX dialogs and User Defined Objects. An author license is required for compiled applications and is checked when applications are loaded during the development cycle and is required to run the signing utility (see Development Cycle Considerations and Signing Process).
Journals are limited to a single source file and have access to most but not all capabilities of the Common API. Journals do not require an author license.

Both Journals and compiled application use feature based license checking at runtime.

Moving an application from a Journal to a compiled and linked application will therefore typically depend on the complexity and benefit of the application.

Development Cycle Considerations

An application development cycle is defined by the following steps:

1. Edit Source
2. Compile Source into Objects
3. Link Objects into an Executable (note some languages and development environments combine steps 2 and 3
4. Run the executable for testing purposes
5. If successful and application development is complete then proceed to step 6, otherwise repeat steps 1-4
6. Release the application to the user base

This section discusses issues to consider during the Testing Cycle (Steps 1-5) and before Releasing the application to the user base.

Testing Cycle (Steps 1 - 5)

Testing the application typically consist of starting NX which is then used to load and execute the application (see Executing NX Open Automation). If the application needs to be corrected and retested then a new executable must be created and loaded by NX. One way to do this is to restart NX and have it load the new current version of the executable. During the development cycle, which could require many testing cycles, restarting NX can be time consuming.
To streamline the testing process, NX Open provides Unload Options. By setting the unload option to "Immediately", NX will unload the application when the application terminates. This will permit you to edit, compile and link a new version of the application. You can then execute the application again without having to restart NX. NX will load and execute the new version of the application.

Release to Users (Step 6)

Before releasing an application to the user base the following steps should be taken.

1. If the application is being distributed to someone without an NX Open author license then the application must be signed to prove that it was developed with a valid author license. Descriptions of how to sign an application are found in: Signing Process.
2. If the application is being distributed to someone without an NX Open author license then the application will perform Feature Based license checking during execution. The application author must be aware of the features required by the application and the features available to the user base. The features required for each Common API method and property are given in the language specific reference manuals. More information on Feature Based license checking can be found in: Feature Based License Checking.

3. If the application has been tested with a debug version it may be beneficial to compile and link without the debug options to improve the applications performance.
4. It may be beneficial to change the unload option to "At Termination" (as described in Unload Options). This will cause NX to only load the application once, regardless of the number of times it is executed. This will increase the startup time for the application by removing the need to reload each time it is executed.

Execution Overview

This section provides an overview of the different execution modes and methods that are available for NX Open applications. Each execution method is covered in more detail it's own topic.

Execution Modes

NX Open applications may be executed in three different modes: Interactive, Batch and Remote. For more information see: Execution Modes.

Execution Methods

NX provides very flexible customization options. This requires many different customization access points which all need different sets of information to execute the target NX Open applications. The following table list all of the different methods available for executing NX Open applications. The table shows which execution mode is supported by the execution method and which NX tools must be present for the execution mode to be available. The table also shows if the method can be used to execute a Journal and/or a fully compiled and linked program.

Method	Execution Mode (interactive/batch)	Required Tool	Journal Applications	Compiled/Linked Application
Journal Manager	Interactive	Gateway	Yes	No
Interactive NX File→Execute→NX Open	Interactive	Gateway	No	Yes
New Menu Item (.men file)	Interactive	Menuscript	Yes	Yes
Existing Menu Item (.men file)	Interactive	Menuscript	No	Yes
Toolbar Button (.tbr file and interactive)	Interactive	Gateway	Yes	Yes
User Tools Dialog (.utd file)	Interactive	Gateway	No	Yes
Dialog from a Menu or Toolbar (.dlg file from UI Styler)	Interactive	UI Styler & Menuscript	No	Yes
Dialog from a Menu or Toolbar (.dlx file from Block Styler)	Interactive	Block Styler & Menuscript	No	Yes
Automatically at NX Startup	Interactive	Gateway	No	Yes
Automatically as an NX Application	Interactive	Gateway	No	Yes
User Exits	Interactive or Batch	Gateway	No	Yes
Command Line	Batch	Gateway	No	Yes
Remote Procedure Call	Interactive or Batch	Gateway & (.NET or Java)	No	Yes

Wizard defined by Process Studio (UI Styler and Journal Play steps)	Interactive	Process Studio	Yes	Yes
From ufmenu (option 4)	Batch	Gateway	No	Yes
From GRIP	Interactive or Batch	GRIP	No	Yes

The following table summarizes the tools list above as requirements for the different execution methods.

Tool	How Purchased	Purpose
Gateway	Available with any NX seat	Provides the foundation for all NX capabilities
Menuscript	Available with any NX seat	Provides a language for defining NX menu and toolbar buttons/commands
User Interface Styler (UI Styler)	Purchased as an add on module	Provides an interactive tool for defining platform independent NX dialogs
Process Studio	Purchases as an add on module	Provides an interactive tool for defining workflow processes

Note:

In all cases if a compiled/linked application is going to be executed by someone without an NX Open author licenses then the executable must be signed (see Signing Process). Signing is not required for Journals.

How NX Finds Application Files

Many execution methods require NX to automatically load different application files. This topic is discussed in How NX Finds Application Files.

Execution Modes

NX Open applications may be executed in three different modes. The following tables defines the differences between the execution modes.

Execution Mode	*Usage*
Interactive (a.k.a. internal)	In this mode the application is running as part of an NX interactive session. The NX display window, menus, toolbars and resource tabs are active. The application may present it's own set of dialogs to the user or the application may run behind the scenes. The application may perform a wide range of activities from geometry generation to design rule validation.
Batch (a.k.a. external)	In this mode the application runs without an NX interactive user interface. The application has full access to NX part models but no NX display options may be executed. Any user interface has to be provided by the application. Batch applications are typically used for

	time consuming tasks that require little human interaction.
Remote	In this mode their is a client and server application. The client and server execute as separate processes. The client and server may or may not reside on the same machine. The client or the server may work with NX in an interactive or batch mode. Communication between the client and server is via remote procedure calls (supported directly by Java and .NET) or by some other inter-process communications (e.g. COM objects, ports). Remote applications are useful when there is some sort of central data or knowledge that has to be shared by multiple sites.

The execution mode will impact the libraries that must be included when the application is linked (see Applications (Compile and Link)). The execution mode will also impact how the application is invoked (see Executing Overview).
Note that the Remote execution mode is note mutually exclusive with the other modes. A remote application is defined by a client and server process. Each of these processes my execute in any combination of an Interactive or Batch mode.

Source Notes: 1. Open C reference manual (duplicates in most other programmer's guide)

How NX Finds Applications Files

Many of the methods used to execute applications require NX to automatically find and load application files such as executables, menu files and dialog files. This topic discusses the general methods used to specify the locations of your applications to NX.

Environment Variables

NX uses the following environment variables to locate a root directory for your application.

Environment Variable Name and Default Value	Environment Variable Value
UGII_CUSTOM_DIRECTORY_FILE= <NX install directory>\UGII\MENUES\custom_dirs.dat	A full directory path to a file containing a list of root directories for all custom applications
UGII_USER_DIR= none (no default variable is defined)	A full directory path to an applications root directory

For released applications, using UGII_CUSTOM_DIRECTORY_FILE is highly recommended. It has the advantage of supporting root directories for multiple applications and it makes it easy to copy the file from release to release. While UGII_USER_DIR is useful during the development

cycle for a single application. Root directories from both the custom directory file and the UGII_USER_DIR variable are searched by NX.

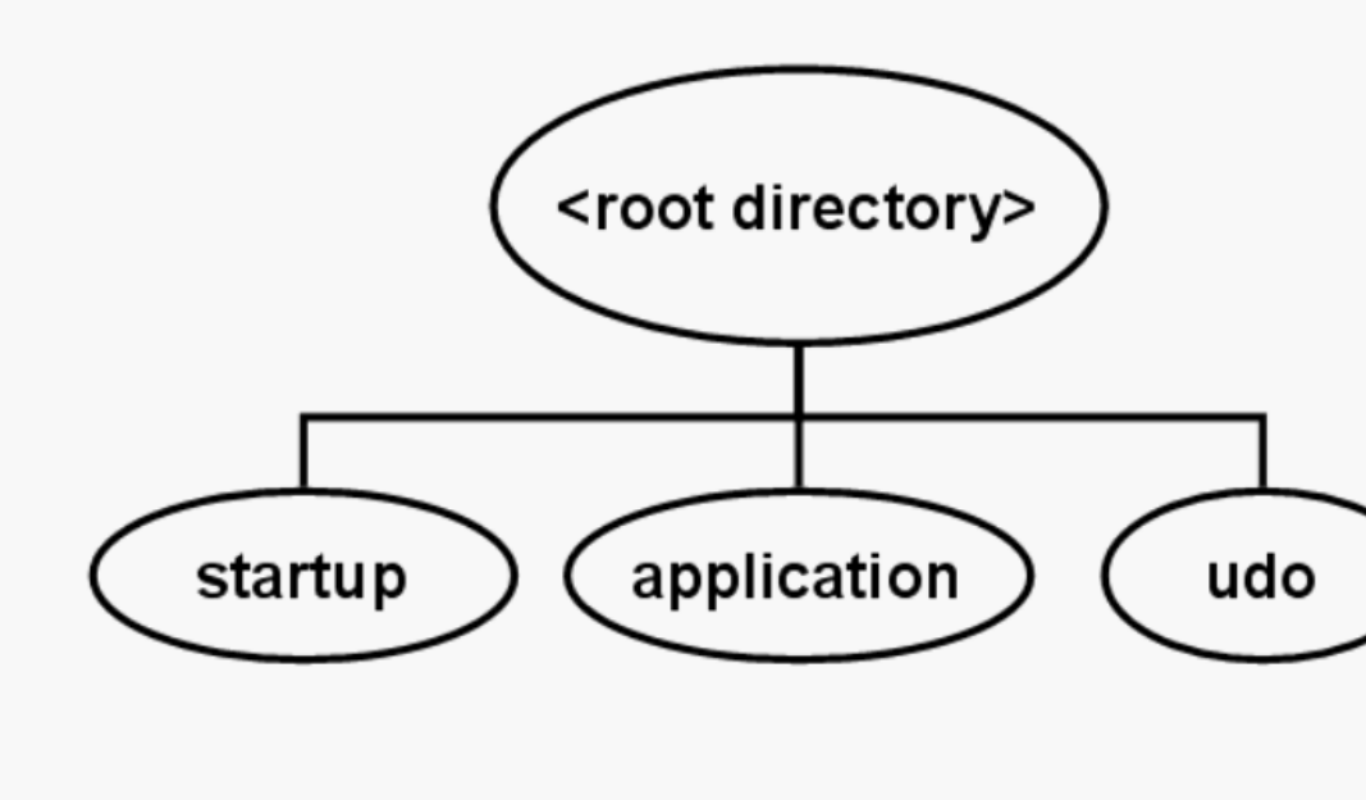
These variables are set when NX starts and are defined in: <NX install directory>\UGII\ugii_env.dat .

Note:

In your NX installation you will also find UGII_UG_CUSTOM_DIRECTORY_FILE and ug_custom_dirs.dat. This environment variable and directory file are for applications release with NX. Do not modify this variable or file.

Application Root Directory

For any root directory specified by the above environment variables, NX will look for a "startup", "application" and "udo" subdirectory.



Each subdirectory is used as follows.

Subdirectory	Usage
startup	Location for custom menu files, dialog files and executables to be loaded by NX during NX initialization. Typically used for applications that provides general functionality.
application	Location for custom menu items and executables that are associated with a new application added to the NX start menu or an existing NX

	application.
udo	Location for executables that register methods during NX initialization that are used to manage User Defined Objects (see User Defined Objects (UDO) .

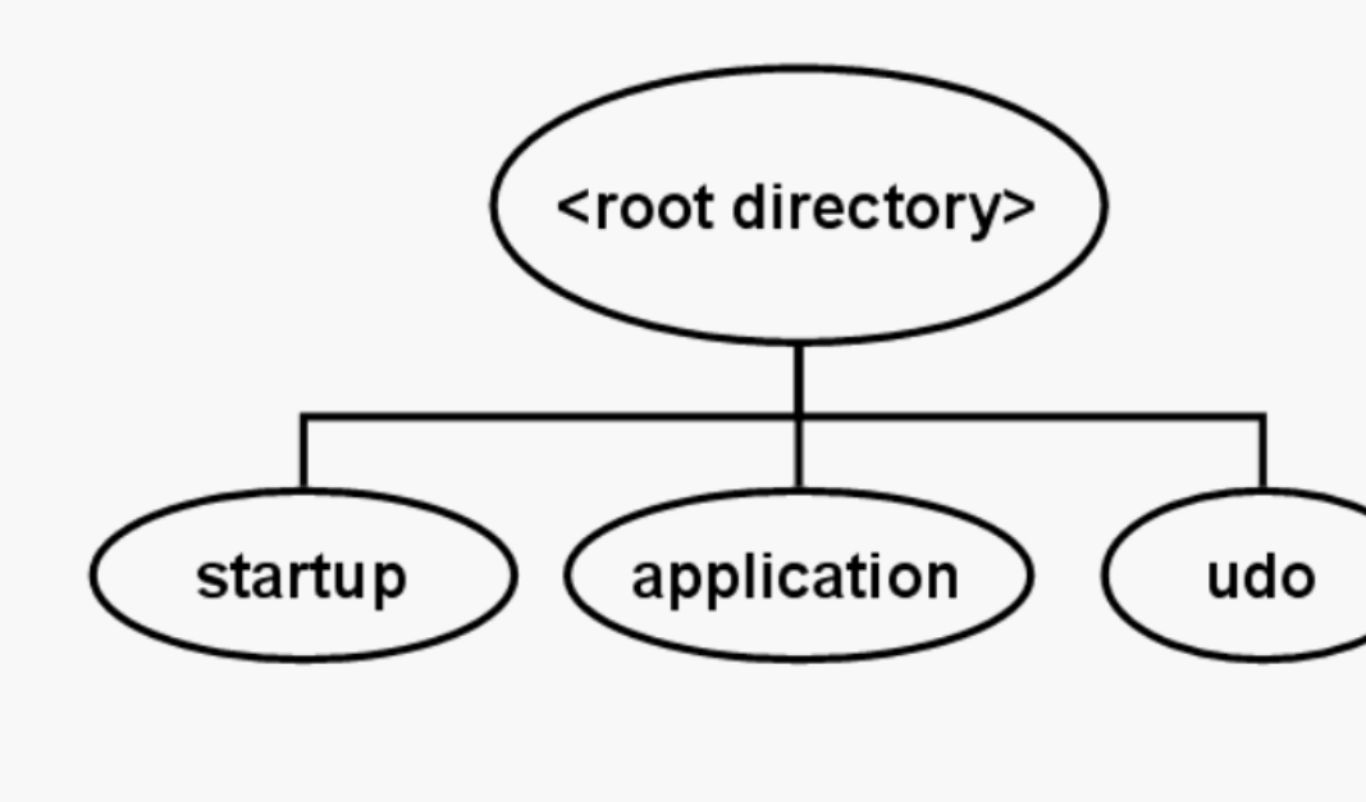
When you start NX, it automatically loads the libraries and menu files contained in the startup and udo directories. As NX loads each shared library, it immediately executes the standard entry point (see Entry Points). The application can then initialize any event handles that are required to respond to menu item, dialog and UDO actions.
To allow NX to start up faster, you can place the libraries into the application directory instead of the startup directory. When you choose this strategy, NX loads the library when a user selects the associated menu button, instead of at startup. This strategy cannot be used for applications that are managing UDO actions. UDO applications must be loaded at startup to ensure the event handlers are available to NX when parts are loaded.

Dynamically loaded shared libraries contained in these subdirectories must contain the proper file extension for the operating system (e.g. .sl, .so, .dll) . If the correct extension is not used then NX will fail to find the target file. As a result NX will display an error indicating that the application has not been properly registered.

The topic for each execution method discusses the specific files that should be copied to these subdirectories.

Application Root Directory

For any root directory specified by the above environment variables, NX will look for a "startup", "application" and "udo" subdirectory.



Each subdirectory is used as follows.

Subdirectory	Usage
startup	Location for custom menu files, dialog files and executables to be loaded by NX during NX initialization. Typically used for applications that provides general functionality.
application	Location for custom menu items and executables that are associated with a new application added to the NX start menu or an existing NX application.
udo	Location for executables that register methods during NX initialization that are used to manage User Defined Objects (see User Defined Objects (UDO) .

When you start NX, it automatically loads the libraries and menu files contained in the startup and udo directories. As NX loads each shared library, it immediately executes the standard entry point (see Entry Points). The application can then initialize any event handles that are required to respond to menu item, dialog and UDO actions.

To allow NX to start up faster, you can place the libraries into the application directory instead of the startup directory. When you choose this strategy, NX loads the library when a user selects the associated menu button, instead of at startup. This strategy cannot be used for applications that are managing UDO actions. UDO applications must be loaded at startup to ensure the event handlers are available to NX when parts are loaded.

Dynamically loaded shared libraries contained in these subdirectories must contain the proper file extension for the operating system (e.g. .sl, .so, .dll) . If the correct extension is not used then NX will fail to find the target file. As a result NX will display an error indicating that the application has not been properly registered.

The topic for each execution method discusses the specific files that should be copied to these subdirectories.

Execution Methods

Interactive NX (File → Execute → NX Open...)

A wide range of applications types can be executed from interactive NX using the menu command:

File → Execute → NX Open...

This NX command displays a file selection dialog box. You can then browse to the location of the executable file and run the application. The following types are available:

Dynamic Loadable Libraries (.dll, .sl or .so)

Executable Files (.exe)

Java Archives (.jar)

Java Class Files (.class)

Programs, Journals, or Callbacks from New Menu Items

This section contains an overview of how to execute programs, journals and callbacks from new menu items added to NX using the menu scripting language provide by NX Menuscript. For a complete discussion of NX Menuscript and how to customize the NX menus see the Menuscript User's Guide.

NX changes it's menu structure based on the active NX application (e.g. modeling, drafting, ...). This topic only discusses how to add new menus items to existing menus that are used for all applications. Adding new applications to the NX start menu, and editing application specific menus are covered in Adding Custom Applications to NX. Instructions for editing the behavior of existing menu items can be found in: Executing Programs, Journals, or Callbacks from Existing Menu Items.
This section assumes the reader is familiar with the startup folders discussed in How NX Finds Application Files.

Quick Links

Overview

- The menu scripting language provides commands to add new menu items to NX. The menu scripting commands are contained in a menu file with a *.men* suffix. In a menu file, each menu item is defined by it's location within the NX menu structure, the display text for the menu item, an ACTIONS text string (used to define the program, journal, or callback to be executed), and a button name to identify the menu item in NX. At NX startup time, NX will read all menu files in the startup directory. It will also execute all initialization entry points in all loaded applications (see Entry Points). During this initialization process any callbacks listed as actions in the menu files must be registered by the applications in the startup directory. These callbacks are also referred to as menu event handlers.

- The MenuBarManager class (or UF_MB chapter for legacy C) defines the methods used to register callbacks. The method AddMenuAction (C++, VB, or CSharp), addMenuAction (JAVA), or UF_MB_add_actions (Open C) is used to register callbacks for each menu item by referencing the ACTIONS text string found in the menu file. AddMenuAction (or it's language specific equivalent) must be called during NX startup to register the callbacks for each new menu item. When a menu item is selected NX will look up and execute the associated program, journal or callback.
- Menu files and the programs containing the callback registration methods must be located in the startup folder, while programs and journals listed as actions must be in the application folder (see How NX Finds Application Files).

Introduction to Menu Files

This section shows example menu files (.men extension) for adding new menu items. The menu files are the same regardless of which NX Open API language is being used.

To add a new menu item the following outlines the basic commands that are required in a menu file.

```
VERSION 120
EDIT UG_GATEWAY_MAIN_MENUBAR
MENU <unique text used to identify an existing or new menu item>
    <new menu item 1>
    <new menu item 2>
    <...>
END_OF_MENU
```

Each new menu item is defined with the following basic commands:

```
BUTTON  <unique text used to reference this menu item within all menu
files>
LABEL      <menu item display text>
ACTIONS <the name of a program or journal in the applications folder or
the text used by NX to associate the menu item to a callback>
```

Executing Programs

Any program may be linked to a button on the menu file. This includes applications that are unrelated to NX such as a web browser, or custom NX applications written via the Common API.

Executing programs unrelated to NX

The following sample menu file (go_to_siemens.men) adds a button below File→Open that launches the Siemens website via Internet Explorer. This menu file must be placed in the startup

directory to register the "Go to Siemens.com" button when NX is initializing its menus during startup.

Menu file: go_to_siemens.men

```
VERSION 120
EDIT UG_GATEWAY_MAIN_MENUBAR

AFTER UG_FILE_OPEN
    BUTTON SAMPLE_GO_TO_SIEMENS
    LABEL Go to Siemens.com
    ACTIONS "iexplore http://www.siemens.com"
END_OF_AFTER
```

Executing Common API Programs

If you want to run a Common API program from the button, the program must include a valid entry point: ufusr (C or C++), Main (C# or VB), or main (Java). Once compiled the program's library file must be placed in an "application" folder. If the program was written in C, C++, C# or VB, you will not need to specify the file extension for the library on the ACTIONS line from the menu file. However, you must specify the program's file extension if it is a Java class or jar file. You can use the same menu file on non-Windows systems as you use on Windows, since the library extensions aren't used and java extensions are the same on both platforms.

The following sample menu file (my_programs.men) adds buttons below File→Open that executes a custom C++ program and a custom VB program.

Menu file: my_programs.men

```
VERSION 120
EDIT UG_GATEWAY_MAIN_MENUBAR

AFTER UG_FILE_OPEN
    BUTTON SAMPLE_MY_CPP_BUTTON
    LABEL Run C++ Program
    ACTIONS my_cpp_program

    BUTTON SAMPLE_MY_VB_BUTTON
    LABEL Run VB Program
    ACTIONS my_vb_program
END_OF_AFTER
```

Note:

VB and C# are not available on non-Windows systems so the menu file should not contain any references to VB or C# programs if you intend to run on non-Windows systems.

Menu file: my_cpp_program.cpp

```
/*****************************************************************************
*******
**
** my_cpp_program.cpp
**
** Description:
** Contains Unigraphics entry points for the application.
```

```
**
************************************************************************
******/
/* Include files */
#if ! defined ( __hp9000s800 ) && ! defined ( __sgi ) && ! defined (
__sun )
#    include <strstream>
#    include <iostream>
    using std::ostrstream;
    using std::endl;
    using std::ends;
    using std::cerr;
#else
#    include <strstream.h>
#    include <iostream.h>
#endif
#include <uf.h>
#include (ug_session.hxx>
#include <ug_exception.hxx>
#include <uf_ui.h>
#include <uf_exit.h>
#include <ug_info_window.hxx>

static void processException( const UgException &exception );


/************************************************************************
*******
** Activation Methods
************************************************************************
******/
/* Explicit Activation
**    This entry point is used to activate the application explicitly,
as in
**    "File->Execute UG/Open->User Function..." */
extern DllExport void ufusr( char *parm, int *returnCode, int rlen )
{
    /* Initialize the API environment */
    UgSession session( true );
    try
    {
        /* TODO: Add your application code here */
                uc1601("Welcom to My Custom C++ Program", TRUE );
    }

    /* Handle errors */
    catch ( const UgException &exception )
    {
        processException( exception );
```

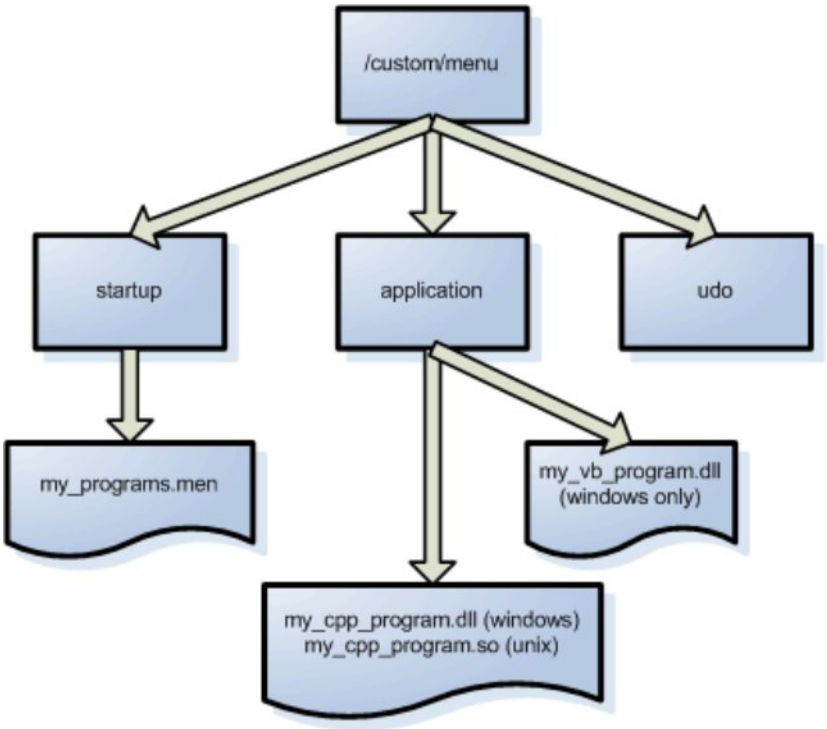
```
    }
}

/*****************************************************************
*******
** Utilities
*****************************************************************
******/

/* Unload Handler
**      This function specifies when to unload your application from
Unigraphics.
**      If your application registers a callback (from a MenuScript
item or a
**      User Defined Object for example), this function MUST return
**      "UF_UNLOAD_UG_TERMINATE". */
extern int ufusr_ask_unload( void )
{
    return( UF_UNLOAD_UG_TERMINATE );
}

/* processException
      Prints error messages to standard error and a Unigraphics
      information window. */
static void processException( const UgException &exception )
{
    /* Construct a buffer to hold the text. */
    ostrstream error_message;

    /* Initialize the buffer with the required text. */
    error_message << endl
                  << "Error:" << endl
                  << ( exception.askErrorText() ).c_str()
                  << endl << endl << ends;
    /* Open the UgInfoWindow */
    UgInfoWindow::open ( );

    /* Write the message to the UgInfoWindow. The str method */
    /* freezes the buffer, so it must be unfrozen afterwards. */
    UgInfoWindow::write( error_message.str() );

    /* Write the message to standard error */
    cerr << error_message.str();
    error_message.rdbuf()->freeze( 0 );
}
```

Menu file: my_vb_program.vb

```
Option Strict Off
Imports System
Imports NXOpen
```

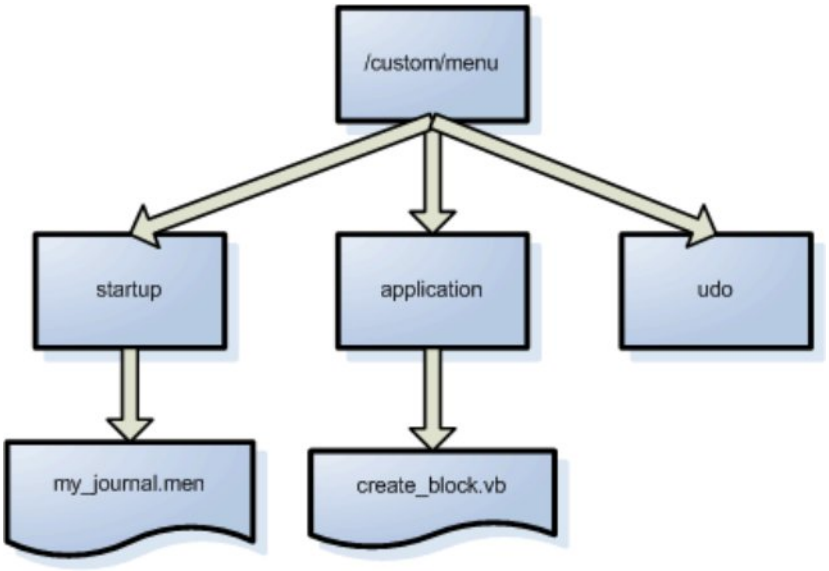
```
Module Module1
    ' Explicit Activation
    '     This entry point is used to activate the application
explicitly
    Sub Main()
        Dim theSession As Session = Session.GetSession()

        MsgBox("Welcom to My Custom VB Program")

    End Sub

    Public Function GetUnloadOption(ByVal dummy As String) As Integer

        'Unloads the image when the NX session terminates
        GetUnloadOption =
NXOpen.Session.LibraryUnloadOption.AtTermination

    End Function

End Module
```

The menu file must be placed in the "startup" folder. After compiling and linking the sample programs (my_cpp_program.cpp and my_vb_program.vb) the executables (my_cpp_program.dll, and my_vb_program.dll) need to be placed in the "application" folder.



Executing Journals

If you want to run a journal from a new menu item, the journal must be placed in an "application" folder.



The following menu file registers a button called "Create Block" below the Open button on the File menu. The journal creates a block in the current display part. If no parts are found a note is printed to the listing window stating that no block was created.

Menu file: my_journal.men

```
VERSION 120
EDIT UG_GATEWAY_MAIN_MENUBAR

AFTER UG_FILE_OPEN
   BUTTON MY_CREATE_BLOCK
   LABEL Create Block
   ACTIONS create_block.vb
END_OF_AFTER
```

Journal: create_block.vb

```
' NX 6.0.0.13
' Journal created by on Thu Nov 15 15:12:48 2007 Central Standard Time
' Edited to check for open part before creating block
'

Option Strict Off
Imports System
Imports NXOpen

Module NXJournal
Sub Main

Dim theSession As Session = Session.GetSession()
Dim workPart As Part = theSession.Parts.Work
```

```
Dim displayPart As Part = theSession.Parts.Display
Dim lw As ListingWindow = theSession.ListingWindow
lw.Open()

' ----------------------------------------------
' Don't create the block if we don't have an open part... ' -----------
----------------------------------
If displayPart Is Nothing Then

    lw.WriteLine("WARNING: Failed to find open part")
    lw.WriteLine(" Skipping block creation")
    lw.WriteLine(" ")

Else
    ' ----------------------------------------------
    ' Menu: Insert->Design Feature->Block...
    ' ----------------------------------------------
    Dim markId1 As Session.UndoMarkId
    markId1 = theSession.SetUndoMark(Session.MarkVisibility.Visible,
"Start")

    Dim nullFeatures_Feature As Features.Feature = Nothing

    Dim blockFeatureBuilder1 As Features.BlockFeatureBuilder
    blockFeatureBuilder1 = workPart.Features.CreateBlockFeatureBuilder
(nullFeatures_Feature)

    blockFeatureBuilder1.BooleanOption.Type =
GeometricUtilities.BooleanOperation.BooleanType.Create

    Dim targetBodies1(0) As Body
    Dim nullBody As Body = Nothing

    targetBodies1(0) = nullBody
    blockFeatureBuilder1.BooleanOption.SetTargetBodies(targetBodies1)

blockFeatureBuilder1.Type =
Features.BlockFeatureBuilder.Types.OriginAndEdgeLengths

    theSession.SetUndoMarkName(markId1, "Block Command")

    Dim markId2 As Session.UndoMarkId
    markId2 = theSession.SetUndoMark(Session.MarkVisibility.Invisible,
"Block")

    blockFeatureBuilder1.Type =
Features.BlockFeatureBuilder.Types.OriginAndEdgeLengths

    Dim point1 As Point
```

```
    point1 = blockFeatureBuilder1.OriginPoint

    blockFeatureBuilder1.OriginPoint = point1

    Dim originPoint1 As Point3d = New Point3d(0.0, 0.0, 0.0)
    blockFeatureBuilder1.SetOriginAndLengths(originPoint1, "100",
"100", "100")

    blockFeatureBuilder1.SetBooleanOperationAndTarget
(Features.Feature.BooleanType.Create, nullBody)

    Dim feature1 As Features.Feature
    feature1 = blockFeatureBuilder1.CommitFeature()

    theSession.DeleteUndoMark(markId2, Nothing)

    theSession.SetUndoMarkName(markId1, "Block")

    blockFeatureBuilder1.Destroy()

    lw.WriteLine("NOTE: A new block was created in the current display
part.")
    lw.WriteLine(" ")

End If ' End check for open part

' ------------------------------------------------
' Menu: Tools->Journal->Stop
' ------------------------------------------------

End Sub
End Module
```

Executing Callbacks (Menu Event Handlers)

Overview

The following examples register the same 2 callbacks "my_app_hello" and "my_app_goodbye" in different places throughout the NX menu. To run the examples, place the sample menu file in the "startup" directory. Then use the language specific details below for instructions on registering the callbacks with NX in your preferred language.

Example 1 - Adding Menu Items to the Tools Menu

For example , the following menu file would add menu items labeled "Hello" and "Goodbye" to the end of the Tools menu. The text strings used by NX to reference these menu items are

"my_app_hello" and "my_app_goodbye". See the language specific examples below for how an application would register the callbacks for the two new menu items.

Menu file: adding_menu_items_to_tools_menu.men

```
VERSION 120
EDIT UG_GATEWAY_MAIN_MENUBAR

MENU UG_TOOLBOX
    BUTTON MY_ITEM1
    LABEL Hello
    ACTIONS my_app_hello

    BUTTON MY_ITEM2
    LABEL Goodbye
    ACTIONS my_app_goodbye
END_OF_MENU
```

To find the unique text string used by NX to identify existing menus such as UG_TOOLBOX find the text displayed for the button in NX. For UG_TOOLBOX the text is displayed as "Tools" . Note that the T is underlined, which means there is an & before the T in the menu file used to define that menu item. If you go to %UGII_ROOT_DIR%/menus and search all .men files for "LABEL &Tools", you will find that the definition of that menu item in ug_main.men. Likewise, if you're looking for "Information→Part→Part History..." you should search .men files in %UGII_ROOT_DIR%/menus for "LABEL Part &History..." and again the search finds the item in ug_main.men, and the name of the "Part History..." BUTTON is UG_INFO_PART_HISTORY.

Example 2 - Adding a New Menu Pull-down

It is also possible to add a new pull-down menu. For example, the following menu file first defines a new menu with an internal name of "MY_MENU". The new menu contains the same menu items as shown in Example 1. The example then shows how to add the new menu as a pull-down to the end of the main NX menu bar with the label "My App".

The menu event handlers would be implemented exactly the same as in Example 1.

Menu file: adding_new_pulldown_menu.men

```
VERSION 120
EDIT UG_GATEWAY_MAIN_MENUBAR

MENU MY_MENU
    BUTTON MY_ITEM1
    LABEL Hello
    ACTIONS my_app_hello

    BUTTON MY_ITEM2
    LABEL Goodbye
    ACTIONS my_app_goodbye
END_OF_MENU
TOP_MENU
    CASCADE_BUTTON MY_MENU
    LABEL My App
```

```
END_OF_TOP_MENU
```

Example 3 - Adding a New Pull-down to an Existing Menu

Using commands introduced in Examples 1 and 2, it is also possible to add a new pull-down to an existing menu. The following example menu file adds the same pull-down to the end of the Tools menu. Again, menu event handlers are implemented the same as in all of the examples in this section.

Menu file: adding_new_pulldown_to_existing_menu.men

```
VERSION 120
EDIT UG_GATEWAY_MAIN_MENUBAR

MENU MY_MENU
    BUTTON MY_ITEM1
    LABEL Hello
    ACTIONS my_app_hello

    BUTTON MY_ITEM2
    LABEL Goodbye
    ACTIONS my_app_goodbye
END_OF_MENU

MENU UG_TOOLBOX
    CASCADE_BUTTON MY_MENU
    LABEL My App
END_OF_MENU
```

Example 4 - Positioning New Menu Items within an Existing Menu

The above examples position the new menu items and pull-down at the end of the existing menus. It is also possible to position new menu items within existing menus using the BEFORE and AFTER commands in place of the MENU command. The following example menu file shows how to add the same pull-down menu to the File menu just after: File → Utilities.

Menu file: positioning_new_items_withing_existing_menu.men

```
VERSION 120
EDIT UG_GATEWAY_MAIN_MENUBAR

MENU MY_MENU
    BUTTON MY_ITEM1
    LABEL Hello
    ACTIONS my_app_hello

    BUTTON MY_ITEM2
    LABEL Goodbye
    ACTIONS my_app_goodbye
END_OF_MENU

AFTER UG_FILE_UTILITIES_MENU
    CASCADE_BUTTON MY_MENU
    LABEL My App
```

```
END_OF_AFTER
```

Callbacks (Menu Event Handlers)

Callbacks are tied to menu buttons via the same ACTIONS command used for journals and programs. However, the menu file does not reference the name of the callback directly (like the name of a program or journal), it references a string that must be registered to the callback. The callbacks are registered with their associated strings in a custom application via AddMenuAction (C++, VB, or CSharp), addMenuAction (JAVA), or UF_MB_add_actions (Open C). For detailed instructions and examples on registering callbacks see the language specific details below.

Callbacks (Menu Event Handlers) - Language Specific Details

NX Open for C++
NX Open for .NET
NX Open for Java

Callback Registration for NX Open for C++

The AddMenuAction method (found in MenuBarManager) is used to register a callback with NX. The input name used in AddMenuAction must match the string used in the menu file. In the examples above the two callbacks used as actions are titled "my_app_hello" and "my_app_goodbye". The example below (hello_goodbye.h and hello_goodbye.cpp) registers two callbacks in the InitializeCallbacks method. After compiling and linking this example, the executable (hello_goodbye.dll) should be placed in the startup folder along with one of the example menu files above.

Header file: hello_goodbye.h

```
//------------------------------------------------------------------------
---------
//
// hello_goodbye.h
//
// Description:
// Contains NX entry points for the customized menu callbacks.
//
//------------------------------------------------------------------------
---------

#include <isotream>
#include <uf_def.h>
#include <uf.h>

#include <NXOpen/Session.hxx>
#include <NXOpen/MenuBar_MenuBarManager.hxx>
#include <NXOpen/MenuBar_MenuButton.hxx>
#include <NXOpen/MenuBar_MenuButtonEvent.hxx>
#include <NXOpen/UI.hxx>
```

```
#include <NXOpen/Callback.hxx>
#include <NXOpen/NXEception.hxx>

using namespace std;
using namespace NXOpen;
class HelloGoodbye;
extern HelloGoodbye *theHelloGoodbye;

class HelloGoodbye
{
    // class members
    public:
        static Session* theSession;
        static UI* theUI;
    static ListingWindow* lw;
        static int registered;

        HelloGoodbye();
    ~HelloGoodbye();

    //------------------------- Callback Prototypes --------------------
----
    MenuBar::MenuBarManager::CallbackStatus HelloCB(
MenuBar::MenuButtonEvent*
buttonEvent );
        MenuBar::MenuBarManager::CallbackStatus GoodbyeCB(
MenuBar::MenuButtonEvent* buttonEvent );

    private:
    void InitializeCallbacks();

};
```

Source file: hello_goodbye.cpp

```
//------------------------------------------------------------------------
---------
//
// hello_goodbye.cpp
//
// Description:
// Contains NX entry points for the customized menu callbacks.
//
//------------------------------------------------------------------------
---------

/* Include files */
#if ! defined ( __hp9000s800 ) && ! defined ( __sgi ) && ! defined (
__sun )
# include <strstream>
# include <iostream>
```

```
using std::ostrstream;
using std::endl;
using std::ends;
using std::cerr;
#else
# include <strstream.h>
# include <iostream.h>
#endif
#include "hello_goodbye.h"
#include <NXOpen/ListingWindow.hxx>
#include <uf_ui_types.h>
#include <uf_ui.h>
//------------------------------------------------------------------------
---------
// Initialize static variables
//------------------------------------------------------------------------
---------
Session *(HelloGoodbye::theSession) = NULL;
UI *(HelloGoodbye::theUI) = NULL;
int HelloGoodbye::registered = 0;
ListingWindow* HelloGoodbye::lw = NULL;

HelloGoodbye *theHelloGoodbye;

//------------------------------------------------------------------------
---------
// Constructor for Callback Status Test
//------------------------------------------------------------------------
---------
HelloGoodbye::HelloGoodbye()
{
    try
    {
        // Initialize the NX Open C++ API environment
              theSession = Session::GetSession();

              // Initialize the Open C API environment
        int errorCode = UF_initialize();
        if(0 != errorCode)
            throw NXOpen::NXException::Create(errorCode);

              theUI = UI::GetUI();
              lw = theSession->ListingWindow();
              InitializeCallbacks();
    }
    catch (const NXOpen::NXException& ex)
    { std::cerr << "Caught exception" << ex.Message() <<
        std::endl;
    }
```

```
    return;
}

//------------------------------------------------------------------------
---------

// Register the callbacks with NX

//------------------------------------------------------------------------
---------

void HelloGoodbye::InitializeCallbacks()
{
        try
        {
              if( registered == 0 )
              {
                    theUI->MenuBarManager()-
>AddMenuAction("my_app_hello",
make_callback(this, &HelloGoodbye::HelloCB) );
                    theUI->MenuBarManager()-
>AddMenuAction("my_app_goodbye",
make_callback(this, &HelloGoodbye::GoodbyeCB) );
                    registered = 1;
               }
        }
    catch (const NXOpen::NXException& ex)
    {
        std::cerr << "Caught exception" << ex.Message() << std::endl;
     }
    return;
}

//------------------------------------------------------------------------
---------
// Startup entrypoint for NX
//------------------------------------------------------------------------
---------
extern "C" DllExport void ufsta(char *param, int *retcod, int
param_len) {
    theHelloGoodbye = new HelloGoodbye();
    return;
}

//------------------------------------------------------------------------
---------
// Public method GetUnloadOption
// This method specifies how a shared image is unloaded from memory
// within NX.
```

```
//------------------------------------------------------------------------
---------
extern "C" DllExport int ufusr_ask_unload()
{
    return (int)Session::LibraryUnloadOptionAtTermination;
}

//------------------------------------------------------------------------
---------
// Method: UnloadLibrary()
// You have the option of coding the cleanup routine to perform any
housekeeping
// chores that may need to be performed. If you code the cleanup
routine, it is // automatically called by NX.
//------------------------------------------------------------------------
---------
extern "C" DllExport void ufusr_cleanup(void)
{
    // do your cleanup here if necessary
    return;
}

//------------------------- Callback Functions ------------------------
---------

//------------------------------------------------------------------------
---------
// Callback Name: ActionStatusTestCB
// Displays a dialog that says "Hello"
//------------------------------------------------------------------------
---------
MenuBar::MenuBarManager::CallbackStatus HelloGoodbye::HelloCB(
NXOpen::MenuBar::MenuButtonEvent* buttonEvent )
{
    if( !UF_initialize() )
    {
              uc1601("Hello", TRUE );
    }
    UF_terminate();
    return MenuBar::MenuBarManager::CallbackStatusContinue;
}

//------------------------------------------------------------------------
---------
// Callback Name: GoodbyeCB
// Displays a dialog that says "Goodbye"
//------------------------------------------------------------------------
---------
MenuBar::MenuBarManager::CallbackStatus HelloGoodbye::GoodbyeCB(
```

```
NXOpen::MenuBar::MenuButtonEvent* buttonEvent )
{
    if( !UF_initialize() )
    {
                uc1601("Goodbye", TRUE );
    }
    UF_terminate();
        return MenuBar::MenuBarManager::CallbackStatusContinue;
}
```

Callback Registration for NX Open for .NET

The AddMenuAction method (found in MenuBarManager) is used to register a callback with NX. The input name used in AddMenuAction must match the string used in the menu file. In the examples above the two callbacks used as actions are titled "my_app_hello" and "my_app_goodbye". The example below (hello_goodbye.vb) registers two callbacks in the Startup method. After compiling and linking this example, the executable (hello_goodbye.dll) should be placed in the startup folder along with one of the example menu files above.

Source file: hello_goodbye.vb

```
Option Strict Off
Imports System
Imports NXOpen
Imports NXOpen.UI

Module Module1
    ' HelloCB
    ' Opens a message box that says "Hello"
    Public Function HelloCB(ByVal buttonEvent As
NXOpen.MenuBar.MenuButtonEvent) As
NXOpen.MenuBar.MenuBarManager.CallbackStatus
        MsgBox("Hello")
        HelloCB = 0
    End Function '

    GoodbyeCB
    ' Opens a message box that says "Goodbye"
    Public Function GoodbyeCB(ByVal buttonEvent
As NXOpen.MenuBar.MenuButtonEvent) As
NXOpen.MenuBar.MenuBarManager.CallbackStatus
        MsgBox("Goodbye")
        GoodbyeCB = 0
    End Function

    ' NX Startup
    '     This entry point activates the application at NX startup
    Function Startup(ByVal args As String()) As Integer
        Dim theUI As UI = GetUI()
        Dim theSession As Session = Session.GetSession
```

```
        theUI.MenuBarManager.AddMenuAction("my_app_hello", AddressOf
HelloCB)
        theUI.MenuBarManager.AddMenuAction("my_app_goodbye", AddressOf
GoodbyeCB)
        Return 0
    End Function

    Public Function GetUnloadOption(ByVal dummy As String) As Integer
        'Unloads the image when the NX session terminates
        GetUnloadOption =
NXOpen.Session.LibraryUnloadOption.AtTermination
    End Function
End Module
```

Callback Registration for NX Open for Java

The addMenuAction method (found in menuBarManager) is used to register a callback with NX. The input name used in addMenuAction must match the string used in the menu file. In the examples above the two callbacks used as actions are titled "my_app_hello" and "my_app_goodbye". The example below (HelloGoodbye.java) registers the entire HelloGoodbye class for both actions in the initializeCallbacks method. Two handle the two separate behaviors (hello vs goodbye) the actionCallback method must determine which button invoked the callback, before calling the appropriate method (helloCB or goodbyeCB) to do the real work. After compiling and linking this example, the executable (HelloGoodbye.class) should be placed in the startup folder along with one of the example menu files above.

Compilation Instructions (windows):

1. Copy the HelloGoodbye.java file and HelloGoodbye_Makefile_win to your computer.
2. Start the NX command prompt: Go to Start → Programs. The shortcut to launch an NX command prompt is located in the same cluster where the shortcut to launch NX is located.
3. Navigate the command prompt to the folder containing HelloGoodbye.java file and HelloGoodbye_Makefile_win.
4. nmake -f Makefile_win

Compilation Instructions (non-Windows):

1. Copy the HelloGoodbye.java file and HelloGoodbye_Makefile to your computer.
2. Start the NX command prompt: Run ugmenu and select the "UGOPEN-API" option. Then select "Non-menu activites" and then the shell type.
3. Navigate the command prompt to the folder containing HelloGoodbye.java file and HelloGoodbye_Makefile.
4. make

Source file: HelloGoodbye.java

```
//--------------------------------------------------------------------------
-------
//
// hello_goodbye.java
//
//--------------------------------------------------------------------------
-------

import nxopen.*;
import nxopen.menubar.*;

// HelloGoodbye class used to demo a custom application with callbacks
in the java
language
public class HelloGoodbye implements
nxopen.menubar.MenuBarManager.ActionCallback
{
     // class members
    public static Session theSession = null;
    public static ListingWindow lw = null;
    public static UI theUI = null;
    public static int testStatus = 0;

    static HelloGoodbye theHelloGoodbye;

    // Used to tell us if we've already registered our callbacks
    public static int registered = 0;
    static int isDisposeCalled;

    // constructor
    public HelloGoodbye()
    {
        try
       {
            theSession = (Session)SessionFactory.get("Session");
            lw = theSession.listingWindow();
            theUI = (UI)SessionFactory.get("UI");
            initializeCallbacks();
        }
        catch(Exception ex)
        {
            System.out.println("Error Message");
            System.out.println(ex.getMessage());
        }
    }
    // InitializeCallbacks - registers the callbacks with NX
    private void initializeCallbacks()
    {
        try
```

```
        {
              if( registered == 0 )
              {
                 theUI.menuBarManager().addMenuAction( "my_app_hello",
this );
                 theUI.menuBarManager().addMenuAction(
"my_app_goodbye", this );
                 registered = 1;
              }
         }
         catch(Exception ex)
         {
              System.out.println("Error Message");
              System.out.println(ex.getMessage());
         }
    }

    //------------------------------------------------------
    // This entry point executes at the startup of NX.
    // Used to register the application and callbacks.
    //------------------------------------------------------
    public static void startup (String [] args)throws NXException,
java.rmi.RemoteException
    {

        try
        {
            theHelloGoodbye = new HelloGoodbye();

        }

        catch(Exception ex)
        {
        }
    }

    //------------------------------------------------------
    // getUnloadOption()
    //
    // Used to tell NX when to unload this library
    //
    // Available options include:
    //    BaseSession.LibraryUnloadOption.IMMEDIATELY
    //    BaseSession.LibraryUnloadOption.EXPLICITLY
    //    BaseSession.LibraryUnloadOption.AT_TERMINATION
    //
    // Any programs that register callbacks must use
    // AtTermination as the unload option.
    //
```

```
    //------------------------------------------------
    public static int getUnloadOption()
    {
        return BaseSession.LibraryUnloadOption.AT_TERMINATION;
    }

    //------------------------------------------------------------------
-------------
    // helloCB
    // Prints "Hello" in the listing window
    //------------------------------------------------------------------
-------------
    public nxopen.menubar.MenuBarManager.CallbackStatus helloCB(
 nxopen.menubar.MenuButtonEvent buttonEvent )
    {
         try
        {
              lw.open();
              lw.writeLine("Hello");
        }
        catch(Exception ex)
        {
              System.out.println("Error Message");
              System.out.println(ex.getMessage());
        }

        return nxopen.menubar.MenuBarManager.CallbackStatus.CONTINUE;
    }
    //------------------------------------------------------------------
-------------
    // goodbyeCB
    // Prints "Goodbye" in the listing window
    //------------------------------------------------------------------
-------------
    public nxopen.menubar.MenuBarManager.CallbackStatus goodbyeCB(
nxopen.menubar.MenuButtonEvent buttonEvent )
    {
        try
        {
              lw.open();
              lw.writeLine("Goodbye");
        }
        catch(Exception ex)
        {
              System.out.println("Error Message");
              System.out.println(ex.getMessage());
        }

        return nxopen.menubar.MenuBarManager.CallbackStatus.CONTINUE;
```

```
    }

    //------------------------------------------------------------------
-------------
    // Callback Name: actionCallback
    // This is a callback method associated with all of the 'Sample
Java' menu
    // action buttons.
    // Whenever a button is activated, we enter this function, and
determine which
    // button triggered the callback, then call the specific function
for the
    // given button.
    //------------------------------------------------------------------
-------------
    public nxopen.menubar.MenuBarManager.CallbackStatus actionCallback(
nxopen.menubar.MenuButtonEvent buttonEvent )
    {
        nxopen.menubar.MenuBarManager.CallbackStatus status =
nxopen.menubar.MenuBarManager.CallbackStatus.CONTINUE;
        try
        {
              // First we need to determine which button's action we
need to perform
              if(
buttonEvent.activeButton().buttonName().equals("MY_ITEM1") )
              {
                    status = helloCB( buttonEvent );
              } else if(
buttonEvent.activeButton().buttonName().equals("MY_ITEM2") )
              { status = goodbyeCB( buttonEvent );
              } else
              {
                    lw.open();
                    lw.writeLine(" ");
                    lw.writeLine("Inside Unknown JAVA actionCallback");

lw.writeLine("'"+buttonEvent.activeButton().buttonName()+"'");
                    lw.writeLine(" ");
              }
        }
        catch(Exception ex)
        {
              System.out.println("Error Message");
              System.out.println(ex.getMessage());
        }
        return status;
    }
}
```

Make file windows: HelloGoodbye_Makefile_win

```
NXOPENJARDIR = $(UGII_ROOT_DIR)
CLASSPATH =
".;$(NXOPENJARDIR)NXOpen.jar;$(NXOPENJARDIR)NXOpenUI.jar;(NXOPENJARDIR)
NXOpenUF.jar"

compile: HelloGoodbye.jar

HelloGoodbye.class:
    javac -classpath $(CLASSPATH) HelloGoodbye.java

HelloGoodbye.jar: HelloGoodbye.class
    echo Main-Class: HelloGoodbye> manifest.txt
    jar cmf manifest.txt HelloGoodbye.jar HelloGoodbye.class

run: HelloGoodbye.jar
    java -classpath $(CLASSPATH) HelloGoodbye

clean_all: clean
    - del *.class
    - del HelloGoodbye.jar
```

Make file non-Windows: HelloGoodbye_Makefile

```
NXOPENJARDIR=$(UGII_ROOT_DIR)
CLASSPATH=".:$(NXOPENJARDIR)/NXOpen.jar:$(NXOPENJARDIR)
/NXOpenUI.jar:$(NXOPENJARDIR)/NXOpenUF.jar"

compile: HelloGoodbye.jar

HelloGoodbye.class:
    javac -classpath $(CLASSPATH) HelloGoodbye.java

HelloGoodbye.jar: HelloGoodbye.class
    echo Main-Class: HelloGoodbye> manifest.txt
    jar cmf manifest.txt HelloGoodbye.jar HelloGoodbye.class

run: HelloGoodbye.class
    $(UGII_BASE_DIR)/ugopen/run_java -classpath $(CLASSPATH)
HelloGoodbye

clean_all: clean
    - rm *.class
```

Programs, Journals, or Callbacks from Existing Menu Items

This topic discusses methods used to customize the behavior of existing NX menu items by executing programs, journals, or callbacks (also known as menu event handlers) either before or after the normal NX function is executed. This topic builds on the discussion found in Executing

Programs, Journals, and Callbacks from New Menu Items and assumes that the reader is familiar with the basics of menu files.

Quick Links

- The ACTIONS Command
- Example Menu File with Actions
- Executing Programs
- Executing Journals
- Executing Callbacks (Menu Event Handlers)

The ACTIONS Command

- The section on adding new menu items introduced the ACTIONS menu script command and showed how the command defines a text string used by NX to identify menu items that are registered for menu event handlers. The ACTIONS command may identify one or more actions. Multiple actions are defined by listing the program names, journal names or text strings that are assigned to the callbacks and by using the REPLACE option to redefine the actions for the existing menu item. Actions for an existing menu item are then defined as follows:
- `BUTTON  <unique text used to reference this menu item within all menu files>`
- `LABEL  <menu item display text>`
- `ACTIONS/REPLACE <action 1> <action 2> ... <action n>`
- The actions are executed in order. To make one of the actions the normal NX function use the STANDARD key word. For instance, the following would redefine the menu item to execute custom actions before and after the normal NX command.
- `ACTIONS/REPLACE <pre action>  STANDARD <post action>`
- Another method of defining actions for existing menu items is to use the PRE and POST options.
- For instance, the following two sets of commands are equivalent:
- `ACTIONS/REPLACE <pre action> STANDARD`
- - is the same as -
- `ACTIONS/PRE <pre action>`
-
- `ACTIONS/REPLACE STANDARD <post action>`
- - is the same as -
- `ACTIONS/POST <post action>`

Example Menu File with Actions

The following menu file (.men extension) could be used to add pre and post actions to the NX menu item: File → Open.

Menu file: sample.men

```
VERSION 120
EDIT UG_GATEWAY_MAIN_MENUBAR

AFTER UG_FILE_NEW
  BUTTON UG_FILE_OPEN
  LABEL Open...
  ACTIONS/REPLACE my_app_hello STANDARD my_app_goodbye
END_OF_AFTER
```

In the example above, my_app_hello could be an entire program, the name of a journal file, or just a callback registered via AddMenuAction (C++, VB, or CSharp), addMenuAction (JAVA), or UF_MB_add_actions (Open C). In any case, using the above menu file would execute my_app_hello before opening the standard NX File Open dialog, and then finally running my_app_goodbye. For sample callback implementations of my_app_hello and my_app_goodbye go to the language specific details section (in your preferred language) here: Executing Programs, Journals, and Callbacks from New Menu Items

Note:

Note that the AFTER command is used to maintain the position of the Open menu item within the File menu. If the MENU command were used then the menu file would also reposition the Open menu item to the end of the File menu.

Executing Programs

Any program may be linked to a button on the menu file. This includes applications that are unrelated to NX such as a web browser, or custom NX applications written via the Common API.

Executing programs unrelated to NX

The following sample menu file (go_to_siemens.men) launches the Siemens website via Internet Explorer before displaying the Open Part dialog whenever the File→Open button is activated. This menu file must be placed in the startup directory to register the actions when NX is intialized.

Menu file: go_to_siemens.men

```
VERSION 120
EDIT UG_GATEWAY_MAIN_MENUBAR

AFTER UG_FILE_OPEN
  BUTTON SAMPLE_GO_TO_SIEMENS
  LABEL Go to Siemens.com
```

```
  ACTIONS/REPLACE "iexplore http://www.siemens.com" STANDARD
END_OF_AFTER
```

Executing Common API Programs

If you want to run a custom Common API program from the menu item, the program must include a valid entry point: ufusr (C or C++), Main (C# or VB), or main (Java). Once compiled the program's library file must be placed in an "application" folder. If the program was written in C, C++, C# or VB, you will not need to specify the file extension for the library on the ACTIONS line from the menu file. However, you must specify the program's file extension if it is a Java class or jar file. You can use the same menu file on non-Windows as you use on Windows, since the library extensions aren't used and java extensions are the same on both platforms.

The following sample menu file (my_programs.men) would invoke a custom C# program followed by a VB program before launching the standard Open File dialog. After the Open File dialog a C++ program and Java program would run. All of these programs are started simply by selecting File→Open... fom the menu bar in NX.

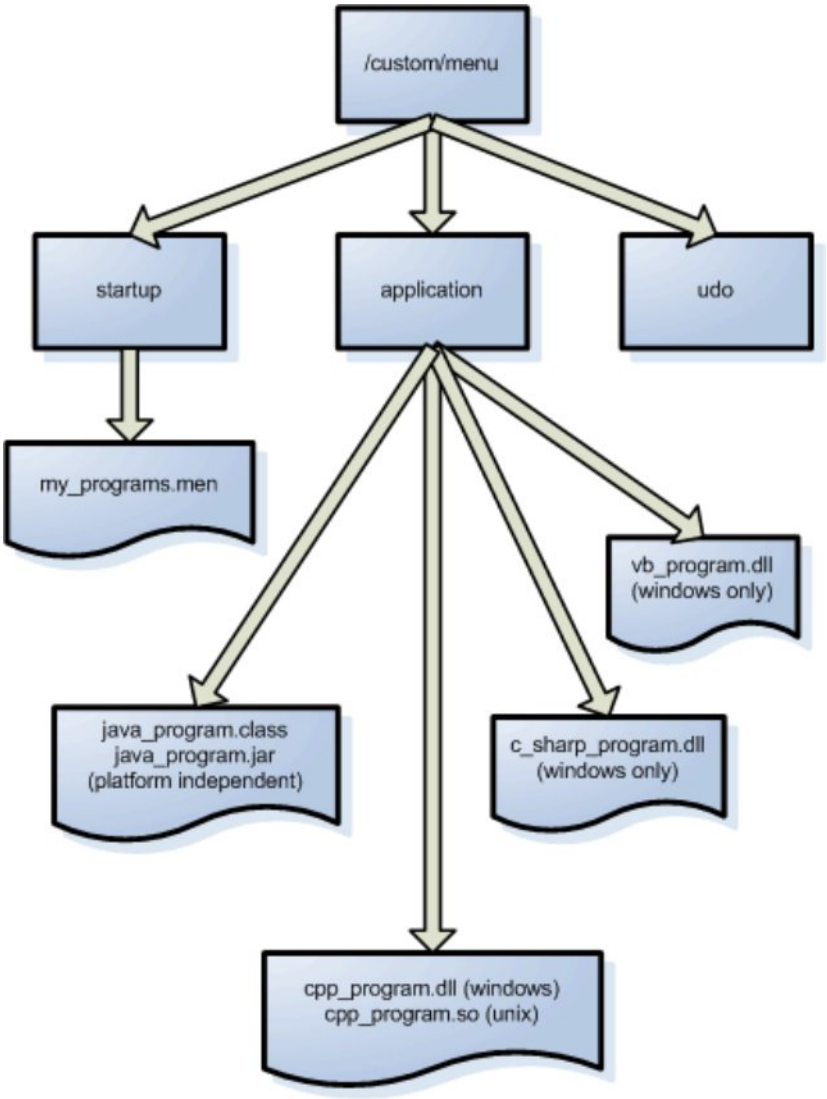
Menu file: my_programs.men

```
VERSION 120
EDIT UG_GATEWAY_MAIN_MENUBAR

AFTER UG_FILE_NEW
  BUTTON UG_FILE_OPEN
  LABEL Open...
  ACTIONS/REPLACE c_sharp_program vb_program STANDARD cpp_program
java_program
END_OF_AFTER
```

Note:

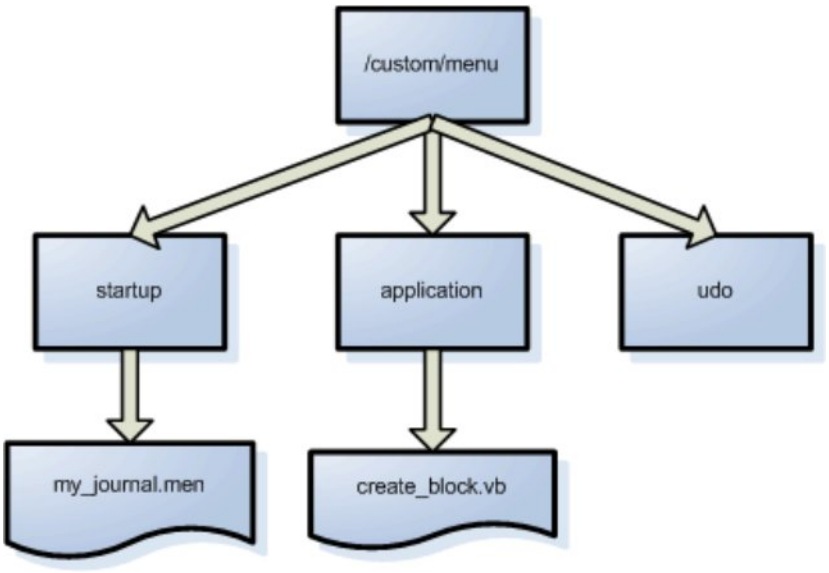
VB and C# are not available on non-Windows systems so the menu file should not contain any references to VB or C# programs if you intend to run on non-Windows.

The c_sharp_program.dll, vb_program.dll, cpp_program.dll, and java_program.class files should all be placed in the "application" folder as shown here:



Executing Journals

If you want to run a journal from the menu item, the journal must be placed in an "application" folder.



The following menu file registers a journal to run after the standard action for the UG_FILE_OPEN button. The journal creates a block in the current display part. If no parts are found (i.e. you canceled out of the open part dialog) a note is printed to the listing window stating that no block was created.

Menu file: my_journal.men

```
VERSION 120
EDIT UG_GATEWAY_MAIN_MENUBAR
AFTER UG_FILE_NEW
  BUTTON UG_FILE_OPEN
  LABEL Open...
  ACTIONS/REPLACE STANDARD create_block.vb
END_OF_AFTER
```

Journal: create_block.vb

```
' NX 6.0.0.13
' Journal created by on Thu Nov 15 15:12:48 2007 Central Standard Time
' Edited to check for open part before creating block
'
Option Strict Off
Imports System
Imports NXOpen

Module NXJournal
Sub Main

Dim theSession As Session = Session.GetSession()
Dim workPart As Part = theSession.Parts.Work

Dim displayPart As Part = theSession.Parts.Display
Dim lw As ListingWindow = theSession.ListingWindow
lw.Open()

' ------------------------------------------------
```

```
' Don't create the block if we don't have an open part...
' ----------------------------------------------
If displayPart Is Nothing Then

lw.WriteLine("WARNING: Failed to find open part")
lw.WriteLine(" Skipping block creation")
lw.WriteLine(" ")

Else
   ' ----------------------------------------------
   ' Menu: Insert->Design Feature->Block...
   ' ----------------------------------------------
   Dim markId1 As Session.UndoMarkId
   markId1 = theSession.SetUndoMark(Session.MarkVisibility.Visible,
"Start")

   Dim nullFeatures_Feature As Features.Feature = Nothing

   Dim blockFeatureBuilder1 As Features.BlockFeatureBuilder
   blockFeatureBuilder1 = workPart.Features.CreateBlockFeatureBuilder(
nullFeatures_Feature)
   blockFeatureBuilder1.BooleanOption.Type =
GeometricUtilities.BooleanOperation.BooleanType.Create

   Dim targetBodies1(0) As Body
   Dim nullBody As Body = Nothing

   targetBodies1(0) = nullBody
   blockFeatureBuilder1.BooleanOption.SetTargetBodies(targetBodies1)

   blockFeatureBuilder1.Type =
Features.BlockFeatureBuilder.Types.OriginAndEdgeLengths

   theSession.SetUndoMarkName(markId1, "Block Command")

   Dim markId2 As Session.UndoMarkId
   markId2 = theSession.SetUndoMark(Session.MarkVisibility.Invisible,
"Block")

   blockFeatureBuilder1.Type =
Features.BlockFeatureBuilder.Types.OriginAndEdgeLengths

   Dim point1 As Point
   point1 = blockFeatureBuilder1.OriginPoint

   blockFeatureBuilder1.OriginPoint = point1

   Dim originPoint1 As Point3d = New Point3d(0.0, 0.0, 0.0)
```

```
    blockFeatureBuilder1.SetOriginAndLengths(originPoint1, "100", "100",
"100")

    blockFeatureBuilder1.SetBooleanOperationAndTarget
(Features.Feature.BooleanType.Create, nullBody)

    Dim feature1 As Features.Feature
    feature1 = blockFeatureBuilder1.CommitFeature()

    theSession.DeleteUndoMark(markId2, Nothing)

    theSession.SetUndoMarkName(markId1, "Block")

    blockFeatureBuilder1.Destroy()

    lw.WriteLine("NOTE: A new block was created in the current display
part.")
    lw.WriteLine(" ")

End If ' End check for open part

' ----------------------------------------------
' Menu: Tools->Journal->Stop
' ----------------------------------------------

End Sub
End Module
```

Executing Callbacks (Menu Event Handlers)

Callbacks are tied to menu buttons via the same ACTIONS command used for journals and programs. However, there are two major differences about the execution of Callbacks.

1. The menu file does not reference the name of the callback directly (like the name of a program or journal), it references a string that must be registered to the callback. The callbacks are registered with their associated strings in a custom application via AddMenuAction (C++, VB, or CSharp), addMenuAction (JAVA), or UF_MB_add_actions (Open C).
2. Callbacks use a return value which can terminate the list of actions defined for the button.

The available return values for callbacks include:

- *Continue* - Continue performing the menu item's actions
- *Cancel* - User interaction requested inhibiting the menu item's actions
- *Override Standard* - Inhibit further actions because a pre action took the place of the standard action for a standard NX menu item
- *Warning* - Inhibit further actions because a warning condition was raised
- *Error* - Inhibit further actions because a error condition was raised

Even though the Cancel, Override Standard, Warning, and Error return values have different definitions, programmatically they all behave the same. Any return value other than Continue will prevent the rest of the actions in the list from executing. The only real difference between them is a note in the syslog stating which return value was used.

Example:

The following example displays registration of multiple callbacks for a single menu button. The status returned from the first callback, controls whether or not the other actions will execute. To run the test compile the supplied cb_status_test program and place the cb_status_test library in the startup folder, along with the sample menu file. Now launch NX and go to File→Open. A dialog will open prompting you to 'Select a return status'. If you choose Continue, the other actions will execute. If you choose any other return value, the file open dialog will not open, and my_end_action will not execute.

Menu file: cb_status_test.men

```
VERSION 120
EDIT UG_GATEWAY_MAIN_MENUBAR

AFTER UG_FILE_NEW
  BUTTON UG_FILE_OPEN
  LABEL Open...
  ACTIONS/REPLACE my_action_status_test STANDARD my_end_action
END_OF_AFTER
```

Header: cb_status_test.h

```
 //----------------------------------------------------------------------
----------
//
// cb_status_test.h
//
// Description:
// Contains NX entry points for the customized menu callbacks.
//
//----------------------------------------------------------------------
---------

#include <iostream>
#include <uf_defs.h>
#include <uf.h>

#include <NXOpen/Session.hxx>
#include <NXOpen/MenuBar_MenuBarManager.hxx>
#include <NXOpen/MenuBar_MenuButton.hxx>
#include <NXOpen/UI.hxx>
#include <NXOpen/Callback.hxx>
#include <NXOpen/NXException.hxx>

using namespace std;
using namespace NXOpen;
class CbStatusTest;
```

```
extern CbStatusTest *theCbStatusTest;

class CbStatusTest
{
    // class members
    public:
        static Session* theSession;
        static UI* theUI;
    static ListingWindow* lw;
        static int registered;

        CbStatusTest();
    ~CbStatusTest();
    //------------------------- Callback Prototypes -------------------
----
    MenuBar::MenuBarManager::CallbackStatus ActionStatusTestCB(
MenuBar::MenuButtonEvent* buttonEvent );
        MenuBar::MenuBarManager::CallbackStatus EndActionCB(
MenuBar::MenuButtonEvent* buttonEvent );

    private:
    void InitializeCallbacks();
};
```

Source: cb_status_test.cpp

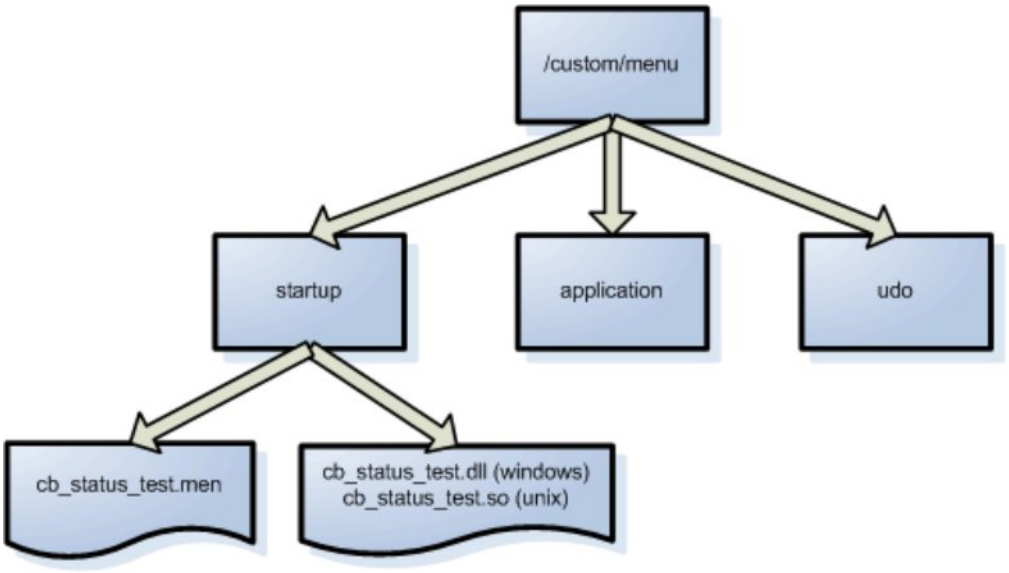
```
 //----------------------------------------------------------------------
----------
//
// cb_status_test.cpp
//
// Description:
// Contains NX entry points for the customized menu callbacks.
//
//----------------------------------------------------------------------
---------
#include "cb_status_test.h" #include
#include <NXOpen/ListingWindow.hxx>
#include  <uf_ui_types.h>
#include <uf_ui.h>

//----------------------------------------------------------------------
---------
// Initialize static variables
//----------------------------------------------------------------------
---------
Session *(CbStatusTest::theSession) = NULL;
UI *(CbStatusTest::theUI) = NULL;
int CbStatusTest::registered = 0;
ListingWindow* CbStatusTest::lw = NULL;
```

```
CbStatusTest *theCbStatusTest;
//------------------------------------------------------------------------
---------
// Constructor for Callback Status Test
//------------------------------------------------------------------------
---------
CbStatusTest::CbStatusTest()
{
    try
    {
        // Initialize the NX Open C++ API environment
                theSession = Session::GetSession();
                // Initialize the Open C API environment
        int errorCode = UF_initialize();
        if(0 != errorCode)
            throw NXOpen::NXException::Create(errorCode);

                theUI = UI::GetUI();
                lw = theSession->ListingWindow();
                InitializeCallbacks();
    }
    catch (const NXOpen::NXException& ex)
    {
        std::cerr << "Caught exception" << ex.Message() << std::endl;
    }
    return;
}

//------------------------------------------------------------------------
---------
// Register the callbacks with NX
//------------------------------------------------------------------------
---------
void CbStatusTest::InitializeCallbacks()
{
    try
    {
                if( registered == 0 )
                {
                      theUI->MenuBarManager()->AddMenuAction
("my_action_status_test", make_callback(this,
&CbStatusTest::ActionStatusTestCB) );
                       theUI->MenuBarManager()-
>AddMenuAction("my_end_action",
make_callback(this, &CbStatusTest::EndActionCB) );
                      registered = 1;
                }
        }
    catch (const NXOpen::NXException& ex)
```

```
    {
        std::cerr << "Caught exception" << ex.Message() << std::endl;
    }
    return;
}

//------------------------------------------------------------------------
---------
// Startup entrypoint for NX
//------------------------------------------------------------------------
---------
extern "C" DllExport void ufsta(char *param, int *retcod, int
param_len)
{
    theCbStatusTest = new CbStatusTest();
    return;

}  //------------------------------------------------------------------------
------------
// Public method GetUnloadOption
// This method specifies how a shared image is unloaded from memory
// within NX.
//------------------------------------------------------------------------
---------
extern "C" DllExport int ufusr_ask_unload()
{
    return (int)Session::LibraryUnloadOptionAtTermination;

}
//------------------------------------------------------------------------
---------
// Method: UnloadLibrary()
// You have the option of coding the cleanup routine to perform any
housekeeping
// chores that may need to be performed. If you code the cleanup
routine, it is
// automatically called by NX.
//------------------------------------------------------------------------
---------
extern "C" DllExport void ufusr_cleanup(void)

{
    // do your cleanup here if necessary
    return;

}
//------------------------- Callback Functions -------------------------
---------
```

```
//------------------------------------------------------------------------
---------
// Callback Name: ActionStatusTestCB
//   This is the first callback exectued when the File->Open button is
activated.
//   It will let you select the status you wish to return from this
callback.
//------------------------------------------------------------------------
---------
MenuBar::MenuBarManager::CallbackStatus
CbStatusTest::ActionStatusTestCB(
NXOpen::MenuBar::MenuButtonEvent* buttonEvent )
{
    MenuBar::MenuBarManager::CallbackStatus cb_status =
MenuBar::MenuBarManager::CallbackStatusContinue;
    if( !UF_initialize() )
    {
        char name[133] = "";
        int response = 0;
        int len = 0;
        char items[5][38];

        sprintf( items[0], "Continue" );
        sprintf( items[1], "Cancel" );
        sprintf( items[2], "Override Standard" );
        sprintf( items[3], "Warning" );
        sprintf( items[4], "Error" );

        lw->Open();
        lw->WriteLine(" ");
       lw->WriteLine("Inside Action Status Test Callback:");

       // set the default button name, to the name of the button which
activated
this event
        sprintf( name, "%s", buttonEvent->ActiveButton()-
>ButtonTypeName
().GetLocaleText() );
        UF_UI_lock_ug_access( UF_UI_FROM_CUSTOM );
        response = uc1603( "Select a return status", 0, items, 5 );
        UF_UI_unlock_ug_access( UF_UI_FROM_CUSTOM );

        if ( response == 1 || response == 5 )
        {
              // Back or Continue
              cb_status =
MenuBar::MenuBarManager::CallbackStatusContinue;
              lw->WriteLine( " Returning CallbackStatusContinue" );
        }
```

```
        else if( response == 2 || response == 6 )
        {
              // cancel
              cb_status =
MenuBar::MenuBarManager::CallbackStatusCancel;
              lw->WriteLine( " Returning CallbackStatusCancel" );
        }
        else if( response == 7 )
        {
              cb_status =
MenuBar::MenuBarManager::CallbackStatusOverrideStandard;
              lw->WriteLine( " Returning
CallbackStatusOverrideStandard" );
        }
        else if( response == 8 )
        {
              cb_status =
MenuBar::MenuBarManager::CallbackStatusWarning;
              lw->WriteLine( " Returning CallbackStatusWarning" );
        }
        else if( response == 9 )
        {
              cb_status = MenuBar::MenuBarManager::CallbackStatusError;
              lw->WriteLine( " Returning CallbackStatusError" );
        }
    }
    UF_terminate();
    return cb_status;
}
//------------------------------------------------------------------------
---------
// Callback Name: EndActionCB
// This is the last action associated with the File-Open button.
// This will only get executed if ActionStatusTestCB
// returns CallbackStatusContinue
//------------------------------------------------------------------------
---------
MenuBar::MenuBarManager::CallbackStatus CbStatusTest::EndActionCB(
NXOpen::MenuBar::MenuButtonEvent* buttonEvent )
{
        lw->Open();
    lw->WriteLine("Inside My End Action Callback");
        return MenuBar::MenuBarManager::CallbackStatusContinue;
}
```



Note:

Additional information about menuscript may be found in the overview of the UF_MB chapter of the Open C Reference Guide.

Applications from a Toolbar Button (interactive)

This topics discusses executing applications from toolbar buttons that are created interactively. To see how to execute applications from buttons defined using the toolbar files (.tbr extension) see Toolbar File

Creating Toolbar Buttons with Interactive NX

The following steps are used to define a toolbar button with NX .

Step 1 - Create New Empty Toolbar

- With the mouse cursor in the main NX menu bar area, click *MB3* and choose *Customize*.
- Click *New* on the *Customize* menu to display the toolbar properties dialog.
- Enter a name for your custom toolbar.
- Optionally select the NX applications. The toolbar will only be available when the selected NX applications are active.
- Click *OK* to create an empty toolbar.
- Perform steps 2 and 3 for each button that you want to add to the new toolbar.

Step 2 - Add Buttons to the Tool Bar

- From the *Customize* dialog, select the *Commands* tab.
- In the *Categories* list, select *New Button*.
- From the *Commands* list, select *New User Command* and drag and drop it to the new toolbar to add a button name User Command.

- Right click on the new User Command button and enter the desired name for the button.
- Right click on the new button and select the type of button you want (text, image or both).
- Right click on the new button and optionally select the image you want to display for the button.
- By right clicking on the button change any of the other desired button properties except Edit Action, which is covered in the next step.

Step 3 - Define a Buttons Action

- Right click on the button and select *Edit Action* to display the Button Action dialog.
- Select the *Type* of action and enter or browse to and select the appropriate object for the action (see the following table).
- Enter the *Button Message* text - The text shows up in button tooltip
- Click *OK* to complete the button definition.

The available button action types and the object required for the action is given in the following table.

Button Action Type	What it Does	What is Required for the Action
Journal File	Runs a Journal	A full path to the Journal file (.vb, .cs).
User Function	Runs an NX Open Application (also runs Open C/C++ applications)	A full path to the executable (.exe. .dll, .sl)
System Commands	Executes an Operating System command script	Any command that can be executed on a system shell e.g. "start notepad" (Windows) / "vi newtextfile" (non-Windows)
User Tools	Loads a UTD file	A full path to the UTD files. For further information on how to create/use UTD files, see (Gateway→Customizing NX→UTD)
Macros	Runs an NX Macro	A full path to an NX macro (.macro) file (NOTE: Macros are not a recommended for Automation)
Grip	Runs a Grip Program	A full path to a GRIP program file (.grx)

Applications from a Toolbar File

This section discusses executing applications from new buttons added to a toolbar file. You can create a toolbar in an external ASCII configuration file with a .TBR extension. For a complete discussion on toolbar files see Customizing NX in the Gateway (Gateway→Customizing NX) user guide. To execute common API programs from a toolbar:

1. Create a Toolbar File
2. Associate Toolbar Button with a Menu Button

OR

1. Add Button Actions to Toolbar Buttons
2. Load the Toolbar File

Create a Toolbar File

To create a toolbar file see (Gateway→Customizing NX→Customize dialog box overview).

Associate Toolbar Button with a Menu Button

If you have custom menu files, with button actions defined, you can create a toolbar file to have the same buttons by referencing the unique button ID of the menu buttons. The example below shows a toolbar file which has same buttons as my_programs.men (see Executing Application from New Menu Item)

Toolbar File: my_programs.tbr

```
!
!
Custom Toolbar File
!
TITLE My Custom Toolbar
VERSION 170
DOCK TOP

BUTTON SAMPLE_MY_CPP_BUTTON

BUTTON SAMPLE_MY_VB_BUTTON
```

The toolbar button will have the same action as the menu file button.

Add Button Actions to Toolbar Buttons

This section describes how common API programs can be executed from a button in a toolbar file. The reader should be familiar with toolbar file format (see Gateway→Customizing NX). To execute a NX Open program from a toolbar button, add an ACTION keyword to the the button and specify the type of action, see example.

Toolbar File: my_programs.tbr

```
!
!
Custom Toolbar File
!
TITLE My Custom Toolbar
VERSION 170
DOCK TOP

BUTTON SAMPLE_MY_CPP_BUTTON
ACTION myCppProgram.dll

BUTTON SAMPLE_MY_VB_BUTTON
ACTION myNETapp.dll
```

When the button is activated on the toolbar, NX performs the action specified by the ACTION keyword. Read How NX Finds Applications Files to see how NX finds the application associated with a specific action. Alternatively, you can also provide a full path to the NX Open application.

Load a Toolbar File

There are two ways to load a toolbar file (.TBR):

1. Interactively

   To load a .TBR file, go to Tools→Customize→Load. Select the appropriate TBR file.

2. Automatically at NX Startup

   To load a TBR file at NX startup, place the file under the startup directory of a root application directory. Read How NX Finds Application Files to see how NX loads custom files and applications at startup.

Automatically at NX Start up

NX will automatically load all application libraries saved under the startup directory (see How NX Finds Application Files) when NX is loaded. First, NX loads all internal libraries, then loads the custom libraries contained in startup folder and then immediately executes the standard entry point for startup applications. The unique entry point for applications loaded at startup is shown in table below:

C/C++	VB.NET	Java
int ufsta(void)	Function Startup ( ) As Integer	int startup (void)

The custom code in the start up entry point should be restricted to methods on session object, for e.g, part open, new part, registering UDO callbacks etc. In some situations, start up is the recommended way to load/execute the application:

1. Registering dialogs to launch for menu bar/ toolbar :- User should register their custom UI Styler or Block styler dialogs in the start up entry point in cases where customized menu bar/toolbar files reference UI styler or Block Styler dialogs (see Dialog from Menu or Toolbar)
2. Registering UDO callbacks: - User Defined Objects can have their own callbacks for display, edit, update, delete etc. These callbacks are specific to a UDO class and should be registered in the start up entry point (see User Defined Objects)
3. Registering part callbacks:- NX Open provides various callbacks on part actions (Part Open, Part Close, Part Save, Part New etc. see Part Callbacks). These callbacks are registered in start up function so NX can call the custom callback module when ever user performs part actions (see Part Callbacks)

Unloading Application

Applications loaded at start up can be terminated using normal procedure. If the custom application registers part callbacks or UDO callbacks, it must unload application only at termination (see Unload Options).

Adding Custom Applications to NX

Applications in NX are listed in the Start menu. Depending on what application is active from the start menu, the other menus may change. For example when in Gateway the menu option to Insert → Design Feature → Block is not available. However once you change to the Modeling application by selecting Modeling from the Start menu, suddenly many new items are added to the Insert menu, including Insert → Design Feature → Block. In addition to the standard NX applications, you may also add your own custom applications to the start menu. By assigning callbacks to the application, you can initialize data whenever the application is entered, run a custom program each time the application is selected from the start menu, and free application data whenever the application is exited.

This section assumes the reader is familiar with the startup and application folders discussed in How NX Finds Application Files.

Sample Applications are provided in each language to show exactly how to add a custom application in your preferred language. For the majority of the document below we will reference the C++ sample application.

The samples, which may be used as templates for your own custom applications, are located in:

- %UGII_BASE_DIR%/ugopen/SampleNXOpenApplications/C++/MenuBarCppApp
- %UGII_BASE_DIR%/ugopen/SampleNXOpenApplications/.NET/MenuBarDotNetApp
- %UGII_BASE_DIR%/ugopen/SampleNXOpenApplications/Java/MenuBarJavaApp

Adding Custom Applications to the Start Menu

To edit any of the menus in NX, you must supply a menu file (with .men extension) to tell NX about your changes. Adding a custom application requires a menu file that must be loaded at startup. Therefore the menu file must be placed in the "startup" folder.

Menu file: MenuBarCppAppButton.men

```
VERSION 120
EDIT UG_GATEWAY_MAIN_MENUBAR
MENU UG_APPLICATION
! ***NOTE button name must match the name you registered for
! your application in the MenuBarManager()->RegisterApplication call
            APPLICATION_BUTTON SAMPLE_CPP_APP
            LABEL Sample CPP Application
            LIBRARIES MenuBarCppApp
            MENU_FILES MenuBarCppApp.men
END_OF_MENU
```

The APPLICATION_BUTTON must match the name registered for the application in the call to MenuBarManager()→RegisterApplication.

The LABEL is what you see displayed in the Start menu.

The LIBRARIES command specifies the name of the library that must be executed from the "application" folder whenever this application is activated.

The MENU_FILES command specifies the name of the name of the menu file in the "application" folder that defines buttons specific to this application.

Registering the Application in NX

In addition to the menu file, the application must be registere" so that NX knows about it. The MenuBarManager class contains the RegisterApplication method. The required call to RegisterApplication must be contained in the library specified by the LIBRARIES command in the startup menu file. That library must be placed in the "application" directory.

The RegisterApplication method takes in several arguments:

1. Name - the name of the APPLICATION_BUTTON specified in the startup menu file.
2. Initialize Callback - used to initialize the application's data. Called only when the application is entered. If the custom application is already active when it's selected again from the Start menu, this callback will not execute. However if the application is exited (for example if you return to gateway), and then select the custom application from the Start menu again, the application will be re-entered and this callback will execute to initialize the data once again.
3. Enter Callback - used to run the guts of the custom application. Called every time the application is selected from the pull down menu. If the custom application is not active when selected from the pull down menu, the initialize callback will execute before this enter callback executes. If the custom application is already active when selected from the pull down menu, only the enter callback will execute.
4. Exit Callback - used to clean up the application's data. Called only when the application is exited. If the custom application is active, and then the user changes applications to something else (like Gateway) this callback will execute to free up any application specific data.
5. Supports Drawings - logical to tell NX whether or not your application supports Drawings.
6. Supports Design in Context - logical to tell NX whether or not your application supports working in Design in Context.
7. Supports Undo - logical to tell NX whether or not your application supports Undo.

For detailed examples in creating the different callbacks and registering them with the custom application in NX please refer to the following language specific examples:

- %UGII_BASE_DIR%/ugopen/SampleNXOpenApplications/C++/MenuBarCppApp/MenuBarCppApp.cpp
- %UGII_BASE_DIR%/ugopen/SampleNXOpenApplications/.NET/MenuBarDotNetApp/MenuBarCSharpApp.cs

- %UGII_BASE_DIR%/ugopen/SampleNXOpenApplications/.NET/MenuBarDotNetApp/MenuBarVbApp.vb
- %UGII_BASE_DIR%/ugopen/SampleNXOpenApplications/Java/MenuBarJavaApp/MenuBarJavaApp.java

Adding a Custom Menu to go with the Custom Application

In addition to the custom application, you can create a custom menu in NX that is only available when the custom application is active. Creating the custom menu requires a new menu file in the "application" directory. Each new menu item defined must have a callback registered to go with it via the AddActions method.

Menu file: MenuBarCppApp.men

```
VERSION 120
EDIT UG_GATEWAY_MAIN_MENUBAR
TOP_MENU
CASCADE_BUTTON SAMPLE_CPP_APP_MENU
LABEL Sample CPP
END_OF_TOP_MENU
MENU SAMPLE_CPP_APP_MENU

BUTTON SAMPLE_CPP_APP_BUTTON1
LABEL Print Button ID
ACTIONS SAMPLE_CPP_APP__action1

BUTTON SAMPLE_CPP_APP_BUTTON2
LABEL Test Callback Returns
ACTIONS SAMPLE_CPP_APP__action2

BUTTON SAMPLE_CPP_APP_BUTTON3
LABEL Print Application ID
ACTIONS SAMPLE_CPP_APP__action3

BUTTON SAMPLE_CPP_APP_BUTTON4
LABEL Print This Button Data
ACTIONS SAMPLE_CPP_APP__action4

TOGGLE_BUTTON SAMPLE_CPP_APP_BUTTON5
LABEL Print Toggle Status
ACTIONS SAMPLE_CPP_APP__action5

END_OF_MENU
```

Note:

Note the name of this menu file must match the name of the MENU_FILES specified in the startup menu file (MenuBarCppAppButton.men).

The TOP_MENU and END_OF_TOP_MENU commands indicate that the items defined between will be at the top menu level (ie at the same level as File, Edit, Information, etc.).

CASCADE_BUTTON indicates that the item is a pull down menu, and LABEL defines the display name for the cascading button.

Note:

Note the name used after CASCADE_BUTTON must match the name used after the MENU command to indicate that we are defining the menu buttons for the given cascading button.

Next each button is defined as:

BUTTON (or TOGGLE_BUTTON) followed by the name of the button. The button name is most important in java applications where it is used to determine which button triggered the callback (see the sample java application MenuBarJavaApp.java for more details).

LABEL used to define the display name of the button.

ACTIONS must match the name of the action used as input to the AddMenuAction method.

For detailed examples in creating the different action callbacks and registering them with the custom menu buttons please refer to the following language specific examples:

- %UGII_BASE_DIR%/ugopen/SampleNXOpenApplications/C++/MenuBarCppApp/MenuBarCppApp.cpp
- %UGII_BASE_DIR%/ugopen/SampleNXOpenApplications/.NET/MenuBarDotNetApp/MenuBarCSharpApp.cs
- %UGII_BASE_DIR%/ugopen/SampleNXOpenApplications/.NET/MenuBarDotNetApp/MenuBarVbApp.vb
- %UGII_BASE_DIR%/ugopen/SampleNXOpenApplications/Java/MenuBarJavaApp/MenuBarJavaApp.java

User Exits

A user exit is an optional feature that allows you to run common API programs automatically at certain predefined locations (or exits) in NX. If you go to one of these exits, NX checks to see if you have defined a pointer to the location of your common API program. If the pointer is defined, NX runs the common API program. The pointer is a an environment variable. The specifications depend on the operating system that you use and the filing system. For each User Exit, you must define an environment variable from your operating system. When NX encounters a User Exit, the system checks for the presence of a particular environment variable that points to your common API program. When the system finds your program it automatically executes the Common API program, then returns to NX.

The procedure for writing an Common API program that makes use of a User Exit is as follows:

1. Write an common API Program that performs the task you desire. All user exits are internal common API programs. Each user exit has an associated entry point name that you use in your subroutine. You write the code for the entry point subroutine using the

specified name. For example, if you decide to use the create part user exit, the associated entry point name is ufcre. See Entry Points for language specific entry points.

2. Define the pointer to the common API program. The pointer is an environment variable. See the table below for the appropriate environment variable name for the desired user exit.
3. If a User Exit uses a return value please initialize it to a valid value. See the return codes in the table below.

User Exits

Table shows all the user exits currently supported by NX.

Function	Environment Variable	Description
Open Part	USER_RETRIEVE	The open part (retrieve) user exit is invoked after the File→Open menu.. If no Common API program error is returned and an active part exists, control is returned to the current module If no active part exists, the next interactive step is determined by the return code as follows: Return Code/Description 1. Gateway menu 2. Choose part name file selection dialog
New Part	USER_CREATE	The new part user exit is invoked after the File→New menu. If no Common API program error is returned and an active part exists, control is returned to the current module. If no active part exists, the next interactive step is determined by the return code as follows: Return Code/Description 1 2 1. Gateway menu 2. Choose part name file selection dialog
Save Part	USER_FILE	The save part user exit occurs after the File→Save menu. If no Common API program error occurs and an active part exists, the next interactive step is to continue with the last main menu (Gateway menu). Return Code/Description 0 NX should go ahead and file the part 1 Gateway menu, user exit filed the part
Save Part As	USER_SAVEAS	The save part user exit occurs after the File→Save As... menu. When the mode is Design in Context and the work part to save is not the displayed part, then for each level of the

		assembly that contains the work part, the full file specification of the current part name is passed as the string parameter to the user exit. This enables you to identify which part is to be "saved as" If no Common API program error occurs and an active part exists, the next interactive step is is determined by the return code as follows: Return Code/Description 1. Gateway menu. Control passes back to the Gateway menu after going through the warnings and clean up routines of the normal NX dialogs if required. 2. 2 Choose part name file selection dialog with the string (from string parameter) as the default. For Design in Context, control passes to the normal NX dialogs for each level of the assembly above the work part (occurrence in an assembly tree) but with a default string for the new part name as specified by the string from string parameter. File→Save As dialog with no default string. n not equal to 1 or 2. For Design in Context, control passes to the normal NX dialogs for each level of the assembly above the work part (occurrence in an assembly tree).
Import Part	USER_MERGE	The import (merge) part user exit occurs after the File→Import→Part menu. If an active part exists, the next interactive step is determined by the return code as follows: Return Code/Description 0 Import Part dialog 2 Import Part dialog
Execute GRIP Program	USER_GRIP	The execute GRIP user exit occurs after the File→Execute→GRIP menu. The next interactive step is determined by the return code as follows: Return Code/Description 1. Runs the GRIP program 2. Disables the Execute GRIP option. The system adminstrator has the option of making this option (and the use of a GRIP license) unavailable.
Add Existing Part	USER_RCOMP	The add existing part (retrieve component) user exit occurs after the Assemblies→Components→Add Existing menu and before the select part dialog. The next interactive step is determined by the return code as follows: Return Code/Description

		1. Cancel current assembly operation 2. Select Part dialog 3. Component Parameters dialog n Normal operation with no default strings. "n" is any other return code except 1, 2 or 3.
Export Part	USER_FCOMP	The export part user exit occurs after the File→Export→Part menu. The next interactive step is determined by the return code as follows: Return Code/Description 1. Cancel current assembly operation 2. If the user opts to specify part from the Export Part menu, the value returned in string parameter will be used for the default part name. 3. Reserved for future use n Normal operation with no default strings. "n" is any other return code except 1, 2 or 3.
Component Where-used	USER_WHERE_USED	The component where-used user exit occurs after the Assemblies→Reports→Where Used menu and before the select components dialog. . If no Common API program error is returned and no active part exists, the file menu displays. If an active part exists, the next interactive step is determined by the return code as follows: Return Code/Description 1. Assemblies→Reports→Where Used dialog with displayed part name as default. 2. Assemblies→Reports→Where Used dialog with the string (from string parameter) as the default component name. 3. Assemblies→Reports→Where Used dialog with the string (from string parameter) as the default directory path name. n Assemblies→Reports→Where Used dialog with no default string. "n" is any other code exept 1, 2, or 3
Plot File	USER_PLOT	The plot file user exit occurs at File→Plot... menu. There is no input or output exit string. If an active part exists, the next interactive step is determined by the return code as follows: Return : Return Code/Description 1 Gateway menu n Plot dialog. "n" is any other code except 1.

2D Analysis Using Curves	USER_AREAPROPCRV	The 2D analysis using curve user exit occurs after the Info→Analysis...→Area Properties - Using Curves menu. This user exit bypasses the curve analysis routine and substitutes your user exit program. There are no return codes associated with this exit.
User Defined Symbols	USER_UDSYMBOL	The user defined symbols user exit occurs after the Application→Drafting→Create→User Defined Symbols menu. There are no return codes for this option. If the user exit exists, your routine executes and then the User Defined Symbol dialog displays. if the user exit does not exist, the User Defined Symbol dialog displays.
Open CLSF	USER_CLS_OPEN	The CLSF open user exit occurs after the Application→Manufacturing... menu. If an active part exists, the next interactive step is determined by the return code as follows: Return Code/Description 0 No CLSF returned 1 CLSF returned, awaiting acceptance 2 CLSF returned and accepted. Select File dialog with the string (from string parameter) as the default.
Save CLSF	USER_CLS_SAVE	The CLSF save exit is activated by any of the following actions: 1. File→Save→CLSF 2. File→Save→CLSF As You use this exit in succession with the USER_CLS_RENAME exit. 3. Tool Path Acceptance: Preferences→Autofile CLSF Toolbox→Operation→Generate→OK You can pass the CLSF name through the string parameter argument. If no Common API program error is returned and no active part exists, the File Main menu displays. If an active part exists, the next interactive step is determined by the return code as follows: Return Code/Description -1 User Exit Error 0 User Exit does not exist. 1 Successful User Exit execution
Rename CLSF	USER_CLS_RENAME	The CLSF rename exit occurs after Application→Manufacturing→File→Save→CLSF As. Selecting this option executes both the USER_CLS_RENAME and USER_CLS_SAVE exits in succession. You can pass the CLSF name through the string parameter argument.

		If no Common API program error is returned and no active part exists, the File Main menu displays. If an active part exists, the next interactive step is determined by the return code as follows: Return Code/Description -1 User Exit Error 0 User Exit does not exist. 1 Successful User Exit execution
Generate CLF	USER_CL_GEN	The CLF Generate exit occurs after Application→Manufacturing→Toolbox→ Tool Path.→Postprocess→Generate CLF. Selecting this option executes the USER_CL_GEN (CLF generation) exit. You can pass the CLF name through the string parameter argument. If no Common API program error is returned and no active part exists, the File Main menu displays. If an active part exists, the next interactive step is determined by the return code as follows: Return Code/Description -1 User Exit Error 0 User Exit does not exist. 1 Successful User Exit execution
Postprocess CLSF	USER_POST	The CLSF postprocess exit occurs after Application→Manufacturing→Toolbox→ Tool Path...→Postprocess→Postprocess. Selecting this option executes both the USER_CL_GEN (CLF generation) and USER_POST (CLSF postprocessing) exits in succession. You can pass the CLSF name through the string parameter argument. If no Common API program error is returned and no active part exists, the File Main menu displays. If an active part exists, the next interactive step is determined by the return code as follows: Return : Return Code/Description -1 User Exit Error 0 User Exit does not exist. 1 Successful User Exit execution
Create Component	USER_CCOMP	The create component user exit occurs after the Assemblies→Components→Create New Component→Add Object Methods menu and before the select part dialog. The next interactive step is determined by the return code

		as follows: Return Code/Description 1. Cancel current assembly operation 2. Select Part dialog with the string (from string parameter) as the default. Note: The full pathname must be specified in the string parameter argument in order for this to work. 3. Reserved for future use n Select Part dialog with no default string. "n" is any other return code except 1, 2 or 3.
Change Displayed Part	USER_CDISP	The change displayed part user exit occurs before the displayed part is about to be changed explicitly from any user interface entry point, e.g. from the Windows main menu. It is not possible to provide a default name for the operation. The next interactive step is determined by the return code as follows: . Return Code/Description 1 Cancel current assembly operation 3 Reserved for future use n Select Part dialog with no default string. "n" is any other return code except 1 or 3
Change Work Part	USER_CWORK	The change work part user exit occurs after the Assemblies→Context Control→Set Work Part before a Component is chosen or when the work part is about to be changed from any other explicit user interface entry point The next interactive step is determined by the return code as follows: Return Code/Description 1. Cancel current assembly operation 2. Select Component dialog with the string (from string parameter) as the default. 3. Reserved for future use n Normal operation with no default strings. "n" is any other return code except 1, 2 or 3.
Remove Component	USER_DCOMP	The remove component user exit occurs after Edit→Delete after a component has been selected. It is not called after a Cut operation. It is not possible to provide a default name for the operation. The next interactive step is determined by the return code as follows:

		Return Code/Description 1 Cancel current assembly operation 3 Reserved for future use n Normal operation with no default strings. "n" is any other return code except 1 or 3.
Reposition Component	USER_MCOMP	The reposition component user exit occurs after the Assemblies→Components→Reposition Component menu and after the component has been selected or when a component is about to be repositioned from any other explicit user interface entry point. It is not possible to provide a default name for the operation. The next interactive step is determined by the return code as follows: Return Code/Description 1 Cancel current assembly operation 3 Reserved for future use n Normal operation with no default strings. "n" is any other return code except 1 or 3.
Substitute Component Out	USER_SCOMP1	Substitute Component Out The substitute component out user exit occurs after the Assemblies→Components→Substitute Component menu and after the component has been selected or when a component is about to be substituted out from any other explicit user interface entry point. . It is not possible to provide a default name for the operation. The next interactive step is determined by the return code as follows: Return Code/Description 1 Cancel current assembly operation 3 Reserved for future use n Normal operation with no default strings. "n" is any other return code except 1 or 3
Substitute Component In	USER_SCOMP2	The substitute component in user exit occurs after the Assemblies→Components→Substitute Component menu. It is called before the component that is to be substituted in is selected. It is also called after MB3→Open Component As has been selected from the Assemblies Navigator. The next interactive step is determined by the return code as follows: Return Code/Description

		1. Cancel current assembly operation 2. Select Components dialog with the string (from string parameter) as the default. 3. Substitute component parameters menu with the part (from string parameter) in case of the Open As function. In case of the Substitute function it will behave as return code 1.
Open Spreadsheet	USER_SPRD_OPN	The open spreadsheet user exit occurs when you activate the spreadsheet from NX. You must be in the modeling or gateway application with an active part and you must be using a full licensed version of the spreadsheet. This event occurs interactively when a spreadsheet is activated by selecting Toolbox→Spreadsheet from menubar. Return codes are ignored with this exit.
Close Spreadsheet	USER_SPRD_CLO	The close spreadsheet user exit occurs when you exit the spreadsheet and return control to NX. You must be in the modeling application and you must be using a full licensed version of the spreadsheet. This event occurs interactively when a spreadsheet is active by selecting either File→Exit or Connections→Disconnect from the spreadsheet menubar. Return codes are ignored with this exit.
Update Spreadsheet	USER_SPRD_UPD	The update spreadsheet user exit occurs at the start of updating expressions into the NX part file. You must be in the modeling application. First, you need to call Tools→Spreadsheet and then, in the Spreadsheet menu call Tools→Extract Expr in order to have some expressions. The interactive entry point is Tools→Update Part.
Finish Updating Spreadsheet	USER_SPRD_UPF	The finish updating spreadsheet exit occurs at the completion of updating expressions. You must be in the modeling application. Return Code/Description 1 Perform a spreadsheet recalc after returning from the user exit. n No spreadsheet recalc. "n" is any other return code except 1
Replace Reference Set	USER_RRSET	The replace reference set user exit occurs after the Format→Reference Sets dialog has been invoked and the "Set Current" button has been pushed or when the Reference Set is about to be changed from any other explicit user interface entry point. It is not possible to provide a default name for the operation. The next interactive step is determined by the return code as follows: Return Code/Description

		1 Cancel current assembly operation . 3 Reserved for future use n Normal operation with no default strings. "n" is any other return code except 1 or 3
Rename Component	USER_NCOMP	The rename component user exit occurs after the Component Name has been changed on the Parameters tab on the Component Properties dialog and the user has pushed either OK or Apply. It is not possible to provide a default name for the operation. The next interactive step is determined by the return code as follows: Return Code/Description 1 Cancel current assembly operation 3 Reserved for future use n Normal operation with no default strings. "n" is any other return code except 1 or 3.
NX Startup	USER_STARTUP	The NX startup user exit occurs when you invoke NX. There are no return codes for this option. If the user exit exists, your routine executes. if the user exit does not exist, then NX starts as it normally would.
Access Genius Library Management System	USER_GENIUS	This exit accesses the Genius Library Management System. Genius is an external Siemens PLM product used by the Manufacturing Module for Tool Data Management. The Genius exit occurs after Application→Manufacturing→Toolbox→Tool→Genius. There are no return codes associated with this exit.
Execute Debug GRIP	USER_GRIPDEBUG	The execute Debug GRIP user exit occurs after the File→Execute→Debug GRIP menu. The next interactive step is determined by the return code as follows: Return Code/Description 1. Runs the Debug GRIP program using the string passed from string parameter. The Motif file dialog is not displayed. 2. Disables the Execute Debug GRIP option. The system administrator has the option of making this option (and the use of a GRIP license) unavailable.
Execute NX Open	USER_UFUNC	The Execute NX Open user exit occurs after the File→Execute→NX Open menu. The next interactive step is determined by the return code as follows: Return Code/Description

		1. Runs the Common API program using the string passed from string parameter. The Execute User Function dialog is not displayed. 2. Disables the Execute Common API option. The system administrator has the option of making this option (and the use of a User Function license) unavailable.
CAM Startup	USER_CAM_STARTUP	The CAM startup user exit occurs after the Application→Manufacturing. menu. Return Code/Description -1 User Exit Error, abort and return to Gateway 0 Successful User Exit execution, proceed normally

User Exits — Language Specific Section

NX Open for C++
NX Open for .NET

NX Open for C++

```
/* User Exit example for File-->Save.The related environment variable,
* "USER_FILE" should be defined to point to the .dll built from this
code */
#include <stdio.h>
#include <string.h>

#include <uf.h>
#include <uf_ui.h>
#include <uf_object_types.h>

#define UF_CALL(X) (report_error( __FILE__, __LINE__, #X, (X)))
static int report_error( char *file, int line, char *call, int irc)
{
    if (irc)
    {
            char err[133],
                     msg[133];

            sprintf(msg, "*** ERROR code %d at line %d in %s:\n+++ ",
                     irc, line, file);
            UF_get_fail_message(irc, err);

            UF_print_syslog(msg, FALSE);
            UF_print_syslog(err, FALSE);
            UF_print_syslog("\n", FALSE);
            UF_print_syslog(call, FALSE);
            UF_print_syslog(";\n", FALSE);

            if (!UF_UI_open_listing_window())
```

```
            {
                    UF_UI_write_listing_window(msg);
                    UF_UI_write_listing_window(err);
                    UF_UI_write_listing_window("\n");
                    UF_UI_write_listing_window(call);
                    UF_UI_write_listing_window(";\n");
            }
    }
    return(irc);
}

#define WRITE(X) UF_UI_open_listing_window();
UF_UI_write_listing_window(X)

static void do_it(void)
{
    uc1601( "Running ufput user exit program.", TRUE );
}

/*ARGSUSED*/
void ufput(char *param, int *retcode, int paramLen)
{
    if (UF_CALL(UF_initialize())) return;
    do_it();

    *retcode = 1; /* ==================== skip default behavior
============*/
    UF_terminate();
}

int ufusr_ask_unload(void)
{
    return (UF_UNLOAD_IMMEDIATELY);
}
```

NX Open for .NET

```
'
'
' The related environment variable, "USER_FILE"
' should be defined to point to the .dll built from this code
'
' Note: This will not run as a Journal, only as a .dll
'
'
Imports System
Imports System.Windows.Forms
Imports NXOpen
Imports NXOpen.uf Imports NXOpen.utilities

Module UserExit
```

```
Function ufput() As Integer
            Dim s As Session = Session.GetSession()
            MessageBox.Show("Saving: " & s.Parts.Work.FullPath)
            ufput = 0 ' set to 1 to stop the save

End Function

    Public Function GetUnloadOption(ByVal dummy As String) As Integer
            Return Session.LibraryUnloadOption.Immediately
    End Function
End Module
```

Executing Batch Applications with a Command Line

Applications can only be executed in the batch mode from a command line. This sections shows the language and platform specific details for executing batch applications.

NX Open for C++
NX Open for .NET
NX Open for Java
Java on Windows
Java on UNIX

NX Open for C++

C++ batch applications must have the normal C/C++ entry point and the linking process must produce a .exe file with execute permission. The standard entry point is:

int main(int argc, char* argv[])

An NX Open application .exe file can be executed directly from a command line as any other executable. For instance, the command to execute myApplication.exe is simply: myApplication <command line arguments>.

Note:

Note, on Windows this section only applies to unmanaged C++ applications. It does not apply to C++ .NET applications (i.e. managed C++ applications).

NX Open for .NET

NX Open for .NET batch programs are standalone executables that you can run from the operating system, outside of NX. Batch applications must be .exe files.

Typically .NET batch applications should have the following entry point:

- public static void Main(string[] args)

However, Visual Studio will allow you to set any method as the applications entry point by setting the Entry Point property found under Project Properties → Linker → Advanced. If you used visual studio for creating batch applications, make sure your project is created as an console application.

Running a Batch Application

An NX Open application .exe file can be executed directly from a command line as any other executable. Since this is a managed application, you will need to do one of the following:

- Copy the NX .NET libraries to your local working directory. To do so, copy all of the libraries from the %UGII_ROOT_DIR%\managed directory to your working directory. Use standard operating system command to execute the application.
- Copy your .EXE to UGII_ROOT_DIR\managed. Use standard operating system command to execute the application.
- Use run_managed.exe (%UGII_ROOT_DIR%\run_managed.exe)

run_managed

run_managed is a standalone executable that runs a managed NXOpen .EXE in the correct environment allowing it to pick up other DLLs from the install when they are not in the same directory as the .EXE itself.

usage:

```
run_managed <executable-file> <arguments>
```

NX Open for Java

The Java Runtime Environment (JRE) includes a utility (java.exe) to execute Java .class and .jar files. When executing a Java application you must provide the same set of NX Open Java Libraries that were provided at compile time (see Applications (Compiling and Linking)).
If the application is being executed by someone without an NX Open author license then the Java executable must be a signed .jar file (see Signing Process).

Java on Windows

On Windows use java.exe as follows:

java -classpath <NX Open Java Libraries> <your class name>.

<NX Open Java Libraries> - the same set of libraries included in the javac command when the application was compiled <your class name> - the class name for the corresponding .class or .jar file

For example, assuming UGII_ROOT_DIR is assigned to <NX install directory>\UGII\ and that the location of java.exe is included in PATH, the following command line could be used to execute a batch Java application that is using the UF wrappers:

java -classpath ".;%UGII_ROOT_DIR%\NXOpen.jar;%UGII_ROOT_DIR%\NXOpenUF.jar" <your class name>

Remote Processes

Remoting allows an NX user to execute an automation program in a separate process from the NX session. You can either connect to an NX session running in a separate process on the same machine, or via the network to an NX session running on a remote machine. Remoting in NX is based on following principles:

1. Uses Standard Framework: NX remoting makes use of remoting services provided by .NET framework and RMI for Java
2. Local and remote call transparency: Once a client application has obtained a reference to a remote session, all remote calls to the NX Open API are identical to the calls made when running in-process. Any application that runs locally can run remotely without changes, once a session reference has been obtained.

Models of Remote Access

The most important factor to consider when writing code for remote access is whether the NX user interface is running or not. The following paragraphs describe each scenario.

No User Interface

In this mode, NX runs as part of a dedicated server process, and no UI is available. The server listens for new client connections. Once it detects a client and establishes a remote session, all commands (events) come in as messages from the client process. The system executes these commands in sequence, and returns to the client after the execution of each command.

User Interface is Running

In this mode, NX runs in interactive mode with a full user interface. The system executes a small piece of automation code from NX that starts to listen for client connections. Once the system detects a client and establishes a remote session, it receives events (commands) from the client process. It also receives user interface events from the NX UI. The system sequences and executes these events according to the order of arrival. Expected use of this mode is integration between NX and stand-alone interactive programs on the same machine, such as integrating NX and a custom geometry program.

In this mode, UI events and client automation events can be mixed. There may be situations when this leads to undefined behavior. For example, NX might be currently displaying the hole dialog and the user selects a placement face for the hole. If the client process sends a command to delete the entire solid, the hole dialog is left holding onto deleted data. We expect that customers will design their integrations so this sort of interaction does not occur.

Executing Remote Processes: - Language Specific Details

NX Open for .NET
NX Open for Java

NX Open for .NET

Server Program

The following shows a sample .NET server program that uses the HTTP protocol. Compile this and run it on the server machine.

```
using System;
using System.IO;
using System.Threading;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
using System.Runtime.Remoting.Lifetime;
using System.Runtime.Remoting.Messaging;
using System.Runtime.Serialization.Formatters;

using System.Collections;

using NXOpen;

public class SimpleService
{
    public static void Main()
    {
            Thread serverThread = new Thread(new ThreadStart(Run));
            serverThread.Start();
    }

    public static void Run()
    {
            int port = 1234;

            LifetimeServices.LeaseTime =
System.TimeSpan.FromDays(10000);

            // Create a custom FormatterSinkProvider so that we can set
its security type
            // filter to Full. This is necessary for ObjectRefs to be
deserialised
```

```
            BinaryServerFormatterSinkProvider
            provider = new BinaryServerFormatterSinkProvider();
provider.TypeFilterLevel = TypeFilterLevel.Full;

            // Create the IDictionary to set the port on the channel
instance.

            IDictionary props = new Hashtable();
            props["port"] = port;

            // Create a new tcp channel with the given provider and
properties

            TcpChannel channel = new TcpChannel(props, null, provider);
            ChannelServices.RegisterChannel(channel);

            // Export the Session object
            RemotingServices.Marshal(theSession, "Session");

            Thread.Sleep(Timeout.Infinite);
    }
}
```

Client Program

The following shows a sample client program. Run the client as a standalone program.

Instead of calling Session.GetSession() the startup code might be:

```
public static void Main(String args) {
            Session theSession =
            (Session)
Activator.GetObject(typeof(Session),"http://localhost:1234/Session");
            ...
```

Once a remote session has been acquired, all other calls are identical.

This sample program uses hardcoded values. In practice, this would be specified on the command line or by using .NET configuration files.

NX Open for Java

The NX Open for Java remoting capability uses Java RMI. Remote method invocation is transparent to the client program. Once a client application has obtained a reference to a remote session, all remote calls to the NX Open API are identical to the calls made when running non-remotely. Any application that runs locally can run remotely without changes, just by changing how the NX session object is obtained. The remote server can be run either with or without the NX user interface running. The "RemotingExample" in the Java user examples directory illustrates the use of the NX Open for Java remoting capability.

To use remoting, perform the following steps:

1. **Write code for the server**

The remote server must export a remotable object to RMI from which the client can obtain the NX session object. See the following sample:

```
import java.rmi.*;
import nxopen.*;
...
public static void main(String[] args) throws Exception
{
   String host = args[0];
   System.out.println("Starting");
   theSession = (Session)SessionFactory.get("Session",
            NXRemotableObject.RemotingProtocol.create());
   System.out.println("Got Session");
   Naming.rebind("//" + host + "/NXSession", theSession);
   System.out.println("ready");
}
```

In this code, host is the internet address of the machine where rmiregistry will run.

2. **Write code for the client**

   Write the client code the same way that you would if the application were run non-remotely. Then, instead of using SessionFactory to get the NX session, use the RMI to obtain the NX session. For example, in this code fragment, remoting is used only if args is not empty.

```
public static void main(String [] args) throws Exception
{
   Session theSession = null;
   if ( args.length > 0 )
   {
           String host = args[0];
           System.out.println("Looking up name of server");
            theSession = (Session)Naming.lookup("//" + host +
"/NXSession");
   }
   else
            theSession = (Session)SessionFactory.get("Session");
   // the rest of the code written the same as it
   // would be if the application did not use remoting
```

3. **Start rmiregistry**

   Non- Windows

```
rmiregistry -J-
D$UGII_ROOT_DIR/NXOpen.jar:$UGII_ROOT_DIR/NXOpenRemote.jar:$UGII_ROO
T_DIR/NXOpenUF.jar:$UGII_ROOT_DIR/NXOpenUFRemote.jar
```

   Windows

```
rmiregistry -J-
D"%UGII_ROOT_DIR%\NXOpen.jar;%UGII_ROOT_DIR%\NXOpenRemote.jar;%UGII_
ROOT_DIR%\NXOpenUF.jar;%UGII_ROOT_DIR%\NXOpenUFRemote.jar"
```

If your server and client use other remotable interfaces than just the NX Open interfaces, add to the classpath the location of the class files for these interfaces and their rmic-generated stubs.

Note:

These instructions do not use RMI dynamic class loading. Using RMI dynamic class loading would decrease the performance of the application and is more difficult to configure. If you want to use dynamic class loading, see Sun's documentation for RMI.

4. **Start the server**

   The server can be run in batch or interactive mode. Run the server program using the same instructions given previously to run an NX Open program, except when running in batch mode, add NXOpenRemote.jar and NXOpenUFRemote.jar to your classpath.

5. **Start the client**

   Windows

   ```
   java -classpath
   ".;%UGII_ROOT_DIR%\NXOpen.jar;%UGII_ROOT_DIR%\NXOpenUF.jar;%UGII_ROOT_DIR%\NXOpenRemote.jar;%UGII_ROOT_DIR%\NXOpenUFRemote.jar" <client program name>
   ```

   Non-Windows

   ```
   java -classpath
   .:$UGII_ROOT_DIR/NXOpen.jar:$UGII_ROOT_DIR/NXOpenUF.jar:$UGII_ROOT_DIR/NXOpenRemote.jar:$UGII_ROOT_DIR/NXOpenUFRemote.jar <client program name>
   ```

   Unlike non-remote programs, it's not necessary to start the client from an NX command prompt; the NX libraries do not need to be in your library path and you do not need to set library preload. Nor do you need to set UGII_ROOT_DIR. However, if you don't set UGII_ROOT_DIR, change UGII_ROOT_DIR to the directory where the NX Open jars are located on the client machine.

Configurability and Security

Java RMI is highly configurable. You can use NXRemotableObject.RemotingProtocol to specify the port and the socket factories that the client and server will use. Using this, you can configure NX Open to use SSL sockets for remote communication. Sun Microsystem's Java RMI documentation contains information on how to use SSL with RMI and includes a sample application.

Executing Applications from GRIP

An interactive or batch application can be executed from a GRIP program using the second format of the GRIP XSPAWN command.

For instance: `XSPAWN/UFUN,'<application name>'[,IFERR,label:]`

For more details on this command see the GRIP User Guide.

UFMENU

Ufmenu is a utility script/command file that provides you with the ability to edit, compile, link and run your Open C API programs. This is only supported on non-Windows workstations.

Note:

The environment variable/logical UGII_USERFCN must be set to point to the directory where the Open C API library file(s) reside before you use the link option. If the environment variable is not set, then uflink defaults to the standard installation directory.

Ufmenu is invoked when you select the UGOPEN-API option from UGMENU. After you invoke ufmenu the User function development environment menu option displays.

```
+-------------------------------------+
|USER FUNCTION DEVELOPMENT ENVIRONMENT |
+-------------------------------------+
1) Edit 5) change Directory
2) Compile 6) liSt directory
3) Link 7) Non-menu activities
4) Run (external user function) q) Quit
Enter option (1-7, q) [q]:
```

UFMENU Options

The following paragraphs describe the options found on ufmenu. You can select options by entering the number of the option or by entering the capitalized letter in each option name as it appears in the main ufmenu. For example, to list the contents of your current directory, you can either enter option number 6 or the letter S (liSt directory).

The intent of this section of the manual is to familiarize you with the general features of the ufmenu utility which apply to all non-Windows platforms. Compile switches vary from platform to platform. We use "<path>" to indicate a directory path; in general, this is specific to your particular site..

Edit

Allows you to edit a file with the currently specified operating system editor (for example, vi). The default editor is specified by the UGII_EDITOR variable. If the file does not reside in your current directory, enter the full file specification. You can include the file extension or use a wildcard. Enter option (1-7,q) [q]: 1 Enter file(s) to edit (vi) [block1.c block2.c block3.c bounded_plane.c testopen.c]: bounded_plane.c

Compile

Invokes the C or C++ compiler which converts the statements of your C or C++ source file into an object file. You can specify a file template, such as *.c. This compiles all the files in your current directory with the appropriate file extension. .

You must include the file extension for your programs. You can compile more than one file by delimiting your file names with a space. The compile option automatically determines the appropriate compile options for your platform. A list of all the files with a ".c" extension appears within square brackets. For example:

Enter option (1-7,q) [q]: 2

Enter file(s) (separated by " ") to compile [block1.c block2.c block3.c bounded_plane.c testopen.c]: bounded_plane.c Compiling...

bounded_plane.c

Default C compile options: -c -KPIC -Xc -I. -I<path>

Change compile options (y/n) [n]: n

bounded_plane.c compiled successfully.

Hit <RETURN> to continue.

How to Change Compile Options

The default compile option switches are for both internal and external Open C API programs. If you wish to change any of the default options, enter y to the prompt: "Change compile options (y/n) [n]:".

The ufmenu script prompts you for the mode (internal or external), and which compile options to remove. It then prompts you for options to add to the compile command line. In the following example we show how to change compile options.

Enter option (1-7,q) [q]: 2

Enter file(s) (separated by " ") to compile [block1.c block2.c block3.c bounded_plane.c testopen.c]:

testopen.c block1.c block2.c block3.c

Compiling... testopen.c block1.c block2.c block3.c

Default C compile options: -c +Z -Aa -I. -I<path>

Change compile options (y/n) [n]: y

Compile internal/external user function (i/e) [i]: e

Remove +Z (y/n) [n]:

Remove -Aa (y/n) [n]:

Remove -I. (y/n) [n]:

Remove -I<path> (y/n) [n]:

Add new options: -g -I/user1/include

New compile options: -c -Z -Aa -I. -I<path> -g -I<path>

testopen.c compiled successfully.

block1.c compiled successfully.

block2.c compiled successfully.

block3.c compiled successfully. Hit <RETURN> to continue.

error log file

If your compile should fail, ufmenu creates an error log file in the current directory. The name of log file is of the form "username<pid>.complog". Error messages are appended to the log file. The script displays a message similar to the following: block1.c did not compile. Refer to username6370.complog for error message. *Link* Links the object file of a primary C or C++ program with the object files of any subprograms which can be referenced. The link option calls the uflink utility . The main object file must reside in your current directory. You can use a file specification for your subroutines. Ufmenu automatically invokes the uflink script. The environment variable UGII_USERFCN must be defined and point to the Open C API library; otherwise, the script defaults to the installation directory. The following example links an external Open C API image in debug mode.

Enter option (1-7,q) [q]: 3

Link internal/external user function (i/e) [i]: e

Link a C++ image (y/n) [n]:

Default uflink options: -m

Change uflink options (y/n) [n]: y

Remove -m (y/n) [n]:

Add new options: -d

New uflink options: -m -d

Enter program to link => testopen

Enter any subroutines => block1.o block2.o block3.o

Enter any libraries =>

uflink:WARNING - UGII_USERFCN variable not set.

Using libraries in <path> as a default.

Linking with: block1.o block2.o block3.o.

/bin/cc options: -Wl,-q,-E,-B,immediate,+s,-L,<path> -g

Linking... testopen for external execution.

uflink:link SUCCESSFUL - Wed Oct 15 10:33:07 PDT 1997

Hit <RETURN> to continue.

Run

Allows you to run an external Open C API program. The following example shows how an argument is passed to the program.

Enter option (1-7,q) [q]: 4

Enter external user function to run [testopen]:

Run debug mode (y/n) [n]: n

Enter arguments to pass to testopen []: newbox

Hit <RETURN> to continue.

The next example shows the prompts when you run in debug mode.

Enter option (1-7,q) [q]: 4

Enter external user function to run [testopen]:

Run debug mode (y/n) [n]: y

Copyright Hewlett-Packard Co. 1985,1987-1994. All Rights Reserved.

<<<< XDB Version A.09.01 HP-UX >>>>

No core file

Procedures: 4

Files: 4

testopen.c: main: 15: int units = 2;

>

From here, you can now enter debug commands. The debugger and debugger prompt are platform specific.

Change Directory

Allows you to change the current directory by entering a new directory pathname.

Enter option (1-7,q) [q]: 5

Current directory => /users/ali/ugopen/test

Enter new directory [.]: /users/ali/ugopen

New directory => /users/ali/ugopen

List Directory

Lists the contents of the current directory. You can specify a file template. When you choose this option, ufmenu displays your current directory and prompts you for a template of the files to list. The default template is to list all files. You can enter any valid wildcard specification.

Current directory => /users/ali/ufun

Enter file(s) to list [*]:

Non-menu Activities

Spawns a child process. The script prompts you for the shell type as follows:

Enter Shell Type (sh/csh/ksh) [ksh]:

The default value is the Korn shell (ksh). You can spawn a Bourne shell (sh) or a C-shell (csh) by entering the appropriate value. For example, enter csh to spawn a C-shell

Common Object Model

Common Object Model

This chapter provides a brief overview of the most important types of classes found in the NX Open API. For more details, look at the NX Open API Reference for whichever programming language you are using. The object model is the same in all programming languages. Classes are organized into namespaces based on application area.

NX Open classes

TaggedObject — Base class for any class used to represent entities in the NX model, e.g. Line, Extrude. TaggedObject is used to represent any persistent entity in the NX model, in other words, entities that are saved when the part is saved. However, some TaggedObject classes represent non-persistent entities. The defining characteristic of a TaggedObject is that it has a Tag that can be used to interoperate with the UF API.

NXObject — Inherits from TaggedObject. Provides methods and properties for setting and getting names and attributes on the entity

TransientObject — Represents something that doesn't have a Tag and is not saved when the part is saved. The Dispose method must be called to dispose of the TransientObject. In languages with garbage collection, Dispose is automatically called during garbage collection, but in C++, Dispose must be explicitly called.

TaggedObjectCollection — Base class for classes representing a collection of tagged objects. These classes also typically have methods for creating new tagged objects. For example, CurveCollection contains methods for creating new curves. You can obtain all the objects in the part of a particular type using the TaggedObjectCollection for that object type. For example, to perform an operation on all curves in a part, in VB.NET you can use

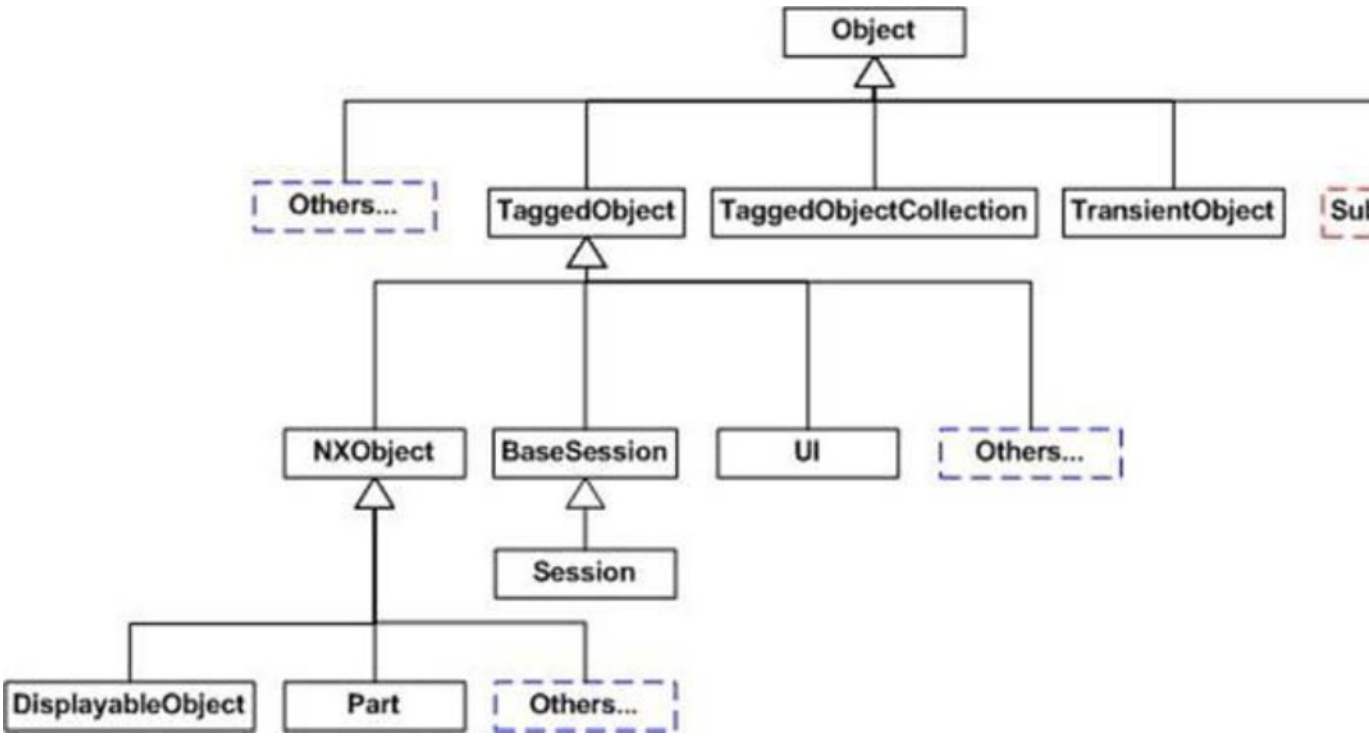
```
Dim curve As NXOpen.Curve
For Each curve In part.Curves
   ...
 Next
```

Session and UI — Serve as the "gateway" classes for the API. References to all other objects in the API are obtained either directly or indirectly via methods and properties on these two classes. UI can be used only when the program is run from within the NX user interface. In other words, UI cannot be used in programs run in the Batch mode of execution.

Part — Represents an NX part. Contains many TaggedObjectCollection objects which can be used to obtain the tagged objects in the part and to create new tagged objects. For example, Part contains a TaggedObjectCollection named Features that contains all the features in the part and that can be used to obtain FeatureBuilder objects which are used to create or edit features.

Builder — Features and many other entities are created and edited through a Builder class. Builders have methods (or properties in .NET) for getting and setting the defining data for the entity being built. These methods often correspond closely to input data on the user interface's command for creating or editing the entity. The entity can also be queried using the Builder. In

order to create the entity or apply the edits made with the builder, you must call the Commit method. When you are finished using the builder, you must call the Destroy method.



Browsing Classes Through an IDE

In Visual Studio, after you add a reference to NXOpen.dll or NXOpenUI.dll, you can use Visual Studio's Object Browser to browse the NX Open classes and read their documentation. IntelliSense also works with the NX Open classes.

Some other IDEs have similar technology. For example, in Eclipse, after you add a reference to NXOpen.jar, you can use Eclipse to browse the NX Open classes and documentation and Eclipse has a technology similar to IntelliSense called "content assist."

For more details, read the documentation for your IDE.

Common NX Objects

Accessing Bodies, Faces and Edges

Each part may contain any number of solid bodies. Each solid body is defined by a set of faces and edges. Each face contains a reference to the body it belongs to and a list of edges that define the face. Each edge will also contain a reference to the owning body and a list of faces that are defined by the edge. NX Open makes it very easy to find the bodies in a part and then to find the relationships between the faces and edges that are used to define the solid body. This

section shows examples of how the methods and properties of the body, face and edge objects are used to access the related objects.

Typically a body will have multiple faces and an edge will be used by two faces. However, there are exceptions. For instance, a sphere will only have a single face and no edges. Another example a cone, which will have two faces and a single edge.

The examples show how to access the following relationships:

- NX session → list of parts
- part → list of solid bodies
- solid body → list of faces

  solid body → list of edges
- face → list of associated edges

  face → solid body
- edge → list of associated faces

  edge → solid body

Bodies, Faces and Edges - Language Specific Details

NX Open for C++
NX Open for .NET
NX Open for Java

NX Open for C++

NX session → list of parts

To access all parts in an NX session, use the Parts property to access the Part Collection. Then use the collection's iterator to access each part.

```
Session *NXSession = Session::GetSession();
PartCollection *partList = NXSession->Parts();
PartCollection::iterator itr;
for ( itr = partList->begin(); itr != partList->end(); ++itr )
{
 processPart(*itr);
}
```

part → list of solid bodies

To access all solid bodies in a part, use the Bodies property to access the Body Collection. Then use the collection's iterator to access each body.

```
void processPart(Part *partObject)
{
   BodyCollection *bodyList = partObject->Bodies();
   BodyCollection::iterator itr;
   for (itr = bodyList->begin(); itr != bodyList->end(); ++itr)
   {
    processBodyFaces(*itr);
    processBodyEdges(*itr);
```

```
    }
}
```

solid body → list of faces

To access the faces of a body use the GetFaces() method to return an array of faces.

```
void processBodyEdges(Body *bodyObject)
{
    std::vector <Edge *> edgeArray = bodyObject->GetEdges();
    for (int inx = 0;  inx < (int)edgeArray.size(); ++inx)
      {
                  processEdge(edgeArray[inx]);
      }
}
```

solid body → list of edges

To access the edges in a body use the GetEdges() method to return an array of edges.

```
void processBodyEdges(Body *bodyObject)
{
    std::vector <Edge *> edgeArray = bodyObject->GetEdges();
    for (int inx = 0;  inx < (int)edgeArray.size(); ++inx)
    {
    processEdge(edgeArray[inx]);
    }
}
```

face → list of associated edges
face → solid body

To access the edges for a face use the GetEdges() method to return an array of edges. To access the face's body use the GetBody() method.

```
void processFace(Face *faceObject)
{
    std::vector<Edge *> edgeArray = faceObject->GetEdges();
    for (int inx = 0;  inx < (int)edgeArray.size(); ++inx)
    {
            processEdge(edgeArray[inx]);
    }
    Body *bodyOfFace = faceObject->GetBody();
}
```

edge → list of associated faces
edge → solid body

To access the faces associated with and edge use the GetFaces() method to return an array of faces. To access the edge's body use the GetBody() method.

```
void processEdge(Edge *edgeObject)
{
    std::vector<Face *> faceArray = edgeObject->GetFaces();
    for (int inx = 0;  inx < (int)faceArray.size(); ++inx)
    {
```

```
            processEdgeFace(faceArray[inx]);
        }
        Body *bodyOfEdge = edgeObject->GetBody();
    }
```

NX Open for .NET

NX session → list of parts

To access all parts in an NX session, use the Parts property to access the Part Collection. Then use a standard iterator method to access each part.

```
Dim NXSession As Session = Session.GetSession
For Each partObject As Part In NXSession.Parts()
  processPart(partObject)
Next partObject
```

part → list of solid bodies

To access all solid bodies in a part, use the Bodies property to access the Body Collection. Then cast the object to the generic Object class to access the face and edge methods.

```
Sub processPart(ByVal partObject As Part)
   For Each bodyObject As DisplayableObject In partObject.Bodies
        processBodyFaces(CType(bodyObject, Object))
        processBodyEdges(CType(bodyObject, Object))
    Next bodyObject
End Sub
```

solid body → list of faces

To access the faces of a body use the GetFaces() method to return an array of faces.

```
Sub processBodyFaces(ByVal bodyObject As Object)
    For Each faceObject As Face In bodyObject.GetFaces()
        processFace(faceObject)
    Next faceObject
End Sub
```

solid body → list of edges

To access a the edges in a body use the GetEdges() method to return an array of edges.

```
Sub processBodyEdges(ByVal bodyObject As Object)
    For Each edgeObject As Edge In bodyObject.GetEdges()
        processEdge(edgeObject)
    Next edgeObject
End Sub
```

face → list of associated edges

face → solid body

To access the edges for a face use the GetEdges() method to return an array of edges. To access the face's body use the GetBody() method.

```
Sub processFace(ByVal faceObject As Face)
    For Each edgeObject As Edge In faceObject.GetEdges()
        processEdge(edgeObject)
```

```
        Next edgeObject
        Dim bodyOfFace As Body = faceObject.GetBody()
    End Sub
```

edge → list of associated faces

edge -→ solid body

To access the edges for a face use the GetEdges() method to return an array of edges. To access the face's body use the GetBody() method.

```
    Sub processEdge(ByVal edgeObject As Edge)
        For Each faceObject As Face In edgeObject.GetFaces()
            processEdgeFace(faceObject)
        Next faceObject
         Dim bodyOfEdge As Body = edgeObject.GetBody()
    End Sub
```

NX Open for Java

NX session → list of parts

To access all parts in an NX session, use the "parts" property to access the Part Collection. Then use the collection's iterator to access each part.

```
    NXSession = (Session) SessionFactory.get("Session");
    Part partObject;
    PartCollection partList = NXSession.parts();
    PartCollection.Iterator itr;
    for (itr = partList.iterator(); itr.hasNext();)
    {
      partObject = (Part) itr.next();
      processPart(partObject);
    }
```

part → list of solid bodies

To access all solid bodies in a part, use the "bodies" property to access the Body Collection. Then use the collection's iterator to access each body.

```
    public static void processPart(Part partObject)
    {
       BodyCollection bodyList = partObject.bodies();
       BodyCollection.Iterator itr;
       for (itr = bodyList.iterator(); itr.hasNext();)
       {
           Body bodyObject = (Body) itr.next();
           processBodyFaces(bodyObject);
           processBodyEdges(bodyObject);
       }
    }
```

solid body → list of faces

To access the faces of a body use the getFaces() method to return an array of faces.

```
    public static void processBodyFaces(Body bodyObject)
```

```
{
    Face faceArray[] = bodyObject.getFaces();
     for (int inx=0; inx <(int)faceArray.size(); ++inx)
    {
       processFace(faceArray[inx]);
    }
}
```

solid body → list of edges

To access a the edges in a body use the getEdges() method to return an array of edges.

```
public static void processBodyEdges(Body bodyObject)
{
    Edge edgeArray[] = bodyObject.getEdges();

    for (int inx = 0; inx < edgeArray.length; ++inx)
    {
       processEdge(edgeArray[inx]);
    }
}
```

face → list of associated edges
face → solid body

To access the edges for a face use the getEdges() method to return an array of edges. To access the face's body use the getBody() method.

```
public static void processFace(Face faceObject)

{

    Edge edgeArray[] = faceObject.getEdges();



    for (int inx=0; inx < edgeArray.length; ++inx)

    {

      processEdge(edgeArray[inx]);

    }



    Body bodyOfFace = faceObject.getBody();

}
```

edge → list of associated faces
edge → solid body

To access the faces associated with and edge use the getFaces() method to return an array of faces. To access the edge's body use the getBody() method.

```
public static void processEdge(Edge edgeObject)
{
    Face faceArray[] = edgeObject.getFaces();

     for (int inx=0; inx <faceArray.length; ++inx)
    {
      processEdgeFace(faceArray[inx]);
    }

     Body bodyOfEdge = edgeObject.getBody();
}
```

Create and Edit Features

Most features use a builder method to create new features and to edit existing features. Features that do not use the builder method use constructor methods that are specific to the feature. The feature specific constructors are defined in the language reference guides. This section discusses the general concepts of the builder method.

Builder Method

The builder method uses the following general steps.

Create a New Feature

1. Create an instance of the builder object for the desired feature type providing a null object as input.
2. Edit the properties of the builder object to set the feature parameters and options.
3. Use the Commit method of the builder object to create an instance of the feature. The Commit method will return the new feature object.
4. Use the Destroy method of the builder object to delete the builder object.

Edit an Existing Feature (same steps except provide an existing object)

1. Create an instance of the builder object for the desired feature type providing the object of the feature to edit.
2. Edit the properties of the builder object to edit the feature parameters and options.
3. Use the Commit method of the builder object to edit the existing feature.
4. Use the Destroy method of the builder object to delete the builder object.

Each type of feature that uses the builder method will have a specific builder class. All builder classes can be found in the Features name space. For instance the class for the Block Feature builder is found in: Features.BlockFeatureBuilder.

The create a builder object, the Part Class contains a Features property that contains create methods for each builder class. For instance, given "workPart" as an instance of a part object, the create method for a Block Feature builder is workPart.Features.CreateBlockFeatureBuilder(), and

the create method for an Edge Blend Feature builder is workPart.Features.CreateEdgeBlendFeatureBuilder().

Create and Edit Features - Language Specific Details

NX Open for C++
NX Open for .NET
NX Open for Java

NX Open for C++

The following examples shows how to create a feature builder for a Block Feature and then use the feature builder to create a new Block Feature. A feature builder is then created to edit the Block Feature.

```
Session *NXSession = Session::GetSession();
Part *workPart (NXSession->Parts()->Work());

Feature *nullFeature (NULL);

Point3d origin = new Point3d(0.0, 0.0, 0.0);

//**************************************************************************
*****
//CREATE BLOCK

BlockFeatureBuilder *newBlock = NULL;

newBlock = workPart->Features()-
>CreateBlockFeatureBuilder(nullFeature);

newBlock->SetOriginAndLengths(origin, "50", "80", "100");

Feature *blockFeature = newBlock->CommitFeature();

newBlock->Destroy();

//**************************************************************************
*****

//EDIT BLOCK

BlockFeatureBuilder *oldBlock = workPart->Features()-
>CreateBlockFeatureBuilder(blockFeature);

oldBlock->SetOriginAndLengths(origin, "100", "20", "50");

oldBlock->CommitFeature();

oldBlock->Destroy();
```

NX Open for .NET

The following examples shows how to create a feature builder for a Block Feature and then use the feature builder to create a new Block Feature. A feature builder is then created to edit the Block Feature.

```
 Dim NXSession As Session = Session.GetSession()
            Dim workPart As Part = NXSession.Parts.Work

            Dim nullFeature As Feature = Nothing

            Dim origin As Point3d = New Point3d(0.0, 0.0, 0.0)


'**********************************************************************
****
            'CREATE BLOCK

            Dim newBlock As BlockFeatureBuilder =
workPart.Features.CreateBlockFeatureBuilder(nullFeature)

            newBlock.SetOriginAndLengths(origin, "50", "80", "100")

            Dim blockFeature As Feature = newBlock.CommitFeature()

            newBlock.Destroy()


'**********************************************************************
****
            'EDIT BLOCK

            Dim oldBlock As BlockFeatureBuilder =
workPart.Features.CreateBlockFeatureBuilder(blockFeature)

            oldBlock.SetOriginAndLengths(origin, "100", "20", "50")

            oldBlock.Commit()

            oldBlock.Destroy()
```

NX Open for Java

The following examples shows how to create a feature builder for a Block Feature and then use the feature builder to create a new Block Feature. A feature builder is then created to edit the Block Feature.

```
Session NXSession = (Session)SessionFactory.get("Session");
Part workPart = NXSession.parts().work();

Point3d origin = new Point3d(0.0, 0.0, 0.0, 0.0);

nxopen.features.Feature nullFeature = null;
```

```
//**************************************************************************
*****
//CREATE BLOCK

nxopen.features.BlockFeatureBuilder newBlock =
workPart.features().createBlockFeatureBuilder(nullFeature);

newBlock.setOriginAndLengths(origin,  "50", "80", "100");

nxopen.features.Feature blockFeature = newBlock.commitFeature();

newBlock.destroy();
//**************************************************************************
*****
//EDIT BLOCK

nxopen.features.BlockFeatureBuilder oldBlock =
workPart.features().createBlockFeatureBuilder(blockFeature);

oldBlock.setOriginAndLengths(origin,  "100", "20", "50");

oldBlock.commitFeature();

oldBlock.destroy();
```

Other NX Operations

[Model Update](#)
[Sketcher Interactions](#)

Model Update

Most NX Open methods which modify the internal NX data model perform a Model Update before returning to the calling application. A few methods are designed to be executed multiple times without Model Update and require the programmer to explicitly invoke a Model Update when the set of operations are complete. The language reference manual identifies which methods require the user to explicitly invoke Model Update. Use the following commands to invoke a Model Update.

Model Update is designed to process a list of changes in a specific order and should never be interrupted or terminated before an update is complete. Interrupting the update process can leave the NX session and part in an unpredictable state, including session and part corruption.

Interactive NX Commands are designed to define a set of changes and when complete to invoke Model Update to execute the desired changes. So interrupting an interactive NX Command and executing a Model Update can also leave the NX session and part in an unpredictable state.

To enable a wide range of customization, NX Open provides many methods that interrupt the normal flow of NX to execute custom application code. For example, a User Defined Object can provide an update callback that is executed during the Model Update process. Other examples are custom selection filters or mouse tracking callbacks which are executed in response to user actions. Also, pre/post menu item callbacks can execute just before or just after interactive NX Commands.

When coding any event handler or callback, the programmer must take care to understand the state of NX and must understand which NX Open methods may not be called within the context of the event. In general no methods should be called during a Model Update which create, edit, or delete objects which have already been processed by current model update.

Since sketches can be created interactively while creating other NX features, any Pre/Post actions added to the interactive sketcher should be limited to model checks and reporting. No model edits should be attempted while entering or exiting a sketch (See Sketcher Interactions).

Note:

A single call to model update may result in many changes to the NX session. The overall update process can be successful even when some objects fail to update. For instance, a single feature may fail to update because the parameters are no longer valid for its construction but the solid body associated with the feature may still be in a valid state. When calling model update it is important to process the error list and report any unexpected failure.

Update

The Update class has all the methods you need to query and control the update process. To get an instance of this class, use the UpdateManager method/property on your session. Update class is also used to properly delete NX objects.

Explicit Update —You can update the NX session explicitly by calling the *DoUpdate()* method. You can specify an undo mark the system rolls back to in case it encounters errors.

theSession->UpdateManager()->DoUpdate(myUndoMark);

Interpart Update — You can choose to limit update only to work part by setting the interpart delay flag. If the flag is turned on, then update is only limited to the current work part. If you choose to update all the parts in the session then, 1) Turn off the interpart delay flag and then call DoUpdate() or 2) Directly call the *DoInterPartUpdate*()

Deleting Objects — A proper way to delete NX objects is to add them to the delete list. There is a global delete list which keeps track of all objects added to the delete list. When a model goes through update, all objects in the delete list (and the child objects dependent on it) are deleted. You can add object to the delete list using *AddToDeleteList().* When you add an object to delete list, all the child objects dependent on it are notified and the child objects decide whether they too should be deleted. For example, if you delete the block feature and you have a hole on the block feature then the hole feature is also added to the delete list.

You can query all the objects in the delete list at any give time using *GetDeleteList().* To remove a particular object from the delete list use *RemoveFromDeleteList().* Be careful while removing

objects from delete list. Only add and remove objects from the delete list that your application creates.

Update Errors — If there are any errors during the model update, then they are added to the update errorlist. You can access this list using *theSession->UpdateManager()->ErrorList()->GetLength() theSession->UpdateManager()->ErrorList()->GetErrorInfo(3) //Gets the third error in the list*

Sketcher Interactions

Sketching is a fundamental core behavior of NX which is widely used by many NX Commands and works in various contexts. When considering impacting any behavior of interactive NX Sketching the risks versus reward need to be strongly considered. Also, thorough testing in all contexts is required to ensure that no undesirable behaviors have been introduced.

NX Open has been designed to enable the programmatic creation of new sketches, the programmatic editing of existing sketches and *limited* customization of core NX interactive sketching. When customizing the behavior of core NX interactive sketching the process of entering or exiting a sketch should not be modified. It is possible to add custom command buttons to the core NX sketch environment but the programmer should limit the NX Open method calls to those found in the sketch classes. For instance, to create geometry the programmer should use one of the *Sketch.AddGeometry()* methods and to do a model update the programmer should use *Sketch.Update()*.

Sketches can be imbedded within NX Features. This can be done on the fly as part of the Feature Command. Therefore, adding a Pre/Post operation to the interactive NX sketcher is not recommended since the pre/post operation would take place during the middle of an interactive NX Feature Command. Placing custom code that may execute a Model Update within an interactive NX Command can lead to unpredictable results including NX session and part corruption, as discussed in Section 4.0 Model Update.

Since sketches can be created interactively while creating other NX features, any Pre/Post actions added to the interactive sketcher should be limited to model checks and reporting. No model edits should be attempted.

Interoperation between the Common API and Open C

Interoperation between Open C and common API

Interoperability is the ability to access constructs written in one programming language from another. NX Open for common API programs contain built-in support for interoperability with Open C APIs by providing .NET and Java wrappers for Open C APIs.

In cases where you want to make calls to constructs in your custom C program from a NXOpen common API program, .NET framework and JNI (Java Native Interface) both allow interoperability between Common API and legacy C applications. Just like NXOpen wrappers for Open C API, you will create wrappers for your legacy APIs and use .NET or JNI for interoperability. See Calling Legacy Open C from Common API Applications for examples.

Wrappers

Open C API is developed over many years, contains thousands of functions and as such provides a wide range of coverage. Common API ensures access to this coverage by generating .NET and Java wrappers for all the functions in the Open C API.

Modules in Open C map to classes in common API and the functions within the modules map to methods on the common API classes (See Naming Conventions). While common API wrappers (both .NET and Java) for Open C API can be called anytime, the only issue is the basic object model. Common API represents objects as classic object-oriented objects, while Open C represents them as tags. Examples below explain how to switch between objects and tags. Some simple concepts to understand before using the wrappers:

UFSession

.NET and Java wrapped common API classes are defined in nxopen.uf namespace (nxopen.uf package in case of Java). To access the wrapped class you should first get an instance of the UFSession. Wrapper classes (corresponding to Open C module) are defined as methods on the UFSession class. For example curve() method on UFSession class returns an instance of UFCurve class.

TaggedObjectManager and NXObjectManager

TaggedObjectManager is an interface class defined in the nxopen package. Use the get() method on this class to obtain the NX Open object corresponding to a tag.

NXObjectManager is .NET equivalent of TaggedObjectManager. Use the Get() method on this class to obtain the NX Open object corresponding to a tag.

Tag Property

All NX Open objects have a property Tag() (tag() method in Java). This property/method returns the tag of NX Open object to use with wrapped methods.

Wrappers - Language Specific Examples

NX Open for .NET NX Open for Java

NX Open for .NET

The following example creates an arc using the Open C API and then queries the arc data using the NX Open API for .NET

```
Dim theSession As Session = Session.GetSession()
Dim theUFSession As UFSession = UFSession.GetUFSession()


' Create an ARC using the Open API

Tag arc;
Dim arc_coords As UFCurve.Arc
arc_coords.radius = 1.0
arc_coords.arc_center = New Double(){1.0, 1.0, 0.0}
arc_coords.start_angle = 0.0
arc_coords.end_angle = Math.PI
arc_coords.matrix_tag =
theSession.Parts.Display.WCS.CoordinateSystem.Orientation.Tag
theUFSession.Curve.CreateArc( arc_coords, arc)


' Get the Arc Object to use with NX Open
NXOpen.Arc nxArc= CType(NXOpen.Utilities.NXObjectManager.Get(arc),
NXOpen.Arc)


'Get the Arc parameters using NX Open APIs
Dim start_angle As Double = nxArc.StartAngle
Dim end_angle As Double = nxArc.EndAngle
Dim arc_center As NXOpen.Point3d = nxArc.CenterPoint
```

NX Open for Java

The following example creates an arc using the Open C API and then queries the arc data using the NX Open API for Java

```
Session theSession = (Session)SessionFactory.get("Session");
UFSession theUFSession = (UFSession)SessionFactory.get("UFSession");

/* Create Arc using Open C API wrapper */
UFCurve ufCurve = theUFSession.curve();
UFCurve.Arc ufArc = new UFCurve.Arc();
UFCsys ufCsys = theUFSession.csys();

/* Fill out the data structure */
ufArc.startAngle = 0.0;
ufArc.endAngle = 3.0;
ufArc.arcCenter=new double[3];
ufArc.arcCenter[0] = 0.0;
ufArc.arcCenter[1] = 0.0;
ufArc.arcCenter[2] = 1.0;
ufArc.radius = 2.0;


/* Create Arc */
```

```
Tag wcsData = ufCsys.askWcs();
ufArc.matrixTag = ufCsys.askMatrixOfObject(wcsData);
Tag arcTag = ufCurve.createArc(ufArc);

/* Get the Arc Object to use with NX Open*/
Arc arc = (Arc)theSession.taggedObjectManager().get(arc2Tag);

/* Get arc parameters using NX Open Java APIs */
double start_angle = arc.startAngle();
```

Mapping Open C to NX Open Common API

This section explains how you can map Open C functions and arguments to NX Open .NET methods and NX Open Java methods.

Naming Conventins

What are the NX Open Common API classes?

Each module in the Open C API maps to an NX Open .NET class and Java Class. Corresponding Java class may be an interface class

Open Module Name	NX Open .NET Class Name	NX Open Java Class Name
UF_PART	UFPart	UFPart
UF_CURVE	UFCurve	UFCurve
UF_UDOBJ	UFUDobj	UFUdobj

How do NX Open methods, structures and enums get their names?

Functions

Functions in C API map to methods in common API

.NET

Open Function Name	NX Open .NET Class Name	NX Open .NET Method Name
UF_CURVE_create_arc()	UFCurve	CreateArc()

JAVA

Open Function Name	NX Open Java Class Name	NX Open Java Method Name
UF_CURVE_create_arc()	UFCurve	createArc()

Structures

.NET

Structures in C API map to .NET structure as follows:

Open Structure Name	NX Open .NET Class Name	NX Open .NET Structure Name
UF_CURVE_spline	UFCurve	Spline

JAVA

There are no structures in Java. Open C API structures are mapped to Java classes

Open Structure Name	NX Open Java Interface Class Name	NX Open Java Class for Open C structure
UF_CURVE_spline	UFCurve	Spline

Enums

.NET

Open C API enums map to .NET enums as follows:

Open Enum Name	NX Open .NET Class Name	NX Open .NET Enum Name
UF_CURVE_direction_e	UFCurve	Direction

JAVA

There are no enums in Java. Enums are mapped to a Java class and the enum values map to constant field integer value.

Open Structure Name	NX Open Java Interface Class Name	NX Open Java

		Class Name for Open C enum
UF_CURVE_direction_e	UFCurve	Direction

Function Pointers

Function Pointers in C are used in variety of situations, like passing operations to a generic algorithm based on the types being used in the algorithm (for example qsort()). .NET has direct equivalent of function pointers in form of delegates and Java supports similar functionality through reflection.

Note:

Open C API uses function pointer mechanism for callbacks (see UF_add_callback_function in Open C Reference Guide) and NX Open common API has wrapped methods for NX callback mechanism (see AddCallbackFunction in .NET reference guide).

Function Pointers - Language Specific Details

NX Open for .NET NX Open for Java

NX Open for .NET

In the .NET Framework, delegates serve the role of function pointers. A delegate is a class that can hold a reference to a method and is equivalent to a type-safe function pointer or a callback function. To use delegates, one first declares a delegate that has the return type and accepts the same number of parameters as the methods one will want to invoke as callback functions. Secondly one needs to define a method that accepts an instance of the delegate as a parameter. Once this is done, a method that has the same signature as the delegate (i.e. accepts same parameters and returns the same type) can be created and used to initialize an instance of the delegate which can then be passed to the method that accepts that delegate as a parameter. Note that the same delegate can refer to static and instance methods, even at the same time, since delegates are multicast.

Example

This example demonstrates how to pass delegates to an unmanaged function expecting function pointers. In this example delegate refers to a static method.

Consider the following TestCallBack() function in TestLib.dll:

```
typedef void (* fn_t)(const char *, int);
extern "C" __declspec(dllexport) TestCallBack (fn_t ptr_to_func, int i)
{
    ptr_to_func("The result is: ", i+1);
}
```

Here, "fn_t" is a function pointer.

The following mapping shows that:

- the fn_t function pointer is mapped as delegate, FnT

- the Application class contains a prototype for the TestCallBack method. This method passes a delegate where an expected parameter is a callback function.

C# Equivalent

```
using System;
using System.Runtime.InteropServices;
class Application
{
    //Mapping function pointer in the C world  to delegate in the .Net
world
    public delegate void FnT(string s, int j);

[DllImport("TestLib.dll",CallingConvention=CallingConvention.Cdecl)]
    private static extern void TestCallBack (FnT Delegatefn, int i);
}
```

Client application

A client implements a function, which has the same signature as that of the delegate. For example, the following implements the DoSomething() function which has the same signature as the delegate, FnT. The sample instantiates the delegate before passing it to TestCallBack.

```
class ClientApp
{
    private static void DoSomething(string s, int j)
    {
        Console.WriteLine(s + j.ToString());
    }
    public static void Main(String []args)
    {
        FnT Delegatefn = new Application.FnT (DoSomething);
        TestCallBack (Delegatefn, 99);
    }
}
```

Output

The result is 100.

NX Open for Java

In Java, reflection can be used to achieve similar results as a function pointer but there is no equivalent to .NET delegates in Java and consequently no straightforward way to pass a Java method to a C function expecting a function pointer.

Data Type Mapping Tables

Use the following mapping tables to map from C data types to .NET data types:

Basic Types
Pointer Types

Basic Types

.NET

C Data Type	.NET Data Type	Remarks
char	byte	An 8-bit unsigned integer
signed char	sbyte	An 8-bit signed integer
short	short	A 16-bit signed integer
int	Int	A 32-bit signed integer
float	Float	A single-precision (32-bit) floating-point number
double	double	A double-precision (64-bit) floating-point number

JAVA

C Data Type	Java Data Type	Remarks
char	char	An 16-unicode character
short	short	A 16-bit signed integer
int	int	A 32-bit signed integer
float	float	A single-precision (32-bit) floating-point number
double	double	A double-precision (64-bit) floating-point number
bool	boolean	

Pointer Types

.NET

C Data Type	Corresponding .NET Data Type	
	Argument	Mapping
enum*	Input	ref enum
	Output	out enum
	Input/Output	ref enum

	Structure member	enum[]
enum**	Output to be freed	out enum[]
char*	Input	string
	Output	out string
	Input/Output	ref string
	Return	string
	Structure member	string
char**	Input	string[]
	Output	out string
	Output to be freed	out string
	Structure member	string[]/ byte[][]
void*	Input	IntPtr
	Output	out IntPtr
	Input/Output	ref IntPtr
	Return	IntPtr
	Structure member	IntPtr
void**	Input	IntPtr
	Input/Output	ref IntPtr
	Output to be freed	out IntPtr
void***	Output	IntPtr
struct*	Input	ref struct
	Output	out struct
	Input/Output	ref struct
	Output to be freed	out struct
	Structure member	struct
bool*	Input	ref bool

	Output	out bool

Java

Since there is no pointer concept in Java, NX Open Java wrappers create class and adds all the output parameters as member of the class. Input pointers for int/float/double are treated as arrays.

C Data Type	Corresponding Java Data Type	
	Argument	**Mapping**
enum*	Input/Output	Class with constant field values
char*	Input/Output	string
char**	Input/Output	string[]
void*	Currently no mapping available	
struct*	Input/Output	Class with struct members as class members

Example: Following example shows mapping of output parameters from Open C to NX Open Java API

Open C API:

```
int UF_CURVE_ask_feature_curves
(
tag_t curve_feature_id
int * num_curves
tag_t ** feature_curves

)
```

To map this function to Java, NX Open creates a Java class UFCurve.AskFeatureCurvesData and the output parameter of Open C API are members of the class

```
public class AskFeatureCurvesData
{
    public int numCurves;
    public nxopen.Tag[] featureCurves;
}
```

Consequently the Java method corresponding to the Open C API is:

```
UFCurve.AskFeatureCurvesData askFeatureCurves(Tag curveFeatureId)
                                              throws NXException,
                                                     RemoteException
```

Fixed Length Arrays

.NET

C Data Type	Corresponding .NET Data Type	
	Argument Type	**Mapping**
ctype[] (int, float, double, tag_t)	Input	ctype[]
	Output	ctype[]
	Output to be freed	ctype[]
ctype[][] (int, float, double, tag_t)	Input	ctype[,]
	Output	ctype[,]
	Output to be freed	ctype[,]
ctype[][][] (int, float, double, tag_t)	Input	ctype[,,]
	Output	ctype[,,]
	Output to be freed	ctype[,,]
char[]	Input	string
	Output	out string
	Input/Output	ref string
	Structure member	string
char[][]	Input	string[]
	Output	string[]
struct[]	Input	struct[]
	Output	struct[]
bool[]	Input	bool[]

Java

C Data Type	Corresponding Java Data Type	
	Argument Type	**Mapping**
ctype[] (int, float, double, tag_t)	Input	ctype[]

	Output	ctype[]
	Output to be freed	ctype[]
ctype[][] (int, float, double, tag_t)	Input	ctype[,]
	Output	ctype[,]
	Output to be freed	ctype[,]
ctype[][][] (int, float, double, tag_t)	Input	ctype[,,]
	Output	ctype[,,]
	Output to be freed	ctype[,,]
char[]	Input/Output	string
char[][]	Input/Output	string[]
struct[]	Input/Output	Class representating the C structure
bool[]	Input/Output	boolean[]

Note:

Ctype represents a basic C language type such as int, double, or float.

Variable Length Array Types

.NET

C Data Type	Corresponding .NET Data Type	
	Argument Type	**Mapping**
ctype* (int, float, double, tag_t)	Input	ctype[]
	Structure member	ctype[]
cytpe*** (int, float, double, tag_t)	Output to be Freed	out ctype[][]
enum*	Input	enum[]
Char***	Output to be freed	out string[]
Struct*	Input	Struct[]
	Structure member	Struct[]

Struct**	Input Input/Output Output to be freed	Struct[] Struct[] Out struct[]
Struct***	Output to be freed	Out struct[]
Bool*	Input	Bool[]

Calling C functions from Common API Applications

In some cases it may preferable to call API of a custom Open C application from NX Open common API application. The following examples show how to call C functions from .NET or Java based application. The example uses standard .NET or JNI (Java Native Interface) functionality to make these calls.

Calling C function from Common API Applications - Language Specific Examples

We will use the following Open C source file as an example application and show how .NET and Java can make calls to functions in this example application.

UFApp - Example Open C file

```
//****************************************************************************
*
// UFapp.cpp
//
// Sample code showing interactions with a .NET applications (VBapp)
//
// 1. Shows how to use dllexport to publish functions for call from
.NET


#include <stdio.h>

#include <uf.h>
#include <uf_ui.h>
#include <uf_mb.h>
#include <uf_exit.h>


//****************************************************************************
*************************
// Two local functions to be called via wrappers from .NET
static int myFunction1(void)
{

            UF_initialize();
```

```
            UF_print_syslog("BEGIN myFunction1\n",0);

            UF_print_syslog("END myFunction1\n",0);

            return(77);
}
//************************************************************************
************************
static int myFunction2(int value, char *string)
{
            int ret1,ret2;
            char message[MAX_STRING_SIZE] = "";

            ret1 = ret2 = 0;

            sprintf_s(message,"value:%d string:%s\n",value,string);

            UF_initialize();
            UF_print_syslog("BEGIN myFunction2\n",0);
            UF_print_syslog(message,0);

            UF_print_syslog("END myFunction2\n",0);

            return(value+1);
}

//************************************************************************
************************
// Wrappers for local functions to be called from .NET

extern "C" __declspec(dllexport) int UFappFunction1(void)
{
            return( myFunction1() );
}

//************************************************************************
************************
// Another local function to be called from .NET

extern "C" __declspec(dllexport) int UFappFunction2(int value, char
*string)
{
            return ( myFunction2(value,string) );

}

//************************************************************************
************************
```

```
// ufsta() - this is only for testing purposes, normally this DLL will
loaded by the .NET DLL
//  using the dllImport construct as shown below.
//
// Note: If this DLL needs to be unloaded before NX exists then the
.NET DLL will need
// to do it when it unloads.

extern void ufsta (char *param, int *retcode, int rlen)
{
            UF_initialize();
            UF_print_syslog("BEGIN UGapp (Running from File-
>Execute)\n",0);
            myFunction1();
            myFunction2(22,"Testing from File->Execute");
            UF_print_syslog("END UGapp\n",0);
}

//************************************************************************
*************************
//END
```

LANGUAGE SPECIFIC SECTIONS

NX Open for .NET
NX Open for Java

Example - Calling C function from .NET Application

The example below shows how VB .NET makes calls to unmanaged C. See the example C file we will use for this - UFapp.cpp

The calling conventions for the C functions is _cdecl while VB .NET uses _stdcall convention. So, you cannot directly call the C functions without "wrapping" them. To call functions UFappFunction1 and UFappFunction2 from NX Open for .NET application:

1. Make sure that the C functions are decorated with "dllexport" and the calling convention "_declspec" as the example shows above.
2. Create a wrapper class and add methods which wrap the C functions you wish to call. Example VB source file (wrappers.vb) below shows the wrapper class and methods. The methods must have same parameters as the function.
3. In the client VB application (see example VB source, VBApp.vb) called the wrapped methods from step 2.

Example source - wrappers.vb

```
'************************************************************************
******************
' wrappers to UFapp

Imports System
Imports System.Runtime.InteropServices
```

```
Public Class wrappers

'Use the path to your UFapp.dll
Const UFAPP_DLLNAME As String = "C:\local\UFapp"

'************************************************************************
**************
' Hookup to a function in UFapp

<DllImport(UFAPP_DLLNAME, EntryPoint:="UFappFunction1", _
CallingConvention:=CallingConvention.Cdecl)>
Shared Function UFappFunction1() As Integer
End Function

'************************************************************************
**************
' Hookup to another function in UFapp


<DllImport(UFAPP_DLLNAME, EntryPoint:="UFappFunction2", _
CallingConvention:=CallingConvention.Cdecl)>
Shared Function UFappFunction2(ByVal value As Integer, ByVal str As
String) As Integer
End Function

End Class
```

Example source - VBapp.vb

```
'************************************************************************
*******************
' VB application

    Imports System
    Imports System.IO
    Imports System.Windows.Forms
    Imports NXOpen
    Imports NXOpen.UF
    Imports VBapp.wrappers


    ************************************************************************
**********************
' Standard Entry Points called from NX

Module VBapp
    Public theNXsession As Session
    Public theUFsession As UFSession

    Public Sub Main()
        Dim value As Integer = 88
```

```
        Dim ret1 As Integer = 0
        Dim ret2 As Integer = 0

        theNXsession = Session.GetSession
        theUFsession = UFSession.GetUFSession

        theNXsession.LogFile.WriteLine("BEGIN VBapp (Running from File-
>Execute)")


        ' Call Functions in UFapp

        ret1 = UFappFunction1()
        ret2 = UFappFunction2(88, "This string is from VBapp")


        theNXsession.LogFile.WriteLine("funtions1:" & ret1.ToString & "
" & _
                        "function2:" & ret2.ToString)

        theNXsession.LogFile.WriteLine("END VBapp")
    End Sub

    Public Function GetUnloadOption(ByVal arg As String) As Integer
        Return Session.LibraryUnloadOption.Immediately
    End Function

    Public Function UnloadLibrary(ByVal arg As String) As Integer
        Session.GetSession().LogFile.WriteLine("VBapp: UnloadLibrary")
        Return 0
    End Function
End Module
```

Example - Calling C function from Java Application

We will use the same example C application (UFApp.cpp). To call the C functions from Java we need to use JNI framework. To create a JNI program one performs the following steps:

1. Create a Java program that contains the declaration of the native method(s) marked with the `native` keyword.
2. Write a main method that loads the library created in step 6 and uses the native method(s).
3. Compile the class containing the declaration of the native method(s) and the main with the `javac` compiler.
4. Use the `javah` compiler with the `jni` compiler option to generate a header file for the native method(s).
5. Write the native method in your language of choice (currently C, C++ or assembly).
6. Compile the header file and native source file into a shared library (i.e. a .dll on Windows or a .so file on non-Windows).

Example Java File

```
import java.util.*;
import nxopen.*;
class JavaApp {
    //Native method declarations
    private native int nativeUFappFunction1();
    private native int nativeUFappFunction2( int value, String name );
    //Load the native library, the library should be in your
environment
    //path
    //On non-Windows - LD_LIBRARY_PATH, on windows PATH
    static {
        System.loadLibrary("UFapp");
    }
    // Accessor methods used by our other Java classes . By hiding
    //  the actual calls to the native methods, we can change the
    //  implementation without affecting the classes calling in.

    public int UFappFunction1()
    {
        return(this.nativeUFappFunction1());
    }

    public int UFappFunction2( int value, String name )
    {
        return(this.nativeUFappFunction2( value, name ));
    }
    //Main functions
    public static void main (String args[]) {

      try
      {
        Session theSession = (Session)SessionFactory.get("Session");
        theSession.logfile().WriteLine("BEGIN Java app (Running from
File->Execute)");
        //Create class instance
        JavaApp app = new JavaApp();
        //Call native methods
        int ret1 = app.UFappFunction1();
        int ret2 = app.UFappFunction2(88, "This is string from Java");
        //Now we are back in Java
        theSession.logFile().writeLine("function1:" +
Integer.toString(ret1) );
      }
      catch (Exception ex)
      {
        System.out.println("Failed");
      }
    }
```

```
    public static int getUnloadOption() {
        return BaseSession.LibraryUnloadOption.EXPLICITLY;
    }
}
```

Generate the Header File

Generate the header file using: javah -jni JavaApp

The generated header file is shown below.

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>

/* Header for class JavaApp */
#ifndef _Included_JavaApp
#define _Included_JavaApp
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     JavaApp
 * Method:    nativeUFappFunction1
 * Signature: ()I
 */
JNIEXPORT jint JNICALL Java_JavaApp_nativeUFappFunction1
  (JNIEnv *, jobject);


/*
 * Class:     JavaApp
 * Method:    nativeUFappFunction2
 * Signature: (ILjava/lang/String;)I
 */
JNIEXPORT jint JNICALL Java_JavaApp_nativeUFappFunction2
  (JNIEnv *, jobject, jint, jstring);
#ifdef __cplusplus
}
#endif
#endif
```

Modify the UFapp.c file to add JNI functions

Revisited UFapp.c

```
//****************************************************************************
*
// UFapp.cpp
//
// Sample code showing interactions with a Java applications
//
//1. Add the JNI generated header to list of includes
//2. Implement the JNI functions - JNI functions directly call
```

```
//   local functions
//


#include <stdio.h>

#include <uf.h>
#include <uf_ui.h>
#include <uf_mb.h>
#include <uf_exit.h>
#include "JavaApp.h"


//============================================================================
//  Two local functions to be called via Java JNI functions
static int myFunction1(void)
{


    UF_initialize();
    UF_print_syslog("BEGIN myFunction1\n",0);

    UF_print_syslog("END myFunction1\n",0);

    return(77);
}
//============================================================================
static int myFunction2(int value, char *string)
{
    int ret1,ret2;
    char message[MAX_STRING_SIZE] = "";

    ret1 = ret2 = 0;

    sprintf_s(message,"value:%d string:%s\n",value,string);

    UF_initialize();
    UF_print_syslog("BEGIN myFunction2\n",0);
    UF_print_syslog(message,0);

    UF_print_syslog("END myFunction2\n",0);

    return(value+1);
}
```

```
//************************************************************************
*************************
// Java functions

JNIEXPORT jint JNICALL Java_JavaApp_nativeUFappFunction1(JNIEnv *env,
jobject obj)
            {    // Call the local functions
    int ret = myFunctions1();
    return(ret);
}
JNIEXPORT jint JNICALL Java_JavaApp_nativeUFappFunction2(JNIEnv *env,
jobject obj, jint a, jstring name)
            {
    // Get the characters from the jstring
    char *str = env->GetStringUTFChars(string, 0);
    // Call the local function
    int ret = myFunctions2( (int)a, str );
    // Release memory (required or else memory leaks)
    env->ReleaseStringUTFChars(string, str);
    return ret;
}

//************************************************************************
*************************
//

extern "C" __declspec(dllexport) int UFappFunction1(void)
{
    int ret1,ret2;
    char message[MAX_STRING_SIZE] = "";

    // Get the characters from the jstring
    char *str = env->GetStringUTFChars(name, 0);

}

//************************************************************************
*************************
//

extern "C" __declspec(dllexport) int UFappFunction2(int value, char
*string)
{
    return ( myFunction2(value,string) );
}

//************************************************************************
*************************
```

```
// ufsta() - this is only for testing purposes, normally this DLL will
loaded by the .NET DLL
//  using the dllImport construct as shown below.
//
// Note: If this DLL needs to be unloaded before NX exists then the
.NET DLL will need
// to do it when it unloads.

extern void ufsta (char *param, int *retcode, int rlen)
{
    UF_initialize();
    UF_print_syslog("BEGIN UGapp (Running from File->Execute)\n",0);
    myFunction1();
    myFunction2(22,"Testing from File->Execute");
    UF_print_syslog("END UGapp\n",0);
}

//************************************************************************
*************************
//END
```

Calling Common API from Legacy Open C Applications

This section shows how to call a common API method from a legacy C application. The example shows calling a NX Open .NET method from a legacy C application. JNI framework can be used to achieve similar functionality for NX Open Java API.

To call common API .NET methods from unmanaged C application:

1. Define the method signatures for calling into .NET. This are basically function pointers with _stdcall calling convention in the open C application.
2. Decorate functions in C application with "dllexport" so .NET can call those functions
3. Define delegates in .NET application which C application can call. The signatures should match those defined in step 1.
4. Define wrapper functions in .NET for the functions exported from C application (see step 2).
5. Define a function in the C application which registers the three signatures (step 1). Export it so .NET can register corresponding .NET methods (step 3).
6. Import the three C functions (see step 2 and step 5) in .NET application.

In short, .NET calls C functions which in turn call the registered .NET methods. This is the recommended way to call .NET methods from unmanaged code.

Example C file

```
//************************************************************************
*
// UFapp.cpp
//
```

```
// Sample code showing interactions with a .NET applications (VBapp)
//
//
// Shows how to capture a method/function pointers for calling into
.NET

#include <stdio.h>

#include <uf.h>
#include <uf_ui.h>
#include <uf_mb.h>
#include <uf_exit.h>

#include "reportError.h"

//============================================================================
*************************
// Define 3 different method signatures for callng into .NET

typedef void (__stdcall *method1Def) (void);
typedef int (__stdcall *functionADef) (int parm1);
typedef int (__stdcall *functionBDef) (char *parm1);

static method1Def VBmethod1 = NULL;
static functionADef VBfunctionA = NULL;
static functionBDef VBfunctionB = NULL;

//============================================================================
*************************
// A function to be called from .NET to register functions using
// the three signatures given above

extern "C" __declspec(dllexport) void UFappRegister(method1Def method1,
functionADef functionA, functionBDef functionB)

{
    UF_initialize();
    UF_print_syslog("\nBEGIN UFappRegister\n",0);
    VBmethod1 = method1;
    VBfunctionA = functionA;
    VBfunctionB = functionB;
    UF_print_syslog("END UFappRegister\n",0);
}

//============================================================================
*************************
// Two local functions to be called via wrappers from .NET
static int myFunction1(void)
{
```

```
    UF_initialize();
    UF_print_syslog("BEGIN myFunction1\n",0);

    //If registered callback into VBapp
    if (VBmethod1 != NULL) VBmethod1();

     UF_print_syslog("END myFunction1\n",0);

     return(77);
}
//************************************************************************
*************************
static int myFunction2(int value, char *string)
{
    int ret1,ret2;
    char message[MAX_STRING_SIZE] = "";

    ret1 = ret2 = 0;

    sprintf_s(message,"value:%d string:%s\n",value,string);

    UF_initialize();
    UF_print_syslog("BEGIN myFunction2\n",0);
    UF_print_syslog(message,0);

    //If registered callback into VBapp
    if (VBfunctionA != NULL) ret1 = VBfunctionA(value);
    if (VBfunctionB != NULL) ret2 = VBfunctionB(string);

    sprintf_s(message,"ret1:%d ret2:%d\n",ret1,ret2);
    UF_print_syslog(message,0);

    UF_print_syslog("END myFunction2\n",0);

    return(value+1);
}

//************************************************************************
*************************
// ufsta() - this is only for testing purposes,
// normally this DLL will loaded by the .NET DLL using the dllImport
construct
//
// Note: If this DLL needs to be unloaded before NX exists then the
.NET DLL will need
// to do it when it unloads.

extern void ufsta (char *param, int *retcode, int rlen)
{
```

```
    UF_initialize();
    UF_print_syslog("BEGIN UGapp (Running from File->Execute)\n",0);
    myFunction1();
    myFunction2(22,"Testing from File->Execute");
    UF_print_syslog("END UGapp\n",0);
}
//****************************************************************************
*************************
//END
```

Wrapper Class: Defines .NET signatures for VB methods to be called by UFapp

```
'****************************************************************************
******************
' wrappers to UFapp

Imports System
Imports System.Runtime.InteropServices

Public Class wrappers
    Const UFAPP_DLLNAME As String = "C:\local\UFapp"


    '************************************************************************
*****************
    ' signature definitions for the VB methods to be called by UFapp

    Public Delegate Sub method1_signature()
    Public Delegate Function functionA_signature(ByVal value As
Integer) As Integer
    Public Delegate Function functionB_signature(ByVal str As String)
As Integer


    '************************************************************************
*****************
    ' UFapp function, records methods/functions in VBapp to be called
from UFapp

    <DllImport(UFAPP_DLLNAME, EntryPoint:="UFappRegister", _

    CallingConvention:=CallingConvention.Cdecl)> _
    Shared Sub UFappRegister(ByVal method1 As method1_signature, _

ByVal function1 As functionA_signature, _

ByVal function2 As functionB_signature)
    End Sub

    '************************************************************************
*****************
```

```
        ' Hookup to a function in UFapp

        <DllImport(UFAPP_DLLNAME, EntryPoint:="UFappFunction1", _
        CallingConvention:=CallingConvention.Cdecl)> _

        Shared Function UFappFunction1() As Integer

        End Function

        '*************************************************************************
    ******************
        ' Hookup to another function in UFapp

        <DllImport(UFAPP_DLLNAME, EntryPoint:="UFappFunction2", _
        CallingConvention:=CallingConvention.Cdecl)> _

        Shared Function UFappFunction2(ByVal value As Integer,
        ByVal str As String) As Integer

        End Function
    End Class

    End Class
```

.NET

```
    Imports System
    Imports System.IO
    Imports System.Windows.Forms
    Imports NXOpen Imports NXOpen.UF
    Imports VBapp.wrappers

    '*************************************************************************
    *******
    ' Methods/Functions to be called by UFapp
    Class callbacks
        Public Shared Sub method1()
                theNXsession.LogFile.WriteLine("BEGIN method1")
                MessageBox.Show("hello from method1")
                theNXsession.LogFile.WriteLine("END method1")
        End Sub

        Public Shared Function functionA(ByVal value As Integer) As Integer
                theNXsession.LogFile.WriteLine("BEGIN functionA")
                theNXsession.LogFile.WriteLine(value.ToString)
                theNXsession.LogFile.WriteLine("END functionA")
                Return value + 1
        End Function

        Public Shared Function functionB(ByVal str As String) As Integer
                theNXsession.LogFile.WriteLine("BEGIN functionB")
```

```
            theNXsession.LogFile.WriteLine(str)
            theNXsession.LogFile.WriteLine("END functionB")
            Return 101
    End Function
End Class

'**********************************************************************
*********
' Standard Entry Points called from NX

Module VBapp
    Public theNXsession As Session
    Public theUFsession As UFSession

    Public Sub Main()
            Dim value As Integer = 88
            Dim ret1 As Integer = 0
            Dim ret2 As Integer = 0

            theNXsession = Session.GetSession
            theUFsession = UFSession.GetUFSession

            theNXsession.LogFile.WriteLine("BEGIN VBapp (Running from
File->Execute)")

            'Register VBapp callbacks that are used by UFapp
            UFappRegister(AddressOf callbacks.method1, _
                      AddressOf callbacks.functionA, _
                      AddressOf callbacks.functionB)
            ' Call Functions in UFapp which will callback to VBapp

            ret1 = UFappFunction1()
            ret2 = UFappFunction2(88, "This string is from VBapp")

            theNXsession.LogFile.WriteLine("funtions1:" & ret1.ToString
& " " & _
                                           "function2:" & ret2.ToString)
            theNXsession.LogFile.WriteLine("END VBapp")
    End Sub

    Public Function GetUnloadOption(ByVal arg As String) As Integer
            Return Session.LibraryUnloadOption.Immediately
    End Function

    Public Function UnloadLibrary(ByVal arg As String) As Integer
            Session.GetSession().LogFile.WriteLine("VBapp:
UnloadLibrary")
            Return 0
    End Function
```

```
End Module
```

User Interactions

Block Styler

Block Styler Automation

The Block Styler is an interactive tool for designing NX dialogs. Previous to the Block Styler, NX provided the User Interface Styler (UI Styler) for dialog creation. The Block Styler enhances the capabilities of the UI Styler by using the same set of reusable blocks currently used by internal NX applications. Using Blocks as the basic dialog building units ensures that NX has consistent behavior across all applications. By using the Block Styler your custom applications will share the same user interactions as used throughout NX. Furthermore, the Block Styler provides platform independent dialogs.

Note:

The UI Styler is still provided to support legacy user interfaces. However, it is recommended that the Block Styler should be used for all new user interfaces or major changes to existing dialogs.

Why Use Blocks

The UI Styler defines dialogs composed of individual controls such as push buttons and input fields. The predefined behaviors of these controls are very basic and can be used to implement any number of actions within NX. The Block Styler defines dialogs composed of Blocks of controls. Although there is a full set of basic blocks, there are also sets of blocks that provide standard NX behaviours for actions that are common in graphical CAD/CAM/CAE applications. For instance, there is a Specify Vector block that is composed of a label, a standard NX vector constructor button, a pull down list for standard inferred vector options and a reverse direction toggle. This block is used throughout NX to let the user select or interactively define a vector. By

using this block within your dialogs you save time by not having to implement the various options and you ensure that your application behaves as other NX applications.

Prerequisites

Before reading this section the reader is expected to already be familiar with building and running NX Open Journals and applications.

The reader is expected to understand how to use the Block Styler to create NX dialogs and how to save and edit dialogs that have been saved to .dlx files. Before attempting to connect a Block Dialog to an NX Open application the user is expected to understand how the block properties are used to control the dialog display and behavior options. Also, the user is expect to know how to generate automation template code for their target NX Open language.

To learn more about the Block Styler and how to use it to create NX dialogs and template code see the **Block Styler** user guide in the Automation section of the NX Help Library.

Fundamentals

When you are done creating your dialog using the Block Styler and you save your work the Block Styler creates the following two files.

1. The Block Dialog File

   This file has an extension of ".dlx". This manual will refer to this file as the DLX file. The DLX file contains all of the information that NX needs to be able to reproduce the dialog at runtime. The file will be input by NX and therefore must be placed in a standard location. For more information see the topic on launching dialogs from NX.

2. Template Code

   This file contains the template source code for the program that will be used to interact with the dialog at runtime. You will modify this file to provide the unique interactions required for your application. This code will contain the methods used to read and load the DLX file and all of the methods that will react to user and system events.

   The Block Styler provides various options to customize the template code. The details of how these options are used is discussed throughout this section. See the Block Styler users guide for a summary of the code generation options. One of the fundamental options is the selection of your target language. The Block Styler is capable of generating template code for all NX Open languages. .

Note:

This file is recreated each time you save your dialog. Once you modify the template code you should copy the working version to a new location to prevent your edits from being overwritten.

The template code will create a class using the name given to the DLX file. For examples if theDLX file name is "testBlockDialog.dlx", the template code will create a class named "testBlockDialog " and the self referencing object name will be "thetestBlockDialog"

What You Will Learn Here

The purpose of this section is to show how NX Open is used to interact at runtime with Block Styler dialogs. This section shows examples of how to modify the template code generated by the Block Styler for inclusion in your NX Open applications.

This section provides examples for the following topics.

Launching Block Styler Dialogs from NX - This topic shows the three different methods for loading and displaying a Block dialog from NX. It shows the standard entry points used to launch Block dialogs.
Block Styler Memory and Callbacks - This topic discusses the various callback methods that can be included in the template code. It discusses how the various callback methods are used to access dialog values and how dialog values interact with internal dialog memory. Examples of the callback methods can be found in the remaining sections.
Dialog and Block Properties - This topic discusses how to access properties that are used to control the display and behavior of the dialog. It shows examples of accessing typical block properties. It does not include a description of all block properties. Those descriptions can be found in the Block Dialog user guide.
Block Styler Update Callback - This topic focuses on the update callback. It shows examples of getting and setting typical Block properties that are used to interact with the user.
Block Styler Selection Blocks - This topic focuses on the specific types of Blocks that are used to implement selection. It shows examples of typical selection options and how to process a selection list.

Launching Block Styler Dialogs from NX

Launching an application that uses one or more Block dialogs is almost identical to launching any other Interactive NX Open application.The only difference is that your application will need to find the DLX file at runtime.

Note:

A Batch or Remote application would never be able to launch a Block dialog.

Finding the DLX File at Runtime

The template code comments include a section on launching the dialog. This section contains comments to remind you where to place the DLX for NX to be able to locate the file at runtime. Finding DLX files is also covered in How NX Finds Application Files. You have the following two options.

1. Place the DLX file in an **application** folder that is contained in a folder that is referenced by one of the search options discussed in How NX Finds Application Files. This is the recommended method and should definitely be used when distributing your application to multiple users.
2. Hard code the full path name in the source. The constructor method will contain a string constant that contains the DLX file name. By default the constant only contains the name of the file and the path is located using the standard NX search methods. If you change this constant to reference a specific path then the given path will be used. This method is only recommended during testing and is not recommended for production application.

Entry Point Options

When generating template code for your dialog you have the following two fundamental choices.

1. The template code can include a standard User Exit entry point which NX will call directly. This is the easiest method to use if your application is structured such that each executable only uses one dialog. The Block Styler gives you two options for generating User Exit template code: Menu and User Exit. These options are discussed below.
2. Or the template code can include a unique entry point that you will call from a larger application. This is most likely the method that you will use if your executable uses more than one dialog. The Block Styler provides the Callback Entry Point Option for this type of template code.

The Block Styler Code Generation options provide the following three Entry Point options.

Note:

You can chose to include any combination of these entry point options in the template code. However, if more than one option is selected, all options will be commented out. In this case you will need to uncomment the method that you want to use.

Note:

All code examples in this section are based on a dialog named: **testBlockDialog**

1. User Exit

   This option generates a standard User Exit entry point for the selected language. The default entry point generated in the template code is the entry point tied to the **File->Execute→NX Open** action. It is possible to change the entry point to another type of User Exit but not all User Exits will permit the display of a dialog. See User Exits for more information about the available User Exits and the appropriate entry points for each type of User Exit.

   Use this method if you want to launch your dialog using **File→Execute→NX Open**.

   Note:

   If the target language supports Journal playback you can also execute the template code as a Journal without having to create an executable using **Tools→Journal→Play**.

   User Exit Option - Language Specific Details

   NX Open for C++
   NX Open for .NET
   NX Open for Java

2. **Menu**

   This option generates the same User Exit entry point as above but comments are included to show how to define a menu button to execute the dialog code. For more information see Executing applications from existing menus and Executing applications from new menu items.

   For language specific details see the User Exit Options above.

3. **Callback**

   This option provides an entry point that you can call from a larger application.

   Callback Option - Language Specific Details

   NX Open for C++
   NX Open for .NET

NX Open for Java

NX Open for C++ - User Exit Entry Point Option

The standard User Exit entry point and dialog initialization for C++ is:

```
extern "C" DllExport void  ufusr(char *param, int *retcod, int
param_len)
{
    try
    {
        thetestBlockDialog = new testBlockDialog();
        // The following method shows the dialog immediately
        thetestBlockDialog->Show();
    }
    catch(exception & ex)
    {
        //---- Enter your exception handling code here -----
        testBlockDialog::theUI->NXMessageBox()->Show("Block Styler",
NXOpen::NXMessageBox::DialogTypeError, ex.what());
    }
    delete thetestBlockDialog;
}
```

The call to the show method will display the dialog and control will not return until the user closes the dialog or hits the Ok or Cancel buttons. All of your interaction with the dialog will be via the callback methods discussed in Block Styler Callbacks and Block Styler Update.
You can add code to this function but it is not necessary. A specific initialize callback is provided for dialog startup and a standard destructor method is provided for dialog shutdown and cleanup.

This entry point is used when you want to execute your dialog using *File→Execute→NX Open* or with a *Menu Button*.

NX Open for .NET - User Exit Entry Point Option

The standard User Exit entry point and dialog initialization for Visual Basic .NET is:

```
Public Shared Sub Main()
    Try
      thetestBlockDialog = New testBlockDialog()
      ' The following method shows the dialog immediately
      thetestBlockDialog.Show()
    Catch ex As Exception
      '---- Enter your exception handling code here -----
      theUI.NXMessageBox.Show("Block Styler",
NXMessageBox.DialogType.Error, ex.ToString)
    Finally
      thetestBlockDialog.Dispose()
    End Try
End Sub
```

The call to the show method will display the dialog and control will not return until the user closes the dialog or hits the Ok or Cancel buttons. All of your interaction with the dialog will be via the callback methods discussed in Block Styler Callbacks and Block Styler Update.

You can add code to this subroutine but it is not necessary. A specific initialize callback is provided for dialog startup and a standard dispose method is provided for dialog shutdown and cleanup.

This entry point is used when you want to execute your dialog using *File→Execute→NX Open* or with a *Menu Button* or with *Tools → Journal → Play*.

NX Open for Java - User Exit Entry Point Option

The standard User Exit entry point and dialog initialization for Jave is:

```
public static void main(String [] argv) throws Exception
{
   try
   {
     thetestBlockDialog = new testBlockDialog();
     // The following method shows the dialog immediately
     thetestBlockDialog.show();
   }
   catch(Exception ex)
   {
     //---- Enter your exception handling code here -----
     theUI.nxmessageBox().show("Block Styler",
nxopen.NXMessageBox.DialogType.INFORMATION, ex.getMessage());
   }
   finally
   {
     thetestBlockDialog.dispose();
   }
}
```

The call to the show method will display the dialog and control will not return until the user closes the dialog or hits the Ok or Cancel buttons. All of your interaction with the dialog will be via the callback methods discussed in Block Styler Callbacks and Block Styler Update.
You can add code to this function but it is not necessary. A specific initialize callback is provided for dialog startup and a standard dispose method is provided for dialog shutdown and cleanup.

This entry point is used when you want to execute your dialog using *File→Execute→NX Open* or with a *Menu Button*.

NX Open for C++ - Callback Entry Point Option

The callback entry point and dialog initialization in C++ is:

```
void testBlockDialog::Show_testBlockDialog()
{
  try
  {
    thetestBlockDialog = new testBlockDialog();
    // The following method shows the dialog immediately
    thetestBlockDialog->Show();
  }
  catch(exception& ex)
```

```
    {
      //---- Enter your exception handling code here -----
      testBlockDialog::theUI->NXMessageBox()->Show("Block Styler",
  NXOpen::NXMessageBox::DialogTypeError, ex.what());
    }
    delete thetestBlockDialog;
  }
```

The call to the show method will display the dialog and control will not return until the user closes the dialog or hits the Ok or Cancel buttons. All of your interaction with the dialog will be via the callback methods discussed in Block Styler Callbacks and Block Styler Update.

You can add code to this function but it is not necessary. A specific initialize callback is provided for dialog startup and a standard destructor method is provided for dialog shutdown and cleanup.

This entry point is used when you want to call the dialog directly from a larger application. This is typically done when multiple dialogs are required for the application. The Show_testBlockDialog() method is declared as a static function in the .hpp file so that it may be called directly without having to create a testBlockDialog object.

NX Open for .NET - Callback Entry Point Option

The callback entry point and dialog initialization in Visual Basic .NET is:

```
  Public Shared Sub Show_testBlockDialog()
      Try
        thetestBlockDialog = New testBlockDialog()
        ' The following method shows the dialog immediately
        thetestBlockDialog.Show()
      Catch ex As Exception
        '---- Enter your exception handling code here -----
        theUI.NXMessageBox.Show("Block Styler",
  NXMessageBox.DialogType.Error, ex.ToString)
      Finally
        thetestBlockDialog.Dispose()
      End Try
  End Sub
```

The call to the show method will display the dialog and control will not return until the user closes the dialog or hits the Ok or Cancel buttons. All of your interaction with the dialog will be via the callback methods discussed in Block Styler Callbacks and Block Styler Update.

You can add code to this subroutine but it is not necessary. A specific initialize callback is provided for dialog startup and a standard dispose method is provided for dialog shutdown and cleanup.

This entry point is used when you want to call the dialog directly from a larger application. This is typically done when multiple dialogs are required for the application. The Show_testBlockDialog() method is declared as a Shared function so that it may be called directly without having to create a testBlockDialog object.

NX Open for Java - Callback Entry Point Option

The syntax to reserve and release the solid modeling license in Java is:

```
public static void show_testBlockDialog()  throws NXException,
RemoteException
{
    try
    {
        thetestBlockDialog = new testBlockDialog();
        // The following method shows the dialog immediately
        thetestBlockDialog.show();
    }
    catch(Exception ex)
    {
        //---- Enter your exception handling code here -----
        theUI.nxmessageBox().show("Block Styler",
nxopen.NXMessageBox.DialogType.INFORMATION, ex.getMessage());
    }
    finally
    {
        thetestBlockDialog.dispose();
    }
}
```

The call to the show method will display the dialog and control will not return until the user closes the dialog or hits the Ok or Cancel buttons. All of your interaction with the dialog will be via the callback methods discussed in Block Styler Callbacks and Block Styler Update.
You can add code to this function but it is not necessary. A specific initialize callback is provided for dialog startup and a standard dispose method is provided for dialog shutdown and cleanup.

This entry point is used when you want to call the dialog directly from a larger application. This is typically done when multiple dialogs are required for the application. The show_testBlockDialog() method is declared as a static function so that it may be called directly without having to create a testBlockDialog object.

Block Styler Memory and Callbacks

Before generating template code you have the option to select the set of callback methods that you want to include in your application (see Code Generation in the Block Styler User Guide). A callback method is a method that is designed to handle specific types of dialog events. When the dialog is created at runtime the callback methods are registered with NX. While processing user interactions NX then knows to call your custom methods for the dialog. By adding code to the callback methods, you can implement the dialog behaviors which are required by your application.

This section defines the set of available callback methods and what events are used to invoke each method. Note that all dialogs must provide an *Apply* and *Initialize* method. All other callback methods are optional.

Your callback methods will access the current dialog values and may update the dialog values. Block Styler dialogs also have an internal memory. It is important to understand how dialog memory works, especially when trying to set initial dialog values. This section discusses dialog memory and how your callbacks interact with the dialog values.

Standard Navigator Buttons

Regardless of which callbacks are defined for a dialog all standard NX dialogs have an Ok, Apply and Cancel button. The standard behavior of these buttons are:

Ok - Execute an Apply action using the current dialog values and exit the dialog.

Apply - Execute the dialog actions using the current dialog values then reinitialize the dialog for a new edit or create.

Cancel - Exit the dialog without taking any further actions.

To ensure standard behavior the Apply callback is required. Furthermore, the template code for the Ok callback will include a call to the Apply callback.

Note:

If you do not generate an Ok callback then NX will execute your Apply callback automatically when the Ok button is hit.

Dialog Memory

Standard NX behavior is for all dialogs to remember the last values entered by the user and to initialize a dialog with those values when the dialog is used again. The only exception is if the dialog is being used to edit an existing NX object. In the case of an edit, the values used to create the object should be used to initialize the dialog values.

For Block Dialogs, dialog memory is automatically maintained and enforced by NX. The sections below describe when dialog memory values are set and when the memory values are used to initialize the current dialog values.

When a dialog is launched in the edit mode it is the NX Open programmer's responsibility to obtain the values from the object being edited and then use those values to establish the current dialog values.

Dialog Values

The following sections include a discussion of how the dialog values are maintained by the different callbacks. The following terms are used to describe the various values used to maintain the dialog values.

Current Dialog Values - These are the values that are shown to the user in the dialog. The user will modify these values and then hit the Ok or Apply button to use the values to create NX objects.

Default Values Given in the DLX File - These are the values that are defined when the dialog is created with Block Styler. Unless overridden at runtime by the NX Open programmer or by dialog memory values the DLX values will be the default dialog values.

Default Values Set by the Initialize Callback - These are values set by the NX Open programmer in the Initialize Callback. These values will override any DLX values. These values will be overridden by Dialog Memory values when the dialog is in Create mode. When the dialog is in edit mode the NX Open programmer should use the Initialize Callback to set the values of the objects that are being edited.

Dialog Memory Values - When the dialog is in Create mode these values are maintained automatically by NX. While in Create mode the dialog memory values are set equal to the current dialog values just after execution of the Ok or Apply callback. The current dialog values are set equal to the dialog memory values at the end of the dialog initiation process (just after the Initialize Callback). When the dialog is in Edit modes the dialog memory values are not changed and are not used to initialize the current dialog values.

Callbacks

Filter - This method is called whenever the user is selecting NX objects. It is used to let you filter out objects that you do not want to pre-select. To see an example of using the Filter callback see Block Styler Selection Blocks.
Update - This method is called to respond to most events from every block. For instance, if the user enters a text string into a String block and hits the Enter key, the Update method will be called. The code in this method can then determine which block generated the event and can react to the event. For instance, in the String example, the value of the string could be examined for proper format. For a complete description of the Update method and examples of typical usage see Block Styler Update Callback.
Ok - This method is called to respond to the user hitting the standard Ok navigator button. The standard behavior is for your application to execute an Apply using the current dialog values and then to exit the dialog. The template code will already contain a call to the Apply method, so examples of adding code to this callback are not given.

When the dialog is in Create mode the dialog memory values are set equal to the current dialog values just after the execution of the Ok callback.

Apply - This is a required callback. The Apply method is called to respond to the user hitting the standard Apply navigator button. The standard behavior is for your application to perform the designed actions (such as creating geometry) using the current dialog values but to leave the dialog open. The user is then able to enter new values and perform another Apply without exiting the dialog. An examples of this callback is given in: Block Styler Selection Blocks.
When the dialog is in Create mode the dialog memory values are set equal to the current dialog values just after the execution of the Apply callback.

Note:

After the Apply callback is executed the dialog is actually reinitialized. Therefore, the Initialize callback will be called after the Apply callback executes.

Cancel - This method is called to respond to the user hitting the standard Cancel navigator button. It is also called if the dialog closes for any other reasons such as the user hitting the

Close button in the dialog Rail Clip. The standard behavior is for your application to exit the dialog without making any further modifications to the NX data model. That is, an Apply should not be done but any previous Applies will not be undone.

DialogShown - This method is called just before the dialog is displayed (If you are transitioning from UI Styler - DialogShown() is similar to post constructor callback). Dialog values set in this callback replace dialog memory values for this instance of the dialog. Remember, dialog values set in the callback do not overwrite the dialog memory values unless you click OK or Apply on the dialog.

Initialize - This is a required callback. The Initialize method is called whenever the dialog is initialized, which is in the following three cases:

1. Just after the dialog is first constructed
2. After an Apply callback to reinitialize the dialog for the next create or edit
3. When the user hits the standard Reset button found on the dialog Rail Clip

How the current dialog values are initialized depends on the dialog edit/create mode and how the Initialize callback is invoked. The current dialog values will be set equal to the dialog memory values or to the default values defined in the DLX file and possibly overridden by the NX Open programmer at runtime. The following table shows how the current dialog values are set.

When Initialize Callback is Called	Create Mode Action	Edit Mode Action
After First Dialog Construction	All Current Dialog Values = Dialog Memory Values	Same as Reset Button
After Apply Callback	All Current Dialog Values = Dialog Memory Values	Same as Reset Button
When Reset Button is Hit	All Current Dialog Values = Default DLX File Values Specific Current Dialog Values = Default Values Set by the Initialize Callback	

The template code for the Initialize callback contains code to establish references to all blocks within the dialog. See the examples in Dialog and Block Properties to see how to use the block references to get and set current dialog values.

Dialog Value Example

As an example of how dialog memory and your callbacks are used to define the current dialog values consider the following example.

Consider a dialog with two string blocks named S1 and S2. Assume the following initial assignments are made to those string blocks.

String Block	Default Value Given in DLX File	Values Assigned by the Initialize Callback
S1	"dlx s1"	nothing - no SetString is made for S1
S2	"dlx s2"	"initial s2"

Now consider the following user actions.

1. When the dialog is first displayed S1="dlx s1" and S2="initial s2".
2. While in create mode the user changes S1 to "abc" and S2 to "xyz". The the user hits Ok which executes the Apply callback.
3. The next time the dialog is displayed in create mode S1="abc" and S2="xyz".
4. The next time the dialog is displayed in edit mode S1="dlx s1" and S2="initial s2". Note, the Initialize callback is really expected to query the object that is being edited and set S1 and S2 equal to the object's values. So really S1 and S2 will be whatever is assigned by the Initialize callback. But in this case S1 is not assigned so the DLX value is maintained and S1 is simply set to a constant.
5. While in edit mode the user changes the values to S1="123" and S2="456". The the user hits Ok which executes the Apply callback.
6. The next time the dialog is displayed in create mode the dialog will display S1="abc" and S2="xyz". Note, dialog memory is NOT changed when the dialog is in edit mode. In edit mode the object being edited is expected to contain the values required to initialize the dialog values.
7. The user hits the reset button on the dialog rail clip. The dialog will display S1="dlx s1" and S2="initial s2".

Sequence of Callbacks

Dialog callbacks are called in the following sequence:

1. Read the dlx file. Instantiate the UI blocks specified in the dlx file. Apply the settings contained in the dlx file to the UI blocks.
2. Call the Initialize callback
3. Apply the settings stored in dialog memory to the UI blocks
4. Call the DialogShown callback

When the dialog is shown using Edit mode, we do not do step 3.

Dialog and Block Properties

Every dialog and every block within a dialog have a set of properties which are used to customize and control the block's look and behavior at runtime. In the Block Styler the main dialog shows a tree in the "Blocks" window. The root of the tree references the Dialog. Expanding down from the root are all of the blocks that are in the dialog. Below the "Blocks" window is the "Properties" window. The Properties window shows all of the properties for the item that is selected in the tree. To display the Dialog properties, pick the root of the tree. To display the properties for a specific block, pick that block in the tree.

Every property is defined by it's Name, *Value* and value *Type*. For instance, a Dialog has a property named "Cue" which has a value of type String. The value of the Cue property is displayed in the NX Cue line when the dialog is active. The Name, Value and Type of each property is shown in the Properties window of the main Block Styler dialog.

To learn how the property values impact the look and behavior of a block see the complete list of properties in the Block Styler user's guide. This list also shows which properties are only available at dialog design time and cannot be accessed by NX Open. For the properties that can be accessed at runtime the list also shows the properties that are read only and those whose values can also be changed using NX Open.

Dialog and Block Object References

The following sections show how to use the dialog and block object references to access specific property values at runtime using NX Open. The template code includes variables that reference the dialog and block objects.

In the template code the dialog object is referenced by the variable named: "theDialog".

The references to all block objects are established in the Initialize callback. The text string used to identify the block is shown in the "Blocks" window of the main Block Styler dialog. The names are used to identify the nodes in the tree that is used to display and edit the structure of the dialog. By default these names are constructed from the type of block and a count, such as "button01" or "toggle04". Using these default names the following lines of pseudo code would then be included in the Initialize callback to establish a reference to the button and toggle block objects.

```
button01 = theDialog.TopBlock.FindBlock("button01")

toggle04 = theDialog.TopBlock.FindBlock("toggle04")
```

The variables named button01 and toggle04 will now reference their respective block objects.

To make these variable names more meaningful for your application change the names in the Block Styler before generating the template code. For instance, if you changed the name of the above toggle block to "blendOption" then the initialization line of code would be:

```
blendOption = theDialog.TopBlock.FindBlock("blendOption")
```

Property List

To access property values you first need to obtain a reference to the list of properties for the dialog or block. The GetProperties() method returns the current property list as follows:

```
dialogPropertyList = theDialog.TopBlock.GetProperties()

blockPropertyList = blockObject.GetProperties()
```

Warning:

Warning: the property list is dynamically generated and is only valid for the current dialog values. If the user changes the current dialog values then you must refresh the property list by again calling GetProperties(). To prevent memory leaks remember to free the memory used for the previous property list by using the dispose or delete methods (see language specific examples found in Block Styler Update Callback).

Get and Set Property Value Methods

To access and set property values at runtime you need to know the property's Name and it's value Type. Specific Get and Set methods are provided for each value type. The general form of the Get/Set methods are:

```
value = propertyList.Get<type>(<property name>)

propertyList.Set<type>(<property name>, ...)
```

For instance, to Get/Set a Dialog's Cue property you would use the methods:

```
dialogPropertyList = theDialog.TopBlock.GetProperties()

text = dialogPropertyList.GetString("Cue")

dialogPropertyList.SetString("Cue", "this is the text to display in the cue line")
```

Another example would be to Get/Set the current state of the above toggle block that has been renamed to blendOption. The current state of a toggle block is contained in the property named "Value". The following code would Get/Set the Value property:

```
togglePropertyList = blendOption.GetProperties()

togglePropertyList.SetLogical("Value", True)

toggleValue = togglePropertyList.GetLogical("Value")
```

Dialog and Block Properties - Language Specific Details

The following language specific code examples show how to access and set the following properties.

Dialog - Cue Line

String Block - Label

String Block - Current Value

String Block - Enable/Disable

Integer Block - Current Value

Additional examples of interacting with Dialog and Block Properties can be found in: Block Styler Update and Block Styler Selection.

NX Open for C++
NX Open for .NET
NX Open for Java

NX Open for C++ - Dialog and Block Properties

```
//Template code to establish block object references
  sourceString = theDialog->TopBlock()->FindBlock("sourceString");
  targetString = theDialog->TopBlock()->FindBlock("targetString");
  integer02 = theDialog->TopBlock()->FindBlock("integer02");

  //Obtain reference to the dialog properties
  PropertyList *dialogProp = theDialog->TopBlock()->GetProperties();
```

```
    //Set the dialogs cue line
    dialogProp->SetString("Cue", "this is the cue line");

    //Obtain reference to the String and Integer block properties
    PropertyList *sourceStringProp = sourceString->GetProperties();
    PropertyList *targetStringProp = targetString->GetProperties();
    PropertyList *integerProp = integer02->GetProperties();

    //Get the value and lable of the source string block
    NXOpen::NXString sourceText = sourceStringProp->GetString("Value");
    NXOpen::NXString sourceLabel = sourceStringProp-
>GetString("Label");

    //Set the value and label of the target string
    targetStringProp->SetString("Label", sourceText);
    targetStringProp->SetString("Value", sourceLabel);

    //Set the current value of the Integer block and then
    // use the current value to set source string value
    integerProp->SetInteger("Value", 5);

    char textBuff[25];
    sprintf_s(textBuff,"%d",integerProp->GetInteger("Value"));
    sourceStringProp->SetString("Value", textBuff);

    //Disable the target string block
    targetStringProp->SetLogical("Enable", false);

    //Cleanup memory
    delete sourceStringProp;
    delete targetStringProp;
    delete integerProp;
    delete dialogProp;
```

Note:

If a property list reference returned by GetProperties() is not delete then the memory used for the list will be lost until the dialog is closed.

NX Open for .NET - Dialog and Block Properties

```
    'Template code to establish block object references
   sourceString = theDialog.TopBlock.FindBlock("sourceString")
    targetString = theDialog.TopBlock.FindBlock("targetString")
    integer02 = theDialog.TopBlock.FindBlock("integer02")

    'Obtain reference to the dialog properties
    Dim dialogProp As PropertyList = theDialog.TopBlock.GetProperties()

    'Set the dialogs cue line
    dialogProp.SetString("Cue", "this is the cue line")
```

```
    'Obtain reference to the String and Integer block properties
    Dim sourceStringProp As PropertyList = sourceString.GetProperties()
    Dim targetStringProp As PropertyList = targetString.GetProperties()
    Dim integerProp As PropertyList = integer02.GetProperties()

    'Get the value and lable of the source string block
    Dim sourceText As String = sourceStringProp.GetString("Value")
    Dim sourceLabel As String = sourceStringProp.GetString("Label")

    'Set the value and label of the target string
    targetStringProp.SetString("Label", sourceText)
    targetStringProp.SetString("Value", sourceLabel)

    'Set the current value of the Integer block and then
    ' use the current value to set source string value
    integerProp.SetInteger("Value", 5)
    sourceStringProp.SetString("Value",
integerProp.GetInteger("Value").ToString)

    'Disable the target string block
    targetStringProp.SetLogical("Enable", False)
```

NX Open for Java - Dialog and Block Properties

```
    //Template code to establish block object references
    sourceString = theDialog.topBlock().findBlock("sourceString");
    targetString = theDialog.topBlock().findBlock("targetString");
    integer02 = theDialog.topBlock().findBlock("integer02");

    //Obtain reference to the dialog properties
    PropertyList dialogProp = theDialog.topBlock().getProperties();

    //Set the dialogs cue line
    dialogProp.setString("Cue", "this is the cue line");

    //Obtain reference to the String and Integer block properties
    PropertyList sourceStringProp = sourceString.getProperties();
    PropertyList targetStringProp = targetString.getProperties();
    PropertyList integerProp = integer02.getProperties();

    //Get the value and lable of the source string block
  String sourceText = sourceStringProp.getString("Value");
    String sourceLabel = sourceStringProp.getString("Label");

    //Set the value and label of the target string
    targetStringProp.setString("Label", sourceText);
    targetStringProp.setString("Value", sourceLabel);

    //Set the current value of the Integer block and then
    // use the current value to set source string value
```

```
integerProp.setInteger("Value", 5);

String numberText = "" + integerProp.getInteger("Value");
sourceStringProp.setString("Value",numberText);

//Disable the target string block
targetStringProp.setLogical("Enable", false);

//Cleanup memory
sourceStringProp.dispose();
targetStringProp.dispose();
integerProp.dispose();
dialogProp.dispose();
```

Block Styler Selection Blocks

Selection blocks define a consistent set of Block Styler Blocks that are used to enable interactive selection of many different kinds of common NX objects. The blocks simplify the selection process by providing predefined filters for common NX objects and consistent selection behavior. For instance, blocks exist to specifically select Faces, Curves/Edges, Features, Blend Faces or any general NX object. Common behaviors include the management of multiple selection lists, pre-selection, unselect, selection highlighting/unhighlighting, selection intent, object filtering, management of the Apply button for required object selections, interaction with the Model Navigator and access to the quick pick tool to resolve ambiguous selections.

Although many of the behaviors are providing automatically by the selection blocks, you are still able to programmatically refine the selection filter and to manage the selection list. This section discusses the common methods that are provided for all selection blocks.

Ok and Apply Button Management

If a dialog requires the user to select NX objects before the dialog action can be taken then the Ok and Apply buttons should not be active until the user has selected the required objects. Block Styler dialogs have the option to automatically manage the activation of the Ok and Apply buttons.

By default every selection block is considered to be a required user input. The Ok and Apply button will remain inactive until NX objects have been selected for all selection blocks. If a selection block is not required then when the dialog is designed with the Block Styler, set the "StepStatus" property to "Optional".

Selection List

Each selection block maintains it's own selection list. To access the selection list use the same methods discussed in Block Styler Properties. In this case the common property name for all selection blocks is "SelectedObjects" and the property type is a TaggedObjectVector. So in general the following method is used to obtain the list of currently selected objects for a given selection block:

```
      <list of selected objects> =
  propertyList.GetTaggedObjectVector("SelectedObjects")
```

As with other properties you can use the corresponding Set method to programmatically set the currently selected objects for a given block.

```
    propertyList.SetTaggedObjectVector("SelectedObjects", <list of
  selected objects>)
```

To clear the selected objects, just pass in an empty list to the Set method.

Note:

The "SelectedObjects" property will not be seen in the Block Styler properties window because the value cannot be set at dialog design time.

Examples of accessing the selection list for multiple selection blocks on a single dialog are given in this section.

Update Callback

As will other blocks any time a selection is made using a selection block a call is made to the Update callback and the reference of the selection block used to make the selection is passed to the Update callback. For more information and example of Update callbacks see Block Styler Update Callbacks.

Selection Filter Callback

Optionally you can have the Block Styler generate a selection filter callback. This callback permits you to filter out objects that you do not want selection intent to pre-highlight.

The selection filter callback will receive the block reference for the selection block that is currently being used by the user and a reference to the specific NX object that is being considered for pre-selection highlighting. Use the selection block reference to select the code to handle different filtering required for your different selection blocks. Use the NX object to determine if you want the user to be able to select that specific object. Any objects rejected by your selection filter callback will not pre-highlight for selection and cannot be selected by the user.

Examples of handling selection filters for multiple selection blocks on a single dialog are given in this section.

Other Selection Filters

It is also possible to set the same selection filters that are available using the Session's selection manager. The same options that are available to the SetSelectionMask() method found in the Selection class are available for selection blocks using the following method and property.

```
    <property list object>.SetSelectionFilter("SelectionFilter",
  <selection action option>, <selection mask array>)
```

The possible selection actions are:

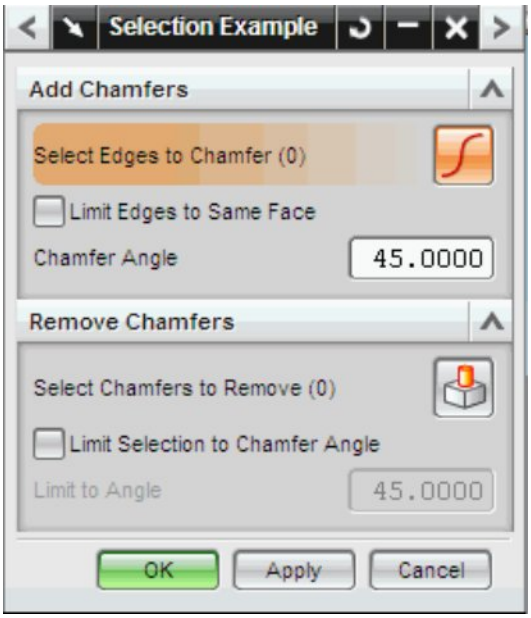
Selection Action Option	Action
Enable All	all selectable NX objects are enabled for selection (the mask array is not used and should be null)
Enable Specific	all object types listed in the mask array are added to the selection type filter (i.e. enabled for selection)

Disable Specific	all object types listed in the mask array are removed from the selection type filter (i.e. disabled for selection)
Clear And Enable Specific	the selection type filter is cleared and than all object types listed in the mask array are added to the selection type filter
All And Disable Specific	all objects are added to the selection type filter except those listed in the mask array

Each entry in the selection mask array defines the object type, subtype and solid body subtype of an object to either be enabled or disabled for selection.

Selection Blocks - Language Specific Details

The following language specific code examples show how to add code to the Block Styler template source for the following dialog.



This dialog contains two selection blocks and options for controlling selection filtering for those blocks.

The first selection block is used to select a set of edges which will be used during the Apply callback to create a chamfer feature. Associated with that block is a toggle. When the toggle is ON the selection filter callback will reject edges that do not belong to the same face. Also associated with the first selection block is an entry field used to set the angle for the chamfer features.

The second selection block is used to select existing features. The filter callback will only accept chamfer features. These features will be removed during the Apply callback. Associated with the second selection block is a toggle. When this toggle is ON the selection filter will only accept chamfers which have the angle given in a second input field that is enabled when the toggle is ON.

Also, when a selection filter toggle is turned ON then the associated selection list is cleared.

The following examples show how to implement the Update, Apply and Selection Filter callbacks to define the dialog actions defined above.

NX Open for C++
NX Open for .NET
NX Open for Java

NX Open for C++ - Selection Blocks

The following code was added to the standard source template generated by the Block Styler for the example dialog shown above.

Additional Include Files Added to the .hpp File

```
#include <NXOpen/Part.hxx>
#include <NXOpen/Face.hxx>
#include <NXOpen/Edge.hxx>
#include <NXOpen/EdgeTangentRule.hxx>
#include <NXOpen/Features_Chamfer.hxx>
#include <NXOpen/Features_FeatureBuilder.hxx>
#include <NXOpen/Features_ChamferBuilder.hxx>
#include <NXOpen/PartCollection.hxx>
#include <NXOpen/Features_FeatureCollection.hxx>
#include <NXOpen/ScCollector.hxx>
#include <NXOpen/ScCollectorCollection.hxx>
#include <NXOpen/ScRuleFactory.hxx>
```

New Property and Methods added to Class Definition in the .hpp File

```
Face *limitingFace; //face to limit edge selection

int faceFilter(Edge *selectedEdge);
int angleFilter(Chamfer *chamferFeature);
Face *sharedFace(Edge *edge1, Edge *edge2) ;

void addChamfer(Edge *selectedEdge, double angle);
void deleteObject(NXObject *selectedObject);
```

Additional Namespace

```
using namespace NXOpen::Features;
```

Initialize Callback - added property initialization

```
//Faced used to limit edge selection
limitingFace = NULL;
```

Apply Callback - used to add and remove chamfers

```
//------------------------------------------------------------------------
-----------
//Callback Name: apply_cb
//------------------------------------------------------------------------
-----------
int SelectionExample::apply_cb()
{
  try
```

```
    {
     //Access the required property lists
     PropertyList *angleProp = angleDouble->GetProperties();
     PropertyList *edgeListProp = edgeSelect->GetProperties();
     PropertyList *chamferListProp = chamferSelect->GetProperties();

//**************************************************************************
*******************
     // Add Chamfers to Selected Edges
     //Get the chamfer angle and edge selection list from the dialog
     double creationAngle = angleProp->GetDouble("Value");
     std::vector<TaggedObject *> edges = edgeListProp-
>GetTaggedObjectVector("SelectedObjects");
     // Add a chamfer to each selected edge
     for (int inx = 0;  inx < (int)edges.size(); ++inx)
     {
       addChamfer((Edge *)edges[inx], creationAngle);
     }

//**************************************************************************
*******************
     // Remove Selected Chamfers
     // Get the selected chamfers from the dialog
     std::vector<TaggedObject *> chamfers = chamferListProp-
>GetTaggedObjectVector("SelectedObjects");
     //Set an undo mark for update
     Session::UndoMarkId undoMark = theSession-
>SetUndoMark(Session::MarkVisibilityVisible, "Remove Chamfers");
     // Add the selected chamfers to the delete list
     for (int inx = 0; inx < (int)chamfers.size(); ++inx)
     {
       deleteObject((Features::Chamfer *) chamfers[inx]);
     }
     //Update the model to delete the chamfers
     int nErrs = theSession->UpdateManager()->DoUpdate(undoMark);
     // Report any errors - normally the error list should be scanned
and each error processed
     if (nErrs > 0)
     {
        char errMsg[25];
        sprintf_s(errMsg,"nErrs = %d",nErrs);
        SelectionExample::theUI->NXMessageBox()->Show("Update
Errors",NXMessageBox::DialogTypeError,errMsg);
     }
     //Cleanup memory
     delete angleProp;
     delete edgeListProp;
     delete chamferListProp;
    }
```

```
    catch(exception& ex)
    {
      //---- Enter your exception handling code here -----
      SelectionExample::theUI->NXMessageBox()->Show("Block Styler",
NXOpen::NXMessageBox::DialogTypeError, ex.what());
    }
    return 0;
  }
```

Update Callback

```
  //------------------------------------------------------------------------
----------
  //Callback Name: update_cb
  //------------------------------------------------------------------------
----------
  int SelectionExample::update_cb(NXOpen::BlockStyler::UIBlock* block)
  {
    try
    {
      if(block == edgeSelect)
      {
       PropertyList *faceTogglProp = faceToggle->GetProperties();
       //When face filtering is on, establish a limiting face after two
edges are selected
       if (faceTogglProp->GetLogical("Value"))
       {
         PropertyList *edgeListProp = edgeSelect->GetProperties();
         std::vector<TaggedObject *> edges = edgeListProp-
>GetTaggedObjectVector("SelectedObjects");
         if (edges.size() == 2)
         {
          limitingFace = sharedFace((Edge *)edges[0],(Edge *)edges[1]);
         }
         delete edgeListProp;
       }
       delete faceTogglProp;
      }
      else if(block == faceToggle)
      {
       PropertyList *faceTogglProp = faceToggle->GetProperties();
       //When the face filter is turned on...
       // Clear the current edge selection list
       if (faceTogglProp->GetLogical("Value"))
       {
         PropertyList *edgeListProp = edgeSelect->GetProperties();
         std::vector<TaggedObject *> edges;
         edgeListProp->SetTaggedObjectVector("SelectedObjects",edges);
         limitingFace = NULL;
         delete edgeListProp;
       }
```

```
        delete faceToggleProp;
       }
       else if(block == angleDouble)
       {
        //---------Enter your code here-----------
       }
       else if(block == chamferSelect)
       {
        //---------Enter your code here-----------
       }
       else if(block == angleToggle)
       {
         PropertyList *angleTogglProp = faceToggle->GetProperties();
         PropertyList *angleLimitProp = angleLimitDouble-
>GetProperties();
        //When the angle fileter is turned on...
        //  1. Clear the current chamfer selection list
        //  2. Also, enable/disable the angle limit entry field
        if (angleTogglProp->GetLogical("Value"))
        {
          PropertyList *chamferListProp = chamferSelect-
>GetProperties();
          std::vector<TaggedObject *> chamfers;
          chamferListProp-
>SetTaggedObjectVector("SelectedObjects",chamfers);
          angleLimitProp->SetLogical("Enable", TRUE);
          delete chamferListProp;
        }
        else
        {
          angleLimitProp->SetLogical("Enable", FALSE);
        }
        delete angleTogglProp;
        delete angleLimitProp;
       }
       else if(block == angleLimitDouble)
       {
        // When the angle limit value changes...
        // Clear the current chamfer selection list
        PropertyList *chamferListProp = chamferSelect->GetProperties();
       std::vector<TaggedObject *> chamfers;
        chamferListProp-
>SetTaggedObjectVector("SelectedObjects",chamfers);
        delete chamferListProp;
       }
    }
    catch(exception& ex)
    {
        //---- Enter your exception handling code here -----
```

```
        SelectionExample::theUI->NXMessageBox()->Show("Block Styler",
NXOpen::NXMessageBox::DialogTypeError, ex.what());
    }
    return 0;
  }
```

Filter Callback

```
  //------------------------------------------------------------------------
----------
  //Callback Name: filter_cb
  //------------------------------------------------------------------------
----------
  int SelectionExample::filter_cb(NXOpen::BlockStyler::UIBlock*  block,
NXOpen::TaggedObject* selectedObject)
  {
   int accept = UF_UI_SEL_ACCEPT;
   if (block == edgeSelect)
   {
    PropertyList *faceTogglePropp = faceToggle->GetProperties();
    //Edge Select Filter
    if (faceTogglePropp->GetLogical("Value"))
    {
      accept = faceFilter((Edge *)selectedObject);
    }
    delete faceTogglePropp;
   }
   else if (block == chamferSelect)
   {
    //Feature Selection Filter - limit selection to chamfer features
    Feature *featureObject = (Feature *)selectedObject;
    if (strcmp(featureObject->FeatureType().GetText(),"CHAMFER") == 0)
    {
      accept = angleFilter((Chamfer *)featureObject);
    }
    else
    {
      accept = UF_UI_SEL_REJECT;
    }
   }
   return accept;
  }
```

New Methods - added to support filtering and feature create/delete

```
//************************************************************************
**********************************
  //FACE FILTER
  int SelectionExample::faceFilter(Edge *selectedEdge)
  {
    int accept = UF_UI_SEL_REJECT;
    //If a limiting face has not been established...
```

```
    // then use the first two selected edges to establish the limiting
face
    // otherwise use the limiting face
    if (limitingFace == NULL)
    {
     //Limit faces to those of the first selected edge
     PropertyList *edgeListProp = edgeSelect->GetProperties();
     std::vector<TaggedObject *> edges = edgeListProp-
>GetTaggedObjectVector("SelectedObjects");
     if (edges.size() < 1)
     {
       accept = UF_UI_SEL_ACCEPT;
     }
     else if (edges.size() == 1)
     {
       if (sharedFace((Edge *)edges[0],selectedEdge) != NULL) accept =
UF_UI_SEL_ACCEPT;
     }
     delete edgeListProp;
    }
    else
    {
     //Limit the edges to just those of the limiting face
     std::vector<Face *> faceArray = selectedEdge->GetFaces();
     for (int inx = 0; inx < (int)faceArray.size(); ++inx)
     {
      if (faceArray[inx] == limitingFace)
      {
        accept = UF_UI_SEL_ACCEPT;
        break;
      }
     }
    }
    return accept;
  }

//**************************************************************************
**********************************
  //ANGLE FILTER
  int SelectionExample::angleFilter(Chamfer *chamferFeature)
  {
   int accept = UF_UI_SEL_ACCEPT;
   PropertyList * angleToggleProp = angleToggle->GetProperties();
   //If the angle filter toggle is ON...
   // then only accept chamfers of the limiting angle
   if (angleToggleProp->GetLogical("Value"))
   {
    PropertyList *angleLimitProp = angleLimitDouble->GetProperties();
    Part *workPart = theSession->Parts()->Work();
```

```
    ChamferBuilder *chamferBuilder;
    chamferBuilder = workPart->Features()-
>CreateChamferBuilder(chamferFeature);
    if (chamferBuilder->AngleExp()->Value() != angleLimitProp-
>GetDouble("Value"))
    {
      accept = UF_UI_SEL_REJECT;
    }
    chamferBuilder->Destroy();
    delete angleLimitProp;
   }
   delete angleToggleProp;
   return accept;
  }

//**************************************************************************
**********************************
  //SHARED FACE - return face shared between two edges
  Face *SelectionExample::sharedFace(Edge *edge1, Edge *edge2)
  {
   Face *foundFace = NULL;
   std::vector<Face *> faceArray1 = edge1->GetFaces();
   std::vector<Face *> faceArray2 = edge2->GetFaces();
   for (int inx1 = 0;  inx1 < (int)faceArray1.size(); ++inx1)
   {
    for (int inx2 = 0; inx2 < (int)faceArray2.size(); ++inx2)
    {
     if (faceArray1[inx1] == faceArray2[inx2])
     {
       foundFace = faceArray1[inx1];
       break;
     }
    }
    if (foundFace != NULL) break;
   }
   return foundFace;
  }

//**************************************************************************
**********************************
  // ADD CHAMFER FEATURE of given angle to given edge
  void SelectionExample::addChamfer(Edge *selectedEdge, double angle)
{
   try
   {
     Part *workPart = theSession->Parts()->Work();
     Features::Feature *nullFeature = NULL;
     Features::ChamferBuilder *chamferBuilder1;
```

```
     chamferBuilder1 = workPart->Features()-
>CreateChamferBuilder(nullFeature);
     chamferBuilder1-
>SetOption(ChamferBuilder::ChamferOptionOffsetAndAngle);
     chamferBuilder1-
>SetMethod(ChamferBuilder::OffsetMethodEdgesAlongFaces);
     chamferBuilder1->SetFirstOffset("20");
     chamferBuilder1->SetSecondOffset("20");
     char angleText[25];
     sprintf_s(angleText,"%lf",angle);
     chamferBuilder1->SetAngle(angleText);
     Edge *nullEdge = NULL;
     EdgeTangentRule *edgeTangentRule1;
     edgeTangentRule1 = workPart->ScRuleFactory()-
>CreateRuleEdgeTangent(selectedEdge, nullEdge, false, 0.5, false);
     std::vector<SelectionIntentRule *> rules1(1);
     rules1[0] = edgeTangentRule1;
     ScCollector *scCollector1;
     scCollector1 = workPart->ScCollectors()->CreateCollector();
     scCollector1->ReplaceRules(rules1, false);
     chamferBuilder1->SetSmartCollector(scCollector1);
     Features::Feature *feature1;
     feature1 = chamferBuilder1->CommitFeature();
     chamferBuilder1->Destroy();
   }
   catch(exception& ex)
   {
    SelectionExample::theUI->NXMessageBox()->Show("Error Adding
Chamfer", NXOpen::NXMessageBox::DialogTypeError, ex.what());
   }
  }

//**************************************************************************
**********************************
  // DELETE OBJECT - add the given object to the delete list
  void SelectionExample::deleteObject(NXObject *selectedObject)
  {
   try
   {
      std::vector<NXObject *> obj(1);
      obj[0] = selectedObject;
      int nErrs = theSession->UpdateManager()->AddToDeleteList(obj);
      //Report any errors - normally the error list should be scanned
and each error processed
      if (nErrs > 0)
      {
        char errMsg[25];
        sprintf_s(errMsg,"nErrs = %d",nErrs);
```

```
        SelectionExample::theUI->NXMessageBox()->Show("Error Adding To
Delete List",NXMessageBox::DialogTypeError,errMsg);
      }
    }
    catch(exception& ex)
    {
     SelectionExample::theUI->NXMessageBox()->Show("Error Removing
Chamfers", NXOpen::NXMessageBox::DialogTypeError, ex.what());
    }
  }
```

Note:

If a property list reference returned by GetProperties() is not delete then the memory used for the list will be lost until the dialog is closed.

NX Open for .NET - Selection Blocks

The following code was added to the standard source template generated by the Block Styler for the example dialog shown above.

Additional Name Space

```
    Imports NXOpen.Features
```

New Object Properties

```
    Private limitingFace As Face = Nothing 'face to limit edge selection
```

Initialize Callback - added property initialization

```
    limitingFace = Nothing
```

Apply Callback - used to add and remove chamfers

```
    '------------------------------------------------------------------------
----------
  'Callback Name: apply_cb
  '------------------------------------------------------------------------
----------
  Public Function apply_cb() As Integer
    Try

'************************************************************************
******************
      'Add Chamfers to Selected Edges
      'Get the chamfer angle and edge selection list from the dialog
      Dim creationAngle =
angleDouble.GetProperties().GetDouble("Value")
      Dim edges() As TaggedObject =
edgeSelect.GetProperties.GetTaggedObjectVector("SelectedObjects")
      'Add a chamfer to each selected edge
      For Each selectedEdge As TaggedObject In edges
        addChamfer(CType(selectedEdge, Edge), creationAngle)
      Next selectedEdge
```

```
'**********************************************************************
******************
      'Remove Selected Chamfers
      'Get the selected chamfers from the dialog
      Dim chamfers() As TaggedObject =
chamferSelect.GetProperties.GetTaggedObjectVector("SelectedObjects")
      'Set an undo mark for update
      Dim undoMark As Session.UndoMarkId
      undoMark = theSession.SetUndoMark(Session.MarkVisibility.Visible,
"Remove Chamfers")
      'Add the selected chamfers to the delete list
      For Each chamferObject As TaggedObject In chamfers
          deleteObject(CType(chamferObject, Features.Chamfer))
      Next chamferObject
      'Update the model to delete the chamfers
      Dim nErrs As Integer =
theSession.UpdateManager.DoUpdate(undoMark)
      'Report any errors - normally the error list should be scanned
and each error processed
      If nErrs > 0 Then
         theUI.NXMessageBox.Show("Update Errors",
NXMessageBox.DialogType.Error, "nErrs = " & nErrs.ToString)
      End If
    Catch ex As Exception
      '---- Enter your exception handling code here -----
      theUI.NXMessageBox.Show("Apply Error",
NXMessageBox.DialogType.Error, ex.ToString)
    End Try
    apply_cb = 0
  End Function
```

Update Callback

```
  '----------------------------------------------------------------------
----------
  'Callback Name: update_cb
  '----------------------------------------------------------------------
----------
  Public Function update_cb(ByVal block As NXOpen.BlockStyler.UIBlock)
As Integer
    Try
      If block Is edgeSelect Then
         'When face filtering is on, establish a limiting face after
two edges are selected
         If faceToggle.GetProperties().GetLogical("Value") Then
            Dim edges() As TaggedObject =
edgeSelect.GetProperties.GetTaggedObjectVector("SelectedObjects")
            If (edges.Length = 2) Then
               limitingFace = sharedFace(CType(edges(0), Edge),
CType(edges(1), Edge))
```

```
            End If
         End If
      ElseIf block Is faceToggle Then
        'When the face filter is turned on...
        '  Clear the current edge selection list
        If faceToggle.GetProperties().GetLogical("Value") Then
           Dim edges(-1) As TaggedObject

edgeSelect.GetProperties().SetTaggedObjectVector("SelectedObjects",
edges)
           limitingFace = Nothing
        End If
      ElseIf block Is angleDouble Then
        '---- Enter your code here -----
      ElseIf block Is chamferSelect Then
        '---- Enter your code here -----
      ElseIf block Is angleToggle Then
        'When the angle fileter is turned on...
        '  1. Clear the current chamfer selection list
        '  2. Also, enable/disable the angle limit entry field
        If angleToggle.GetProperties().GetLogical("Value") Then
           Dim chamfers(-1) As TaggedObject

chamferSelect.GetProperties().SetTaggedObjectVector("SelectedObjects",
chamfers)
           angleLimitDouble.GetProperties().SetLogical("Enable", True)
        Else
           angleLimitDouble.GetProperties().SetLogical("Enable", False)
        End If
      ElseIf block Is angleLimitDouble Then
        ' When the angle limit value changes...
        '    Clear the current chamfer selection list
        Dim chamfers(-1) As TaggedObject

chamferSelect.GetProperties().SetTaggedObjectVector("SelectedObjects",
chamfers)
      End If
    Catch ex As Exception
      '---- Enter your exception handling code here -----
      theUI.NXMessageBox.Show("Block Styler",
NXMessageBox.DialogType.Error, ex.ToString)
    End Try
    update_cb = 0
  End Function
```

Filter Callback

```
  '------------------------------------------------------------------------
----------
  'Callback Name: filter_cb
```

```
  '-------------------------------------------------------------------
----------
  Public Function filter_cb(ByVal block As NXOpen.BlockStyler.UIBlock,
ByVal selectedObject As NXOpen.TaggedObject) As Integer
    Dim accept As Integer = NXOpen.UF.UFConstants.UF_UI_SEL_ACCEPT
    If block Is edgeSelect Then
       'Edge Select Filter
       If faceToggle.GetProperties().GetLogical("Value") Then
          accept = faceFilter(CType(selectedObject, Edge))
       End If
    ElseIf block Is chamferSelect Then
       'Feature Selection Filter - limit selection to chamfer features
       Dim featureObject As Feature = CType(selectedObject, Feature)
       If featureObject.FeatureType = "CHAMFER" Then
          accept = angleFilter(CType(featureObject, Features.Chamfer))
       Else
          accept = NXOpen.UF.UFConstants.UF_UI_SEL_REJECT
       End If
    End If
    filter_cb = accept
  End Function
```

New Methods - added to support filtering and feature create/delete

```
'*********************************************************************
**********************************
  'FACE FILTER
  Function faceFilter(ByVal selectedEdge As Edge) As Integer
    Dim accept As Integer = NXOpen.UF.UFConstants.UF_UI_SEL_REJECT
    'If a limiting face has not been established...
    '  then use the first two selected edges to establish the limiting
face
    '  otherwise use the limiting face
    If limitingFace Is Nothing Then
       'Limit faces to those of the first selected edge
       Dim edges() As TaggedObject =
edgeSelect.GetProperties.GetTaggedObjectVector("SelectedObjects")
       If edges.Length < 1 Then
          accept = NXOpen.UF.UFConstants.UF_UI_SEL_ACCEPT
       ElseIf edges.Length = 1 Then
          If sharedFace(CType(edges(0), Edge), selectedEdge) IsNot
Nothing Then
             accept = NXOpen.UF.UFConstants.UF_UI_SEL_ACCEPT
          End If
       End If
    Else
       'Limit the edges to just those of the limiting face
       For Each faceObject As Face In selectedEdge.GetFaces()
         If faceObject Is limitingFace Then
            accept = NXOpen.UF.UFConstants.UF_UI_SEL_ACCEPT
            Exit For
```

```
          End If
        Next faceObject
    End If
    faceFilter = accept
  End Function

'************************************************************************
**********************************
  'ANGLE FILTER
  Function angleFilter(ByVal chamferFeature As Features.Chamfer) As
Integer
    Dim accept As Integer = NXOpen.UF.UFConstants.UF_UI_SEL_ACCEPT
    'If the angle filter toggle is ON...
    '  then only accept chamfers of the limiting angle
    If angleToggle.GetProperties().GetLogical("Value") Then
       Dim workPart As Part = theSession.Parts.Work
       Dim chamferBuilder As Features.ChamferBuilder
       chamferBuilder =
workPart.Features.CreateChamferBuilder(chamferFeature)
       If chamferBuilder.AngleExp.Value <>
angleLimitDouble.GetProperties().GetDouble("Value") Then
          accept = NXOpen.UF.UFConstants.UF_UI_SEL_REJECT
       End If
       chamferBuilder.Destroy()
    End If
    angleFilter = accept
  End Function
'************************************************************************
**********************************
  'SHARED FACE - return face shared between two edges
  Function sharedFace(ByVal edge1 As Edge, ByVal edge2 As Edge) As Face
    Dim foundFace As Face = Nothing
    For Each face1 As Face In edge1.GetFaces()
      For Each face2 As Face In edge2.GetFaces()
        If face1 Is face2 Then
           foundFace = face1
           Exit For
        End If
      Next face2
      If foundFace IsNot Nothing Then Exit For
    Next face1
    Return foundFace
  End Function
'************************************************************************
**********************************
  'ADD CHAMFER FEATURE of given angle to given edge
  Sub addChamfer(ByVal selectedEdge As Edge, ByVal angle As Double)
    Try
       Dim workPart As Part = theSession.Parts.Work
```

```
        Dim nullFeatures_Feature As Features.Feature = Nothing
        Dim chamferBuilder1 As Features.ChamferBuilder
        chamferBuilder1 =
workPart.Features.CreateChamferBuilder(nullFeatures_Feature)
        chamferBuilder1.Option =
Features.ChamferBuilder.ChamferOption.OffsetAndAngle
        chamferBuilder1.Method =
Features.ChamferBuilder.OffsetMethod.EdgesAlongFaces
        chamferBuilder1.FirstOffset = "20"
        chamferBuilder1.SecondOffset = "20"
        chamferBuilder1.Angle = angle.ToString
        Dim nullEdge As Edge = Nothing
        Dim edgeTangentRule1 As EdgeTangentRule
        edgeTangentRule1 =
workPart.ScRuleFactory.CreateRuleEdgeTangent(selectedEdge, nullEdge,
False, 0.5, False)       Dim rules1(0) As SelectionIntentRule
        rules1(0) = edgeTangentRule1
        Dim scCollector1 As ScCollector
        scCollector1 = workPart.ScCollectors.CreateCollector()
        scCollector1.ReplaceRules(rules1, False)
        chamferBuilder1.SmartCollector = scCollector1
        Dim feature1 As Features.Feature
        feature1 = chamferBuilder1.CommitFeature()
        chamferBuilder1.Destroy()
    Catch ex As Exception
        theUI.NXMessageBox.Show("Error Adding Chamfer",
NXMessageBox.DialogType.Error, ex.ToString)
    End Try
  End Sub

'**********************************************************************
**********************************
  'DELETE OBJECT - add the given object to the delete list
  Private Sub deleteObject(ByVal selectedObject As NXObject)
    Try
        Dim obj(0) As NXObject
        obj(0) = selectedObject
        Dim nErrs As Integer =
theSession.UpdateManager.AddToDeleteList(obj)
        'Report any errors - normally the error list should be scanned
and each error processed
        If nErrs > 0 Then
           theUI.NXMessageBox.Show("Add To Delete Errors",
NXMessageBox.DialogType.Error, "nErrs = " & nErrs.ToString)
         End If
     Catch ex As Exception
         theUI.NXMessageBox.Show("Error Removing Chamfer",
NXMessageBox.DialogType.Error, ex.ToString)
     End Try
```

```
End Sub
```

NX Open for Java - Selection Blocks

The following code was added to the standard source template generated by the Block Styler for the example dialog shown above.

Additional Imports

```
import nxopen.features.*;
```

New Property

```
private Face limitingFace; //face to limit edge selection
```

Initialize Callback - added property initialization

```
//Faced used to limit edge selection
limitingFace = null;
```

Apply Callback - used to add and remove chamfers

```
  //------------------------------------------------------------------------
----------
  //Callback Name: apply
  //Following callback is associated with the "theDialog" Block.
  //------------------------------------------------------------------------
----------
  public int apply() throws NXException, RemoteException
  {
    try
    {
       //Access the required property lists
       PropertyList angleProp = angleDouble.getProperties();
       PropertyList edgeListProp = edgeSelect.getProperties();
       PropertyList chamferListProp = chamferSelect.getProperties();

//************************************************************************
*******************
       // Add Chamfers to Selected Edges
       //Get the chamfer angle and edge selection list from the dialog
double creationAngle = angleProp.getDouble("Value");
       TaggedObject[] edges =
edgeListProp.getTaggedObjectVector("SelectedObjects");
       // Add a chamfer to each selected edge
       for (int inx = 0;  inx < edges.length; ++inx)
       {
         addChamfer((Edge)edges[inx], creationAngle);
       }

//************************************************************************
*******************
      // Remove Selected Chamfers
      // Get the selected chamfers from the dialog
```

```
      TaggedObject[] chamfers =
chamferListProp.getTaggedObjectVector("SelectedObjects");
      //Set an undo mark for update
      int undoMark =
theSession.setUndoMark(Session.MarkVisibility.VISIBLE, "Remove
Chamfers");
      // Add the selected chamfers to the delete list
      for (int inx = 0; inx < chamfers.length; ++inx)
      {
        deleteObject((Chamfer) chamfers[inx]);
      }
      //Update the model to delete the chamfers
      int nErrs = theSession.updateManager().doUpdate(undoMark);
      // Report any errors - normally the error list should be scanned
and each error processed
      if (nErrs > 0)
      {
        String errMsg = "nErrs = " + nErrs;
        theUI.nxmessageBox().show("Update
Errors",nxopen.NXMessageBox.DialogType.INFORMATION,errMsg);
       }
        //Cleanup memory
        angleProp.dispose();
        edgeListProp.dispose();
        chamferListProp.dispose();
    }
    catch(Exception ex)
    {
        //---- Enter your exception handling code here -----
        theUI.nxmessageBox().show("Block Styler",
nxopen.NXMessageBox.DialogType.INFORMATION, ex.getMessage());
    }
    return 0;
  }
```

Update Callback

```
  //------------------------------------------------------------------------
-----------
  //Callback Name: update
  //Following callback is associated with the "theDialog" Block.
  //------------------------------------------------------------------------
-----------
  public int update( nxopen.blockstyler.UIBlock block) throws
NXException, RemoteException
  {
    try
    {
       if(block == edgeSelect)
       {
         PropertyList faceToggleProp = faceToggle.getProperties();
```

```
         //When face filtering is on, establish a limiting face after
two edges are selected
         if (faceToggleProp.getLogical("Value"))
         {
            PropertyList edgeListProp = edgeSelect.getProperties();
            TaggedObject[] edges =
edgeListProp.getTaggedObjectVector("SelectedObjects");
            if (edges.length == 2)
            {
              limitingFace = sharedFace((Edge)edges[0],(Edge)edges[1]);
            }
            edgeListProp.dispose();
         }
         faceToggleProp.dispose();
       }
       else if(block == faceToggle)
       {
         PropertyList faceToggleProp = faceToggle.getProperties();
         //When the face filter is turned on...
         // Clear the current edge selection list
         if (faceToggleProp.getLogical("Value"))
         {
            PropertyList edgeListProp = edgeSelect.getProperties();
            TaggedObject[] edges = new TaggedObject[0];

edgeListProp.setTaggedObjectVector("SelectedObjects",edges);
            limitingFace = null;
            edgeListProp.dispose();
         }
         faceToggleProp.dispose();
       }
       else if(block == angleDouble)
       {
         //---------Enter your code here-----------
       }
       else if(block == chamferSelect)
       {
         //---------Enter your code here-----------
       }
       else if(block == angleToggle)
       {
         PropertyList angleToggleProp = angleToggle.getProperties();
         PropertyList angleLimitProp =
angleLimitDouble.getProperties();
         //When the angle filter is turned on...
         //  1. Clear the current chamfer selection list
         //  2. Also, enable/disable the angle limit entry field
         if (angleToggleProp.getLogical("Value"))
         {
```

```
            PropertyList chamferListProp =
chamferSelect.getProperties();
            TaggedObject[] chamfers = new TaggedObject[0];

chamferListProp.setTaggedObjectVector("SelectedObjects",chamfers);
            angleLimitProp.setLogical("Enable",true);
            chamferListProp.dispose();
          }
          else
          {
            angleLimitProp.setLogical("Enable", false);
          }
          angleTogglProp.dispose();
          angleLimitProp.dispose();
        }
        else if(block == angleLimitDouble)
        {
          // When the angle limit value changes...
          // Clear the current chamfer selection list
          PropertyList chamferListProp = chamferSelect.getProperties();
TaggedObject[] chamfers = new TaggedObject[0];

chamferListProp.setTaggedObjectVector("SelectedObjects",chamfers);
          chamferListProp.dispose();
        }
    }
    catch(Exception ex)
    {
        //---- Enter your exception handling code here -----
        theUI.nxmessageBox().show("Block Styler",
nxopen.NXMessageBox.DialogType.INFORMATION, ex.getMessage());
    }
    return 0;
  }
```

Filter Callback

```
  //------------------------------------------------------------------------
-----------
  //Callback Name: filter
  //Following callback is associated with the "theDialog" Block.
  //------------------------------------------------------------------------
-----------
  public int filter(nxopen.blockstyler.UIBlock block,
nxopen.TaggedObject selectedObject) throws NXException, RemoteException
  {
    int accept = nxopen.uf.UFConstants.UF_UI_SEL_ACCEPT;
    if (block == edgeSelect)
    {
     PropertyList faceToggleProp = faceToggle.getProperties();
      //Edge Select Filter
```

```
      if (faceToggleProp.getLogical("Value"))
      {
       accept = faceFilter((Edge)selectedObject);
      }
      faceToggleProp.dispose();
    }
    else if (block == chamferSelect)
    {
      //Feature Selection Filter - limit selection to chamfer features
      Feature featureObject = (Feature)selectedObject;
      if (featureObject.featureType().equals("CHAMFER"))
      {
       accept = angleFilter((Chamfer)featureObject);
      }
      else
      {
       accept = nxopen.uf.UFConstants.UF_UI_SEL_REJECT;
      }
    }
    return accept;
  }
```

New Methods - added to support filtering and feature create/delete

```
//**************************************************************************
**********************************
  //FACE FILTER
  private int faceFilter(Edge selectedEdge)  throws NXException,
RemoteException
  {
    int accept = nxopen.uf.UFConstants.UF_UI_SEL_REJECT;
    //If a limiting face has not been established...
    // then use the first two selected edges to establish the limiting
face
    // otherwise use the limiting face
    if (limitingFace == null)
    {
      //Limit faces to those of the first selected edge
      PropertyList edgeListProp = edgeSelect.getProperties();
      TaggedObject edges[] =
edgeListProp.getTaggedObjectVector("SelectedObjects");
      if (edges.length < 1)
      {
        accept = nxopen.uf.UFConstants.UF_UI_SEL_ACCEPT;
      }
      else if (edges.length == 1)
      {
       if (sharedFace((Edge)edges[0],selectedEdge) != null) accept =
nxopen.uf.UFConstants.UF_UI_SEL_ACCEPT;
      }
```

```
     edgeListProp.dispose();
    }
    else
    {
      //Limit the edges to just those of the limiting face
      Face faceArray[] = selectedEdge.getFaces();
      for (int inx = 0; inx < faceArray.length; ++inx)
      {
        if (faceArray[inx] == limitingFace)
        {
          accept = nxopen.uf.UFConstants.UF_UI_SEL_ACCEPT;
          break;
        }
      }
    }
   return accept;
  }

//**************************************************************************
**********************************
  //ANGLE FILTER
  private int angleFilter(Chamfer chamferFeature) throws NXException,
RemoteException
  {
    int accept = nxopen.uf.UFConstants.UF_UI_SEL_ACCEPT;
    PropertyList angleToggleProp = angleToggle.getProperties();
    //If the angle filter toggle is ON...
    // then only accept chamfers of the limiting angle
    if (angleToggleProp.getLogical("Value"))
    {
      PropertyList angleLimitProp = angleLimitDouble.getProperties();
      Part workPart = theSession.parts().work();
      ChamferBuilder chamferBuilder;
      chamferBuilder =
workPart.features().createChamferBuilder(chamferFeature);
      if (Double.compare(chamferBuilder.angleExp().value(),
angleLimitProp.getDouble("Value")) !=0)
      {
        accept = nxopen.uf.UFConstants.UF_UI_SEL_REJECT;
      }
      chamferBuilder.destroy();
      angleLimitProp.dispose();
    }
   angleToggleProp.dispose();
   return accept;
  }

//**************************************************************************
**********************************
```

```
  //SHARED FACE - return face shared between two edges
  private Face sharedFace(Edge edge1, Edge edge2)  throws NXException,
RemoteException
  {
    Face foundFace = null;
    Face faceArray1[] = edge1.getFaces();
    Face faceArray2[] = edge2.getFaces();
    for (int inx1 = 0;  inx1 < faceArray1.length; ++inx1)
    {
      for (int inx2 = 0; inx2 < faceArray2.length; ++inx2)
      {
         if (faceArray1[inx1] == faceArray2[inx2])
         {
           foundFace = faceArray1[inx1];
           break;
         }
      }
      if (foundFace != null) break;
    }
   return foundFace;
  }
//**************************************************************************
**********************************
  //ADD CHAMFER FEATURE of given angle to given edge
  private void addChamfer(Edge selectedEdge, double angle) throws
NXException, RemoteException
  {
    try
    {
       Part workPart = theSession.parts().work();
       Feature nullFeature = null;
       ChamferBuilder chamferBuilder1;
       chamferBuilder1 =
workPart.features().createChamferBuilder(nullFeature);

chamferBuilder1.setOption(ChamferBuilder.ChamferOption.OFFSET_AND_ANGLE
);

chamferBuilder1.setMethod(ChamferBuilder.OffsetMethod.EDGES_ALONG_FACES
);
       chamferBuilder1.setFirstOffset("20");
       chamferBuilder1.setSecondOffset("20");
       String angleText = "" + angle;
       chamferBuilder1.setAngle(angleText);
       Edge nullEdge = null;
       EdgeTangentRule edgeTangentRule1;
       edgeTangentRule1 =
workPart.scRuleFactory().createRuleEdgeTangent(selectedEdge, nullEdge,
false, 0.5, false);
```

```
        SelectionIntentRule[] rules1 = new SelectionIntentRule[1];
       rules1[0] = edgeTangentRule1;
        ScCollector scCollector1;
        scCollector1 = workPart.scCollectors().createCollector();
        scCollector1.replaceRules(rules1, false);
        chamferBuilder1.setSmartCollector(scCollector1);
        Feature feature1;
        feature1 = chamferBuilder1.commitFeature();
        chamferBuilder1.destroy();
    }
    catch(Exception ex)
    {
        theUI.nxmessageBox().show("Error Adding Chamfer",
nxopen.NXMessageBox.DialogType.INFORMATION, ex.getMessage());
    }
  }
//**************************************************************************
**********************************
  //DELETE OBJECT - add the given object to the delete list
  private void deleteObject(NXObject selectedObject) throws
NXException, RemoteException
  {
    try
    {
      NXObject[] obj = new NXObject[1];
      obj[0] = selectedObject;
      int nErrs = theSession.updateManager().addToDeleteList(obj);
      //Report any errors - normally the error list should be scanned
and each error processed
      if (nErrs > 0)
      {
         String errMsg = "nErrs = " + nErrs;
         theUI.nxmessageBox().show("Error Adding To Delete
List",nxopen.NXMessageBox.DialogType.INFORMATION,errMsg);
      }
    }
    catch(Exception ex)
    {
        theUI.nxmessageBox().show("Error Removing Chamfer",
nxopen.NXMessageBox.DialogType.INFORMATION, ex.getMessage());
    }
  }
```

Note:

If a property list reference returned by getProperties() is not disposed then the memory used for the list will be lost until the dialog is closed.

Block Styler Update Callback

The Update callback is called anytime the user changes the value of any dialog block. By adding your own code to this callback you can customize the runtime behavior of your dialog. This section shows how to determine which block triggered the call to the Update callback and shows examples of how to access and set property values during the update callback. Other update examples can be found in Block Styler Selection Blocks.

The Update callback is an optional callback. Many dialogs simply collect values and options from a user and do not require the NX Open program to actively respond to specific user inputs. In these cases the Apply callback will simply access the current dialog property values to determine the desired actions.

Note:

To reduce the amount of coding required to manage a dialog and to enforce consistent behavior throughout NX, many interaction between the user and a dialog are handled automatically by NX and do not generate calls to the Update callback.

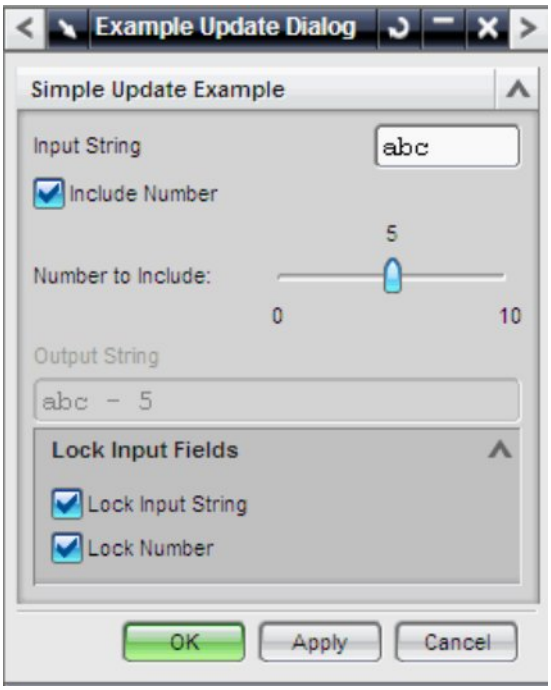
Determining Which Block Called the Update Callback

The Update callback is passed a reference to the block that was changed and thus produced the update event. To determine which block caused the even, just compare the block object that is passed as input to the Update callback to the block object that is established in the Initialize callback (see Dialog and Block Properties for a discussion of the block objects).

If an Update callback is requested when the dialog is saved then the template code for the Update callback will include an IF-THEN-ELSE block that is doing the necessary compares to determine which block causes the update event. To customize the dialogs actions just add your desired code to the block within the IF-THEN-ELSE that matches the block actions you are customizing.

Update Callback - Language Specific Details

The following language specific code examples show an Update callback that has been customized to manage the following dialog.



This dialog is used to build a string that is shown in the middle of the dialog in the string block labeled "Output String". The output string is a result of concatenating the input string with a number set by a slider bar. The number is optionally included in the output string depending on the state of the "Include Number" toggle block.

The Update callback shown in the examples has added code the IF-THEN-ELSE block that was contained in the original template code generated by the Block Styler. The IF-THEN-ELSE block contains a section to handle events from all of the blocks in the dialog. Each section will perform the following operation. Note that a method (updateOutputString) has been added by the programmer to perform the operation of building the output string and updating the output string block in the dialog. This method is also shown in the examples.

Block Update Event	Action
Input String	Call updateOutputString()
Include Number Toggle	Call updateOutputString()
Number	Call updateOutputString()
Output String	Nothing (this block is disabled and is only used for display)
Lock Input String Toggle	Enable or disable the input string block
Lock Number Toggle	Enable or disable the number block

NX Open for C++
NX Open for .NET
NX Open for Java

NX Open for C++ - Dialog and Block Properties

```
 //------------------------------------------------------------------------
------------
// updateOutputString() - copies the input string to the output string
and
// optionally concatenates the number to the output string
 //------------------------------------------------------------------------
------------
void updateExample::updateOutputString()
{
 try
 {
  //Obtain the block's property lists
  NXOpen::BlockStyler::PropertyList *inputStringProp   = inputString-
>GetProperties();
  NXOpen::BlockStyler::PropertyList *outputStringProp  = outputString-
>GetProperties();
```

```
  NXOpen::BlockStyler::PropertyList *includeNumberProp = includeNumber-
>GetProperties();
  NXOpen::BlockStyler::PropertyList *numberProp        = number-
>GetProperties();
  //Get the input string and establish an empty number string
  NXOpen::NXString inputText = inputStringProp->GetString("Value");
  NXOpen::NXString numberText = "";
  //If the include number toggle is ON (true)
  // then get the number and create a string for it
  if (includeNumberProp->GetLogical("Value"))
  {
   int numberValue = numberProp->GetInteger("Value");
   char textBuff[25];
   sprintf_s(textBuff,"%d",numberValue);
   numberText += " - ";
   numberText += textBuff;
  }
  //Set the output string to the concatenated input string and optional
number string
  outputStringProp->SetString("Value", inputText + numberText);
  //Clean up memory
  delete inputStringProp;
  delete outputStringProp;
  delete includeNumberProp;
  delete numberProp;
 }
 catch (exception &ex)
 {
  updateExample::theUI->NXMessageBox()->Show("Error Updating Output
String", NXOpen::NXMessageBox::DialogTypeError, ex.what());
 }
}
 //------------------------------------------------------------------------
----------
//Callback Name: update_cb
 //------------------------------------------------------------------------
----------
int updateExample::update_cb(NXOpen::BlockStyler::UIBlock* block)
{
 try
 {
  //Get the block's property lists
  NXOpen::BlockStyler::PropertyList *inputStringProp   = inputString-
>GetProperties();
  NXOpen::BlockStyler::PropertyList *numberProp        = number-
>GetProperties();
  NXOpen::BlockStyler::PropertyList *lockStringProp = lockString-
>GetProperties();
```

```
  NXOpen::BlockStyler::PropertyList *lockNumberProp = lockNumber-
>GetProperties();
  //Any changes to the input string, number, or the include number
toggle
  // could result in a new output string
  if (block == inputString)
  {
   updateOutputString();
  }
  else if (block == includeNumber)
  {
   updateOutputString();
  }
  else if (block == number)
  {
   updateOutputString();
  }
  else if (block == outputString)
  {
   //The output string block is disabled and only used for display
  }
  else if (block == lockString)
  {
   //Enable or disable the input string block based on the state
   //of the string lock toggle block
   inputStringProp->SetLogical("Enable", !lockStringProp-
>GetLogical("Value"));
  }
  else if (block == lockNumber)
  {
   //Enable or disable the number block based on the state
   //of the number lock toggle block
   numberProp->SetLogical("Enable", !lockNumberProp-
>GetLogical("Value"));
  }
  //Free memory
  delete inputStringProp;
  delete numberProp;
  delete lockStringProp;
  delete lockNumberProp;
 }
 catch (exception &ex)
 {
  //---- Enter your exception handling code here -----
  updateExample::theUI->NXMessageBox()->Show("Block Styler",
NXOpen::NXMessageBox::DialogTypeError, ex.what());
 }
 return 0;
}
```

Note:

If the property list reference returned by GetProperties() is not delete then the memory used for the list will be lost until the dialog is closed.

NX Open for .NET - Dialog and Block Properties

```
    '------------------------------------------------------------------
--------------
    ' updateOutputString() - copies the input string to the output
string and
    '                        optionally concatenates the number to the
output string
    '------------------------------------------------------------------
--------------
    Public Sub updateOutputString()
        Try
            'Get the input string and establish an empty number string
            Dim inputText As String =
inputString.GetProperties().GetString("Value")
            Dim numberText As String = ""
            'If the include number toggle is ON (true)
            '  then get the number and create a string for it
            If includeNumber.GetProperties().GetLogical("Value") Then
                Dim numberValue As Integer =
number.GetProperties().GetInteger("Value")
                numberText = " - " & numberValue.ToString
            End If
            'Set the output string to the concatenated input string and
optional number string
            outputString.GetProperties().SetString("Value", inputText &
numberText)
        Catch ex As Exception
            theUI.NXMessageBox.Show("Error Updating Output String",
NXMessageBox.DialogType.Error, ex.ToString)
        End Try
    End Sub
    '------------------------------------------------------------------
------------
    'Callback Name: update_cb
    '------------------------------------------------------------------
------------
    Public Function update_cb(ByVal block As
NXOpen.BlockStyler.UIBlock) As Integer
        Try
            'Any changes to the input string, number, or the include
number toggle
            ' could result in a new output string
            If block Is inputString Then
                updateOutputString()
            ElseIf block Is includeNumber Then
```

```
                updateOutputString()
            ElseIf block Is number Then
                updateOutputString()
            ElseIf block Is outputString Then
                'The output string block is disabled and only used for
display
            ElseIf block Is lockString Then
                'Enable or disable the input string block based on the
state
                'of the string lock toggle block
                inputString.GetProperties().SetLogical("Enable", _
                      Not
lockString.GetProperties().GetLogical("Value"))
            ElseIf block Is lockNumber Then
                'Enable or disable the number block based on the state
                'of the number lock toggle block
                number.GetProperties().SetLogical("Enable", _
                      Not
lockNumber.GetProperties().GetLogical("Value"))
            End If
        Catch ex As Exception
            '---- Enter your exception handling code here -----
            theUI.NXMessageBox.Show("Block Styler",
NXMessageBox.DialogType.Error, ex.ToString)
        End Try
        update_cb = 0
    End Function
```

Note:

.NET automatically handles memory management required for the property lists returned by GetProperties().

NX Open for Java - Dialog and Block Properties

```
  //------------------------------------------------------------------------
-------------
  //updateOutputString() - copies the input string to the output string
and
  //optionally concatenates the number to the output string
  //------------------------------------------------------------------------
-------------
  void updateOutputString()  throws NXException, RemoteException
  {
    try
    {
      //Obtain the block's property lists
      PropertyList inputStringProp   = inputString.getProperties();
      PropertyList outputStringProp  = outputString.getProperties();
      PropertyList includeNumberProp = includeNumber.getProperties();
      PropertyList numberProp        = number.getProperties();
      //Get the input string and establish an empty number string
```

```
      String inputText = inputStringProp.getString("Value");
      String numberText = "";
      //If the include number toggle is ON (true)
      // then get the number and create a string for it
      if (includeNumberProp.getLogical("Value"))
      {
        int numberValue = numberProp.getInteger("Value");
        numberText = " - " + numberValue;
      }
      //Set the output string to the concatenated input string and
optional number string
      outputStringProp.setString("Value", inputText + numberText);
      //Clean up memory
      inputStringProp.dispose();
      outputStringProp.dispose();
      includeNumberProp.dispose();
      numberProp.dispose();
    }
    catch (Exception ex)
    {
      theUI.nxmessageBox().show("Error Updating Output String",
nxopen.NXMessageBox.DialogType.INFORMATION, ex.getMessage());    }
  }
  //------------------------------------------------------------------------
-----------
  //Callback Name: update
  //------------------------------------------------------------------------
-----------
  public int update( nxopen.blockstyler.UIBlock block) throws
NXException, RemoteException
  {
    try
    {
      //Get the block's property lists
      PropertyList inputStringProp = inputString.getProperties();
      PropertyList numberProp      = number.getProperties();
      PropertyList lockStringProp  = lockString.getProperties();
      PropertyList lockNumberProp  = lockNumber.getProperties();
     //Any changes to the input string, number, or the include number
toggle
     // could result in a new output string
     if (block == inputString)
     {
       updateOutputString();
     }
     else if (block == includeNumber)
     {
       updateOutputString();
     }
```

```
        else if (block == number)
        {
          updateOutputString();
        }
        else if (block == outputString)
        {
          //The output string block is disabled and only used for display
        }
        else if (block == lockString)
        {
          //Enable or disable the input string block based on the state
          //of the string lock toggle block
          inputStringProp.setLogical("Enable",
!lockStringProp.getLogical("Value"));
        }
        else if (block == lockNumber)
        {
          //Enable or disable the number block based on the state
          //of the number lock toggle block
          numberProp.setLogical("Enable",
!lockNumberProp.getLogical("Value"));
        }
        //Free memory
        inputStringProp.dispose();
        numberProp.dispose();
        lockStringProp.dispose();
        lockNumberProp.dispose();
      }
      catch(Exception ex)
      {
         theUI.nxmessageBox().show("Block Styler",
nxopen.NXMessageBox.DialogType.INFORMATION, ex.getMessage());
      }
      return 0;
    }
```

UI Styler

The UI styler is a tool to design NX dialogs interactively. Starting NX6, Block Styler with many added capabilities is available for third-party developers interested in designing NX style dialogs.

UI styler guide explains step by step details on how to create a dialog using the User Interface Styler application. File→Save on the dialog also saves automatically generated code to interact with the dialog in the desired automation language.

Overview

This sections lists the steps one should follow to embed a UI styler dialog in an interactive application. For detailed information each step, refer the UI styler guide.

1. Create the dialog with UI Styler interactive tools - See Building a Dialog chapter in the UI styler guide
2. Associate dialog items with callback - Callbacks are added to dialog at design time interactively in UI styler. Callbacks depend on type of dialog item, a push button item will have a activate callback, a radio box will have a value changed callback. Give your own callback name, this callback will be registered with the dialog and the code generation will create a place holder where you add your own logic for the callback.
3. Save the dialog - This also saves the template code in the desired language
4. Copy the dialog file to appropriate location - see How NX finds application files
5. Add your code to all dialog item callbacks.
6. Add your code to dialog callbacks like apply, OK, cancel, post constructor
7. Associate the dialog with a menu button or define a user exit for dialog execution - If the dialog is invoked directly through the menu button, then register the dialog first with menu file using RegisterWithUiMenu() method on the dialog.
8. Compile and link your application
9. Invoke the dialog through a menu button or perform the interactive NX action which will invoke the user exit

Selection in UI Styler

You can enable selection for your UI styler dialog by setting up selection filters and callbacks in the dialog constructor callback. All UI styler dialogs provide access to a selection handle. You can set up filter to allow selection of only certain types of objects and then register a selection callback to collect all valid selected objects and act on them.

When a UI styler dialog is active and the user moves the cursor over any displayable object on screen, the filter callback registered with the dialog is called. The filter callback uses the selection mask you set up in the constructor and provides the possible selectable object for processing. You can decide if the object is valid for your application and accept it or reject it. If you accept the object, then the registered selection callback is invoked for further processing.

The code snippet below sets up a selection mask to allow edges. The filter callback filters out non-linear edges and only accepts linear edges as valid objects.

Also see the example code in UGII_BASE_DR\ugopen\NXOpenSampleApplications\<language>\Selection_UIStyler

Selection Code Snippet - Language Specific details

VB .NET
C#
Java
C++

VB .NET

```
'Get the selection Handle for this dialog
selectH = changeDialog.GetSelectionHandle()

'Set up Selection Mask - Allow only solid edges
```

```
Dim selectionMask_array(0) As NXOpen.Selection.MaskTriple
With selectionMask_array(0)
            .Type = NXOpen.UF.UFConstants.UF_solid_type
            .Subtype = NXOpen.UF.UFConstants.UF_solid_edge_subtype
            .SolidBodySubtype =
NXOpen.UF.UFConstants.UF_UI_SEL_FEATURE_ANY_EDGE
End With

'Following sets the Selection mask for Edge
UI.GetUI().SelectionManager.SetSelectionMask(selectH,
NXOpen.Selection.SelectionAction.ClearAndEnableSpecific,
selectionMask_array)

'Following sets the Selection and Filter callbacks which are invoked
during selection
UI.GetUI().SelectionManager.SetSelectionCallbacks(selectH, AddressOf
filter_cb, AddressOf sel_cb)

'Filter callback to filter out non-linear edges and select only linear
ones
Public Function filter_cb(ByVal selectedObject As NXOpen.NXObject,
ByVal selectionMask_array As NXOpen.Selection.MaskTriple, ByVal
selectHandle As SelectionHandle) As Integer
                  Try
                          If (edge.SolidEdgeType=Edge.EdgeType.Linear)
Then filter_cb = NXOpen.UF.UFConstants.UF_UI_SEL_ACCEPT
                          Else
                                  filter_cb =
NXOpen.UF.UFConstants.UF_UI_SEL_REJECT
                          End If
                  Catch ex As NXException
                          ' ---- Enter your exception handling code
here -----
                          MsgBox(ex.Message)
                  End Try
End Function

'Following is Selection Callback
            Public Function sel_cb(ByVal selectedObjects As NXObject(),
ByVal deselectedObjects() As NXObject, ByVal selectHandle As
SelectionHandle) As Integer
                  Try
                          For Each selObj As NXObject In
selectedObjects
                          '----do something----
                          Next
                          For Each deselObj As NXObject In
deselectedObjects
                                  '-----do something-----
```

```
                        Next

                    Catch ex As NXException
                        ' ---- Enter your exception handling code
here -----
                        MsgBox(ex.Message)
                    End Try
                    'Continue the dialog
                    sel_cb = NXOpen.UIStyler.DialogState.ContinueDialog
            End Function
```

C#

```
 //Get the selection Handle for this dialog
selectH = changeDialog.GetSelectionHandle();

//Set up Selection Mask - Allow only solid edges
NXOpen.Selection.MaskTriple[] selectionMask_array = new
NXOpen.Selection.MaskTriple[1];
{
            selectionMask_array(0).Type =
NXOpen.UF.UFConstants.UF_solid_type;
            selectionMask_array(0).Subtype =
NXOpen.UF.UFConstants.UF_solid_edge_subtype;
            selectionMask_array(0).SolidBodySubtype =
NXOpen.UF.UFConstants.UF_UI_SEL_FEATURE_ANY_EDGE;
}

//Following sets the Selection mask for Edge
UI.GetUI().SelectionManager.SetSelectionMask(selectH,
NXOpen.Selection.SelectionAction.ClearAndEnableSpecific,
selectionMask_array);

//Following sets the Selection and Filter callbacks which are invoked
during selection
UI.GetUI().SelectionManager.SetSelectionCallbacks(selectH, filter_cb,
sel_cb);

            //Filter callback to filter out non-linear edges and select
only linear ones
            public int filter_cb(NXOpen.NXObject selectedObject,
NXOpen.Selection.MaskTriple selectionMask_array, SelectionHandle
selectHandle)
             {
                    int functionReturnValue = 0;
                    try {
                            if ((edge.SolidEdgeType ==
Edge.EdgeType.Linear)) {
                                    functionReturnValue =
NXOpen.UF.UFConstants.UF_UI_SEL_ACCEPT;
                            }
```

```
                        else {
                                functionReturnValue =
NXOpen.UF.UFConstants.UF_UI_SEL_REJECT;
                        }
                }
                catch (NXException ex) {
                        // ---- Enter your exception handling code
here -----
                }
                return functionReturnValue;
        }
        //Following is Selection Callback

        public int sel_cb(NXObject[] selectedObjects, NXObject[]
deselectedObjects, SelectionHandle selectHandle)
        {
                try {
                        foreach (NXObject selObj in selectedObjects)
{
                                //----do something----
                        }
                        foreach (NXObject deselObj in
deselectedObjects) {
                                //-----do something-----
                        }
                }

                catch (NXException ex) {
                        // ---- Enter your exception handling code
here -----
                }
                return NXOpen.UIStyler.DialogState.ContinueDialog;
                //Continue the dialog
    }
```

Java

```
SelectionHandle selectH=CHANGEDialog.getSelectionHandle();
Selection.MaskTriple selection_MaskArray[]=new Selection.MaskTriple[1];
                selection_MaskArray[0] = new Selection.MaskTriple();

    selection_MaskArray[0].type=UFConstants.UF_solid_type;

    selection_MaskArray[0].subtype=UFConstants.UF_solid_body_subtype;

    selection_MaskArray[0].solidBodySubtype=UFConstants.UF_UI_SEL_FEATU
RE_ANY_EDGE;

// Following sets the Selection mask for Edge
theUI.selectionManager().setSelectionMask(selectH,Selection.SelectionAc
tion.CLEAR_AND_ENABLE_SPECIFIC,selection_MaskArray);
```

```
// Following sets the Selection and Filter callbacks which are invoked
during selection
theUI.selectionManager().setSelectionCallbacks(selectH,this,this);

public int selectionCallback(NXObject[] selectedObjects, NXObject[]
deSelectedObjects, SelectionHandle selectH) throws NXException,
RemoteException
{
                    try{

                                    if(selectedObjects!=null)
                                    {
                                                    for(int
i=0;i<selectedObjects.length;i++)
                                                    {
                                    //do something
                                                    }
                                    }
                                    if(deSelectedObjects!=null)
                                    {
                                                    for(int
j=0;j<deSelectedObjects.length;j++)
                                                    {
                                    //do something
                                                    }
                                    }
                            }
                            catch(NXException ex)
                            {
                                    // ---- Enter your exception handling
code here -----
                                    theUI.nxmessageBox().show("UI
Styler", nxopen.NXMessageBox.DialogType.ERROR, ex.getMessage());
                            }
                            return 0;
}
// Following is Filter Callback - This function gets invoked during
selection.
// Here, we can put a logic to accept or reject the selected entities

public int filterCallback(NXObject selectedObject, MaskTriple
selectionMask, SelectionHandle arg2) throws NXException,
RemoteException
{

    Edge edge = (Edge)SelectedObject;
    if(edge.solidEdgeType==Edge.edgeType.LINEAR)
    {
```

```
                return UFConstants.UF_UI_SEL_ACCEPT;
        }
        else
        { return UFConstants.UF_UI_SEL_REJECT;
        }
    }
```

C++

```
    // Get the selection handle
        NXOpen::SelectionHandle *selectH = changeDialog-
    >GetSelectionHandle();

    //Setup the Selection Mask
        std::vector<NXOpen::Selection::MaskTriple> selectionMask_array;
        NXOpen::Selection::MaskTriple
        selectionMask_arrayElem;
        selectionMask_arrayElem.Type = UF_solid_type;
        selectionMask_arrayElem.Subtype = UF_solid_edge_subtype;
        selectionMask_arrayElem.SolidBodySubtype =
    UF_UI_SEL_FEATURE_ANY_EDGE;
    selectionMask_array.push_back(selectionMask_arrayElem);

    //Set the Selection Mask in the SelectionManager

        UI::GetUI()->SelectionManager()->SetSelectionMask(selectH,
    NXOpen::Selection::SelectionActionClearAndEnableSpecific,
    selectionMask_array);

    //Set the Selection callbacks in the SelectionManager
        UI::GetUI()->SelectionManager()->SetSelectionCallbacks(selectH,
    make_callback(&filter_cb), make_callback(&sel_cb));
    //------------------------------------------------------------------
    ---------
    //------------------------- UIStyler Callback Functions ---------------
    ---------
    //------------------------------------------------------------------
    ---------

    //------------------------------------------------------------------
    ---------
    // Callback Name: sel_cb
    // Following callback is associated with the "changeDialog" Styler
    item.
    // Input:
    // 1. selectedObject - vector of selected object
    // 2. deselectedObjects - vector of deselected object
    // 3. selectHandle SelectionHandle
    //------------------------------------------------------------------
    ---------
```

```
int sel_cb(std::vector<NXOpen::NXObject *> selectedObject,
std::vector<NXOpen::NXObject *> deselectedObjects,
NXOpen::SelectionHandle* selectHandle)
{
    try
    {
            std::vector<NXOpen::NXObject *>::iterator selIter;
            for (selIter = selectedObject.begin(); selIter !=
selectedObject.end(); selIter++)
            {
                    //do something
            }
            for (selIter = deselectedObjects.begin(); selIter !=
deselectedObjects.end(); selIter++)
            {
                    //do something
            }
    }
    catch (const NXOpen::NXException& ex)
    {
            // ---- Enter your exception handling code here -----
            UIStylerSelectionExample::theUI->NXMessageBox()->Show("UI
Styler", NXOpen::NXMessageBox::DialogTypeError, ex.Message());
    }
    // Callback acknowledged, do not terminate dialog
    // A return value of NXOpen::UIStyler::DialogStateExitDialog will
not be accepted
    // for this callback type. You must respond to your apply button.
    return NXOpen::UIStyler::DialogStateContinueDialog;
}
//------------------------------------------------------------------------
---------
// Callback Name: filter_cb
// Input:
// 1. selectedObject - pointer to NXOpen::NXObject object
// 2. selectionMask_array - NXOpen::Selection::MaskTriple object
// 3. selectHandle SelectionHandle
//------------------------------------------------------------------------
---------
int filter_cb(NXOpen::NXObject *selectedObject,
NXOpen::Selection::MaskTriple* selectionMask_array,
NXOpen::SelectionHandle* selectHandle)
{
    try
    {
            Edge edge = (Edge)SelectedObject;
            if(edge.SolidEdgeType==Edge.EdgeTypeLinear)
                    return UF_UI_SEL_REJECT;
            else
```

```
                return UF_UI_SEL_ACCEPT;
        }
        catch (const NXOpen::NXException& ex)
        {
                // ---- Enter your exception handling code here -----
                UIStylerSelectionExample::theUI->NXMessageBox()->Show("UI
Styler", NXOpen::NXMessageBox::DialogTypeError, ex.Message());
        }
        // Callback acknowledged, do not terminate dialog
        // A return value of NXOpen::UIStyler::DialogStateExitDialog will
not be accepted
        // for this callback type. You must respond to your apply button.
        return NXOpen::UIStyler::DialogStateContinueDialog;
}
```

Microsoft Windows Forms

Using Winforms with NX

To write traditional GUI applications using Microsoft .NET you'll use Windows Forms. Windows Forms are a style of application built around classes in the .NET Framework. They have a programming model all their own that is cleaner, more robust, and more consistent than models based on the Win32 API or MFC, and they run in the managed environment of the .NET Common Language Runtime (CLR).

NX Open .NET applications can use WinForms to create the UI which works with NX but should heed to following suggestions:

.NET programming allows for two different type of form methods

.Show() which is modeless and ShowDialog() which is modal.

Care should be taken when selecting the method to use.

form.Show()	Everything in NX is still running.
form.ShowDialog()	Will not return to NX unless application triggers to do so since this is modal dialog behavior.

Unload Options

If the application is using Show() method to display a winform, then it should NOT use UnloadImmediately to unload the application. For more information on unload options, see Unloading NX Open Applications

User Defined Objects

UDO Overview

User Defined Objects (UDOs) allow you to add your own custom objects inside of NX to maximize your productivity. You use a UDO when you need an object that does not already exist in NX. You define the data stored in the UDO, and you define it's behavior in NX.

Warning:

It is important to ensure that the required libraries containing the UDO's class and defined behaviors (ie callbacks, and properties) are present in the NX Session before opening a part that contains a UDO. By default NX will not issue any warning or notification about such missing libraries. The required libraries can be automatically loaded into NX during system initialization.

UDOs contain customized data including:

- UDO Name - A UDO class name specifier.
- Free Form Data - May consist of integers, doubles, and strings.
- Convertible Data - May consist of lengths, areas, and volumes.
- Links to NX Objects - There are five different types of links.

UDO behavior inside NX is customized by implementing the following callback methods:

- Display- draws the UDO on the screen via primitive shapes like points, lines, arcs, curves, and facets. If you don't implement this callback - the UDO will be invisible.
- Attention Point - defines the attention point of the UDO (ie where we place temporary notes about the object - such as the numbers drawn on objects after using Information→Object). It is recommended that you use the same method for attention point as used for display. See display section below for more details.
- Fit - defines the boundaries of the UDO. The boundaries of each object in the part (including UDOs) are evaluated when you select View→Operation→Fit. It is recommended that you used the same method for fit as used for display. See display section below for more details.

- Selection - defines regions of space on the screen used to select the UDO. Note: the selection method should be the same method used for display, so that the set of points drawn for the UDO are also the same points that the mouse must hover over to select the UDO. If you implement a different method, you could theoretically have invisible points on the screen that allow you to select the UDO - this is not recommended. Also note that you can add and remove the UDO class from the Type list of the class selection dialog, so simply implementing this method does not mean your UDO is selectable.
- Update - allows the UDO to update whenever the UDO's linked object(s) goes through update (note execution of this method is dependent on the link type used).
- Delete - allows the UDO to do cleanup whenever the UDO's linked object(s) have been deleted (note execution of this method is dependent on the link type used).
- Edit - invoked whenever the user attempts to edit the UDO (either by Edit→User Defined Object or by right clicking on the UDO and selecting Edit from the MB3 popup menu).
- Information - invoked whenever the user goes to Information→Object and selects the UDO.

The following properties also effect UDO behavior:

- Is Occurrenceable - does the UDO display in an assembly, or only when the part containing the UDO is also the displayed part?
- Allow Owned Object Selection - specifies whether or not you have permission to select objects owned by this class.
- Warn User Flag - should we warn users when a part is loaded that contains UDO's of this class (if the class is not yet registered in the session)? Note this warning will only occur once per session (even if there are multiple unregistered classes that attempt to use this warning). After that first warning, additional warnings are suppressed if any new parts are loaded with unregistered UDO classes.

Additional points of interest:

- UDO Status - the status value on the UDO indicates whether the UDO is out of date, and why it's out of date.
- UDO Features - UDO's can also be features so that they can be time-stamped and updated in order with respect to other features in the model.

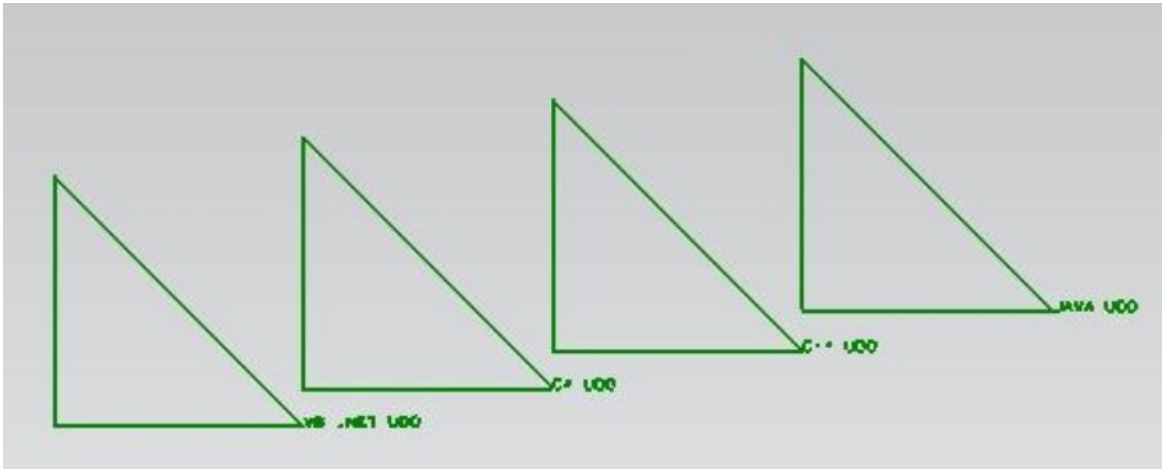
Sample Code

The following 4 programs do essentially the same thing, and snippets of the code will be used throughout this document: Each program creates a very simple UDO example that demonstrates each of the following callbacks:

- Display
- Selection
- Attention Point
- Fit
- Edit
- Information

In NX execute this program via: File → Execute → NX Open... This program begins by opening a new part (if there were no open parts). Next it will prompt you to select a position on the screen. The screen position will be used as a reference point for the UDO. The UDO will display as a

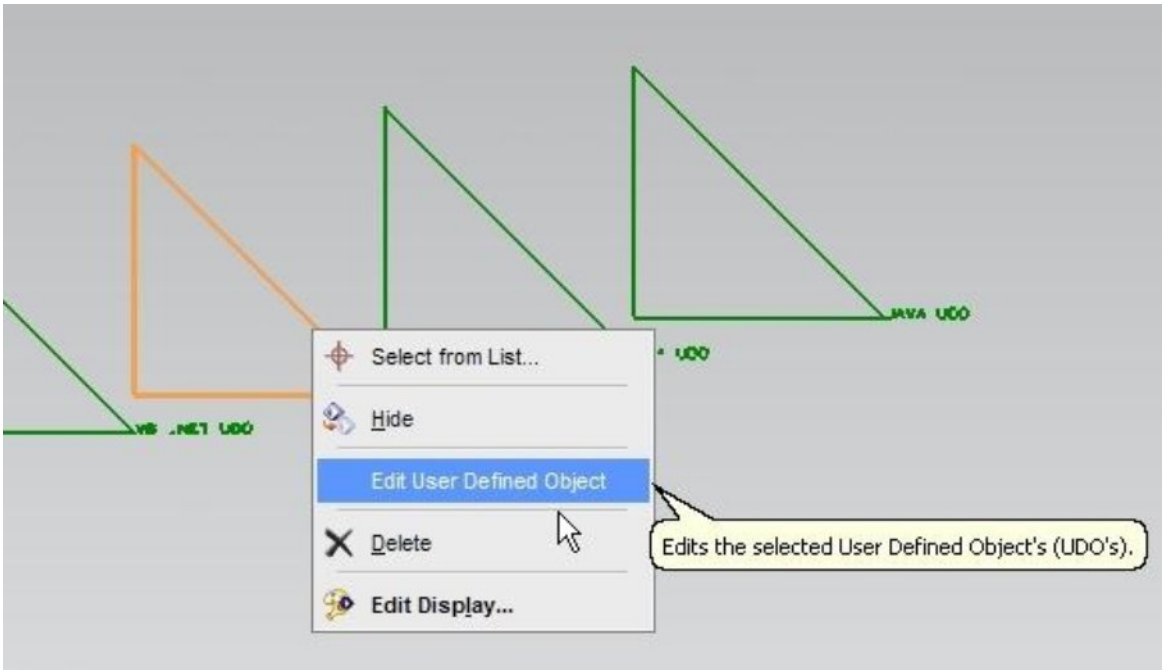
triangle on that point with a name next to the triangle. The name will indicate which language was used to create the UDO.



The image above shows a part with one UDO created by each of the four programs. From left to right you see the VB .NET UDO, the C# UDO, the C++ UDO and finally the JAVA UDO.

This UDO is selectable. If you go to Information→Object and select the UDO you will see custom information output to the listing window from the function myInfoCB defined in this program.

You can also edit the location of the UDO. Start by right-clicking on the UDO and selecting "Edit User Defined Object" from the MB3 popup menu.



Editing the UDO will invoke the myEditCB function defined in this program. You will be prompted to select a new screen position, and after you make the selection the UDO will move to the new location.

Language Specific Section

VB
C#

C++
Java
VB

```
 '------------------------------------------------------------------------
---------
' This program creates a very simple UDO example.
'
' It demonstrates each of the following callbacks:
' * Display
' * Selection
' * Attention Point
' * Fit
' * Edit
' * Information
'
' In NX execute this program via: File -> Execute -> NX Open...
' This program begins by opening a new part (if there were no open
parts).
' Next it will prompt you to select a position on the screen.
' The screen position will be used as a reference point for the UDO.
' The UDO will display as a triangle on that point with a name
' "VB .Net UDO" next to the triangle.
'
' This UDO is selectable. If you go to Information->Object and select
' the UDO you will see custom information output to the listing window
from
' the function myInfoCB defined in this program.
'
' You can also edit the location of the UDO. Start by right-clicking
' on the UDO and selecting "Edit User Defined Object" from the MB3
popup menu.
' Editing the UDO will invoke the myEditCB function defined in this
program.
' You will be prompted to select a new screen position, and after you
make
' the selection the UDO will move to the new location.
'------------------------------------------------------------------------
--------

Option Strict Off
Imports System
Imports NXOpen
Imports NXOpenUI
Imports NXOpen.UF

Module Module1
    Dim theSession As Session = Nothing
    Dim theUI As UI = Nothing
    Dim theUFSession As UFSession = Nothing
```

```
    Dim myUDOclass As UserDefinedObjects.UserDefinedClass = Nothing
    '------------------------------------------------------------------
------------
    ' Callback Name: myDisplayCB
    ' This is a callback method associated with displaying a UDO.
    ' This same callback is registered for display, select, fit, and
attention point
    '------------------------------------------------------------------
------------

    Public Function myDisplayCB
        (ByVal displayEvent As_
         UserDefinedObjects.UserDefinedDisplayEvent)_
         As Integer

        Try
        ' Get the doubles used to define the selected screen position
for this UDO.
        Dim myUDOdoubles() As Double =_
             displayEvent.UserDefinedObject.GetDoubles()

        ' Use the doubles to define points of a triangle
        Dim myPoints(3) As Point3d
        myPoints(0).X = myUDOdoubles(0) + 0
        myPoints(0).Y = myUDOdoubles(1) + 0
        myPoints(0).Z = myUDOdoubles(2) + 0

        myPoints(1).X = myUDOdoubles(0) + 100
        myPoints(1).Y = myUDOdoubles(1) + 0
        myPoints(1).Z = myUDOdoubles(2) + 0

        myPoints(2).X = myUDOdoubles(0) + 0
        myPoints(2).Y = myUDOdoubles(1) + 100
        myPoints(2).Z = myUDOdoubles(2) + 0

        myPoints(3).X = myUDOdoubles(0) + 0
        myPoints(3).Y = myUDOdoubles(1) + 0
        myPoints(3).Z = myUDOdoubles(2) + 0

        ' Display the triangle
        displayEvent.DisplayContext.DisplayPolyline(myPoints)

        ' Display the text next to the triangle
        Dim myPt As Point3d
        myPt.X = myUDOdoubles(0) + 100
        myPt.Y = myUDOdoubles(1) + 0
        myPt.Z = myUDOdoubles(2) + 0
        displayEvent.DisplayContext.DisplayText("VB .Net UDO", myPt, 0)
        Catch ex As NXException
```

```
        ' the display/selection/fit/attention callback is called so
many times
        ' that it's best to print this error handling stuff in the
syslog
        ' any interactive messages in the UI would drive the user crazy
;)
        theUfSession.UF.PrintSyslog("Caught Exception in myDisplayCB:_
              '" & ex.Message() & "'" & vbCrLf, False)
        End Try

        myDisplayCB = 0
    End Function

    '------------------------------------------------------------------
------------

    ' Callback Name: myEditCB
    ' This is a callback method associated with editing a UDO.
    '------------------------------------------------------------------
------------

    Public Function myEditCB
       (ByVal editEvent As_
            UserDefinedObjects.UserDefinedEvent) As Integer
      Try
        Dim myView As NXOpen.View = Nothing
        Dim myCursor As Point3d
        myCursor.X = 0
        myCursor.Y = 0
        myCursor.Z = 0

        ' highlight the current udo we are about to edit
        ' this is helpful if multiple udo's were on the selection
        ' list when the user decided to edit them
        editEvent.UserDefinedObject.Highlight()

        ' ask the user to select a new origin for this UDO
        Dim myResponse As Selection.DialogResponse =_
           theUI.SelectionManager.SelectScreenPosition("Select New
Origin_
                                       for VB UDO", myView, myCursor)
        ' we are done asking the user for input... unhighlight the udo
         editEvent.UserDefinedObject.Unhighlight()
        ' use the new screen position (if the user picked one)
        If myResponse = Selection.DialogResponse.Pick Then
          Dim myUDOdoubles(3) As Double
          myUDOdoubles(0) = myCursor.X
          myUDOdoubles(1) = myCursor.Y
          myUDOdoubles(2) = myCursor.Z
```

```
          ' store the newly selected origin with the udo
          editEvent.UserDefinedObject.SetDoubles(myUDOdoubles)

          ' add the udo to the display list manually
          ' this will force the udo display to regenerate
          ' immediately and show the changes we just made
          theUFSession.Disp.AddItemToDisplay_
             (editEvent.UserDefinedObject.Tag())
        End If
      Catch ex As NXException
        Dim theLW As ListingWindow = theSession.ListingWindow
theLW.Open()
        theLW.WriteLine("Caught Exception in myEditCB: '" &
ex.Message() & "'")
      End Try
        myEditCB = 0
    End Function
    '-------------------------------------------------------------------
------------

    ' Callback Name: myInfoCB '
    This is a callback method associated with querying information for
a UDO.
    ' The information is printed in the listing window.
    '-------------------------------------------------------------------
------------

    Public Function myInfoCB(ByVal infoEvent As
UserDefinedObjects.UserDefinedEvent) As Integer
        Dim theLW As ListingWindow = theSession.ListingWindow
        theLW.Open()
      Try
        theLW.WriteLine(" ")
        theLW.WriteLine("-----------------------------------------------
--------------")
        theLW.WriteLine("Begin Custom Information")
        theLW.WriteLine(" ")
        theLW.WriteLine("UDO Class Name: '" &
infoEvent.UserDefinedObject.UserDefinedClass.ClassName & "'")
        theLW.WriteLine("UDO Friendly Name: '" &
infoEvent.UserDefinedObject.UserDefinedClass.FriendlyName & "'")
        Dim myUDOdoubles() As Double =
infoEvent.UserDefinedObject.GetDoubles
()
        theLW.WriteLine("myUDOdoubles(0) = " &
myUDOdoubles(0).ToString)
        theLW.WriteLine("myUDOdoubles(1) = " &
myUDOdoubles(1).ToString)
```

```
        theLW.WriteLine("myUDOdoubles(2) = " &
myUDOdoubles(2).ToString)
        theLW.WriteLine(" ") theLW.WriteLine_
             ("End Custom Information")
      Catch ex As NXException
        theLW.WriteLine("Caught Exception in myInfoCB:'" & ex.Message()
& "'")
      End Try
        myInfoCB = 0
    End Function

    '------------------------------------------------------------------
------------

    ' initUDO
    ' Checks to see which (if any) of the application's static
variables are
    ' uninitialized, and sets them accordingly.
    ' Initializes the UDO class and registers all of its callback
methods.
    '------------------------------------------------------------------
------------

    Public Function initUDO(ByVal alertUser As Boolean) As Integer
      Try
        If theSession Is Nothing Then
            theSession = Session.GetSession()
        End if
        If theUI Is Nothing Then
            theUI = UI.GetUI()
        End if
        If theUFSession Is Nothing Then
            theUFSession = NXOpen.UF.UFSession.GetUFSession()
        End if
        If myUDOclass Is Nothing Then
            If alertUser = True Then
                MsgBox("Registering VB UDO Class", MsgBoxStyle.OkOnly)
            End If
        ' Define your custom UDO class
        myUDOclass =_
theSession.UserDefinedClassManager.CreateUserDefinedObjectClass_
("Sample_VB_UDO", "Sample VB UDO")
        ' Setup properties on the custom UDO class
        myUDOclass.AllowQueryClassFromName =_
UserDefinedObjects.UserDefinedClass.AllowQueryClass.On
        ' Register callbacks for the UDO class
        myUDOclass.AddDisplayHandler(AddressOf myDisplayCB)
        myUDOclass.AddAttentionPointHandler(AddressOf myDisplayCB)
        myUDOclass.AddFitHandler(AddressOf myDisplayCB)
```

```
        myUDOclass.AddSelectionHandler(AddressOf myDisplayCB)
        myUDOclass.AddEditHandler(AddressOf myEditCB)
        myUDOclass.AddInformationHandler(AddressOf myInfoCB)
        ' Add this class to the list of object types available for
selection in NX.
        ' If you skip this step you won't be able to select UDO's of
this class,
        ' even though you registered a selection callback.
theUI.SelectionManager.SetSelectionStatusOfUserDefinedClass(myUDOclass,
True)
    End If
     Catch ex As NXException
        ' We may be initializing the UDO class during NX Startup
        ' Print any error messages directly to the syslog
        If theUFSession Is Nothing Then
            theUFSession = NXOpen.UF.UFSession.GetUFSession()
        End if
        theUfSession.UF.PrintSyslog("Caught Exception in initUDO: '" &
ex.Message() & "'" & vbCrLf, False)
        End Try
        initUDO = 0
    End Function
    '------------------------------------------------------------------
------------

    ' Main (Explicit Activation)
    ' This entry point is used to activate the application explicitly,
as in
    ' "File->Execute UG/Open->NX Open..."
    '------------------------------------------------------------------
------------

    Sub Main()
        Try
            ' initialize the UDO - if we didn't load this library at
            ' startup, here is our second chance to load it
            initUDO(True)

            ' if we don't have any parts open create one
            Dim myBasePart As BasePart = theSession.Parts.BaseDisplay
            If myBasePart Is Nothing Then
                myBasePart =
theSession.Parts.NewBaseDisplay("test_vb_udo.prt",
BasePart.Units.Millimeters)
            End If

            Dim myView As NXOpen.View = Nothing
            Dim myCursor As Point3d
            myCursor.X = 0
```

```
            myCursor.Y = 0
            myCursor.Z = 0
            ' ask the user to select an origin for this UDO
            Dim myResponse As Selection.DialogResponse =
theUI.SelectionManager.SelectScreenPosition("Select Origin of VB UDO",
myView, myCursor)
            If myResponse = Selection.DialogResponse.Pick Then
                ' The user selected a point - go ahead and create the
udo
                Dim myUDOmanager As
UserDefinedObjects.UserDefinedObjectManager =
myBasePart.UserDefinedObjectManager
                Dim firstUDO As UserDefinedObjects.UserDefinedObject =
myUDOmanager.CreateUserDefinedObject(myUDOclass)
                ' set the color property of the udo - just for fun :)
                firstUDO.Color = 36
                ' store the origin selected by the user with the udo
                Dim myUDOdoubles(3) As Double
                myUDOdoubles(0) = myCursor.X
                myUDOdoubles(1) = myCursor.Y
                myUDOdoubles(2) = myCursor.Z
                firstUDO.SetDoubles(myUDOdoubles)
                ' add the udo to the display list manually
                ' this will force the udo to display immediately
                theUFSession.Disp.AddItemToDisplay(firstUDO.Tag())
            End If
    Catch ex As NXException
            Dim theLW As ListingWindow = theSession.ListingWindow
            theLW.Open()
            theLW.WriteLine("Caught Exception in Main: '" &
ex.Message() & "'")
    End Try
End Sub

'----------------------------------------------------------------------
--------
' Startup
' Entrypoint used when program is loaded automatically
' as NX starts up. Note this application must be placed in a
' special folder for NX to find and load it during startup.
' Refer to the NX Open documentation for more details on how
' NX finds and loads applications during startup.
'----------------------------------------------------------------------
--------

Public Function Startup() As Integer
    initUDO(False)
    Startup = 0
End Function ' Startup ends
```

```
'-------------------------------------------------------------------------
--------

' GetUnloadOption
' Make sure you specify AtTermination for the unload option.
' If you unload the library before the NX Session Terminates
' bad things could happen when we try to execute a udo
' callback that no longer exists in the session.
'-------------------------------------------------------------------------
--------

Public Function GetUnloadOption(ByVal dummy As String) As Integer
    Return CType(Session.LibraryUnloadOption.AtTermination, Integer)
End Function
End Module
```

C#

```
 //-------------------------------------------------------------------------
----------
// This program creates a very simple UDO example.
//
// It demonstrates each of the following callbacks:
//     * Display
//     * Selection
//     * Attention Point
//     * Fit
//     * Edit
//     * Information
//
// In NX execute this program via: File -> Execute -> NX Open...
// This program begins by opening a new part (if there were no open
parts).
// Next it will prompt you to select a position on the screen.
// The screen position will be used as a reference point for the UDO.
// The UDO will display as a triangle on that point with a name
// "C# UDO" next to the triangle.
//
// This UDO is selectable. If you go to Information->Object and select
// the UDO you will see custom information output to the listing window
from
// the function myInfoCB defined in this program.
//
// You can also edit the location of the UDO. Start by right-clicking
// on the UDO and selecting "Edit User Defined Object" from the MB3
popup menu.
// Editing the UDO will invoke the myEditCB function defined in this
program.
// You will be prompted to select a new screen position, and after you
make
```

```
// the selection the UDO will move to the new location.
//------------------------------------------------------------------------
---------
using System;
using NXOpen;
using NXOpen.UF;
using NXOpen.UserDefinedObjects;

public class Program
{
    // class members
    static Session theSession = null;
    static UI theUI = null;
    static UFSession theUFSession = null;
    static UserDefinedClass myUDOclass = null;

    //------------------------------------------------------------------------
-------------

    // Callback Name: myDisplayCB
    // This is a callback method associated with displaying a UDO.
    // This same callback is registered for display, select, fit, and
attention point
    //------------------------------------------------------------------------
-------------

    public static int myDisplayCB(UserDefinedDisplayEvent displayEvent)
    {
        try
        {
            // Get the doubles used to define the selected screen
position for this UDO.
            double[] myUDOdoubles =
displayEvent.UserDefinedObject.GetDoubles();

            // Use the doubles to define points of a triangle
            Point3d[] myPoints = new Point3d[4];
            myPoints[0].X = myUDOdoubles[0] + 0;
            myPoints[0].Y = myUDOdoubles[1] + 0;
            myPoints[0].Z = myUDOdoubles[2] + 0;

            myPoints[1].X = myUDOdoubles[0] + 100;
            myPoints[1].Y = myUDOdoubles[1] + 0;
            myPoints[1].Z = myUDOdoubles[2] + 0;

            myPoints[2].X = myUDOdoubles[0] + 0;
            myPoints[2].Y = myUDOdoubles[1] + 100;
            myPoints[2].Z = myUDOdoubles[2] + 0;
```

```
            myPoints[3].X = myUDOdoubles[0] + 0;
            myPoints[3].Y = myUDOdoubles[1] + 0;
            myPoints[3].Z = myUDOdoubles[2] + 0;

            // Display the triangle
            displayEvent.DisplayContext.DisplayPolyline(myPoints);

            // Display the text next to the triangle
            Point3d myPt = new Point3d();
            myPt.X = myUDOdoubles[0] + 100;
            myPt.Y = myUDOdoubles[1] + 0;
            myPt.Z = myUDOdoubles[2] + 0;
            displayEvent.DisplayContext.DisplayText("C# UDO", myPt, 0);
        }
        catch (NXOpen.NXException ex)
        {
            // ---- Enter your exception handling code here -----
            UI.GetUI().NXMessageBox.Show("Caught exception",
                     NXMessageBox.DialogType.Error, ex.Message);
        }
        return 0;
    }
    //------------------------------------------------------------------
-------------

    // Callback Name: myEditCB
    // This is a callback method associated with editing a UDO.
    //------------------------------------------------------------------
-------------

    public static int myEditCB(UserDefinedEvent editEvent)
    {
        try
        {
            View myView = null;
            Point3d myCursor;
            myCursor.X = 0;
            myCursor.Y = 0;
            myCursor.Z = 0;

            // highlight the current udo we are about to edit
            // this is helpful if multiple udo's were on the selection
            // list when the user decided to edit them
            editEvent.UserDefinedObject.Highlight();

            // ask the user to select a new origin for this UDO
            Selection.DialogResponse myResponse =
theUI.SelectionManager.SelectScreenPosition("Select New Origin for C#
UDO", out myView, out myCursor);
```

```
            // we are done asking the user for input... unhighlight the
udo
            editEvent.UserDefinedObject.Unhighlight();

            // use the new screen position (if the user picked one)
            if( myResponse == Selection.DialogResponse.Pick )
            {
                double[] myUDOdoubles = new double[3];
                myUDOdoubles[0] = myCursor.X;
                myUDOdoubles[1] = myCursor.Y;
                myUDOdoubles[2] = myCursor.Z;
                // store the newly selected origin with the udo
                editEvent.UserDefinedObject.SetDoubles(myUDOdoubles);

                // add the udo to the display list manually
                // this will force the udo display to regenerate
                // immediately and show the changes we just made
                theUFSession.Disp.AddItemToDisplay
(editEvent.UserDefinedObject.Tag);
            }
        }
        catch (NXOpen.NXException ex)
        {
            // ---- Enter your exception handling code here -----
            UI.GetUI().NXMessageBox.Show("Caught exception",
                   NXMessageBox.DialogType.Error, ex.Message);
        }
        return 0;
    } //---------------------------------------------------------------
--------------- /

    / Callback Name: myInfoCB
    // This is a callback method associated with querying information
for a UDO.
    // The information is printed in the listing window.
    //---------------------------------------------------------------
-------------

    public static int myInfoCB(UserDefinedEvent infoEvent)
    {
        try
    {
        ListingWindow
        theLW = theSession.ListingWindow;
        theLW.Open(); theLW.WriteLine(" ");
        theLW.WriteLine("-------------------------------------------
--------------");
        theLW.WriteLine("Begin Custom Information");
        theLW.WriteLine(" ");
```

```
        theLW.WriteLine("UDO Class Name: '" +

infoEvent.UserDefinedObject.UserDefinedClass.ClassName + "'");
        theLW.WriteLine("UDO Friendly Name: '" +

infoEvent.UserDefinedObject.UserDefinedClass.FriendlyName + "'");
        double[] myUDOdoubles =
infoEvent.UserDefinedObject.GetDoubles();
        theLW.WriteLine("myUDOdoubles(0) = " + myUDOdoubles[0]);
        theLW.WriteLine("myUDOdoubles(1) = " + myUDOdoubles[1]);
        theLW.WriteLine("myUDOdoubles(2) = " + myUDOdoubles[2]);
        theLW.WriteLine(" "); theLW.WriteLine("End Custom
Information");
    }
    catch (NXOpen.NXException ex)
    {
        // ---- Enter your exception handling code here -----
        UI.GetUI().NXMessageBox.Show("Caught exception",
NXMessageBox.DialogType.Error, ex.Message);
    }
    return 0;
}
//------------------------------------------------------------------------
---------
// initUDO // Checks to see which (if any) of the application's static
variables are
// uninitialized, and sets them accordingly.
// Initializes the UDO class and registers all of its callback methods.
//------------------------------------------------------------------------
---------
static int initUDO( bool alertUser)
{
    try
    {
        if (theSession == null)
        {
            theSession = Session.GetSession();
        }
        if( theUI == null )
        {
            theUI = UI.GetUI();
        }
        if( theUFSession == null )
        {
        theUFSession = UFSession.GetUFSession();
        }
        if (myUDOclass == null)
        {
            if (alertUser)
```

```
            {
                ListingWindow theLW = theSession.ListingWindow;
                theLW.Open();
                theLW.WriteLine("Registering C# UDO Class");
            }
            // Define your custom UDO class
            myUDOclass =
theSession.UserDefinedClassManager.CreateUserDefinedObjectClass
                        ("Sample_CSharp_UDO", "Sample C# UDO");
            // Setup properties on the custom UDO class
            myUDOclass.AllowQueryClassFromName =
UserDefinedClass.AllowQueryClass.On;
            // Register callbacks for the UDO class
            myUDOclass.AddDisplayHandler(new
UserDefinedClass.DisplayCallback(Program.myDisplayCB));
            myUDOclass.AddAttentionPointHandler(new
UserDefinedClass.DisplayCallback   (Program.myDisplayCB));
            myUDOclass.AddFitHandler(new
UserDefinedClass.DisplayCallback
(Program.myDisplayCB));
            myUDOclass.AddSelectionHandler(new
UserDefinedClass.DisplayCallback(Program.myDisplayCB));
            myUDOclass.AddEditHandler(new
UserDefinedClass.GenericCallback(Program.myEditCB));
            myUDOclass.AddInformationHandler(new
UserDefinedClass.GenericCallback(Program.myInfoCB));
            // Add this class to the list of object types available for
selection in NX.
            // If you skip this step you won't be able to select UDO's
of this class,
            // even though you registered a selection callback.
theUI.SelectionManager.SetSelectionStatusOfUserDefinedClass(myUDOclass,
true);
        }
    }
    catch (NXOpen.NXException ex)
    {
        // ---- Enter your exception handling code here -----
        UI.GetUI().NXMessageBox.Show("Caught exception",
NXMessageBox.DialogType.Error, ex.Message);
    }
    return 0;
}
//------------------------------------------------------------------------
---------

// Main (Explicit Activation)
// This entry point is used to activate the application explicitly, as
in
```

```
// "File->Execute UG/Open->NX Open..."
//------------------------------------------------------------------------
---------
public static int Main(string[] args)
{
    int retValue = 0;
    try
    {
        // initialize the UDO - if we didn't load this library at
        // startup, here is our second chance to load it
        initUDO(true);

        // if we don't have any parts open create one
        BasePart myBasePart = theSession.Parts.BaseDisplay;
        if( myBasePart == null)
        { myBasePart = theSession.Parts.NewBaseDisplay
("test_csharp_udo.prt", BasePart.Units.Millimeters);
        }

        View myView = null;
        Point3d myCursor;
        myCursor.X = 0;
        myCursor.Y = 0;
        myCursor.Z = 0;
        // ask the user to select an origin for this UDO
        Selection.DialogResponse myResponse =
theUI.SelectionManager.SelectScreenPosition("Select Origin of C# UDO",
out myView, out myCursor);
        if
        ( myResponse == Selection.DialogResponse.Pick )
        {
        // The user selected a point - go ahead and create the udo
    UserDefinedObjectManager myUDOmanager =
myBasePart.UserDefinedObjectManager;
        UserDefinedObject firstUDO =
myUDOmanager.CreateUserDefinedObject(myUDOclass);
        // set the color property of the udo - just for fun :)
        firstUDO.Color = 36;
        // store the origin selected by the user with the udo
        double[] myUDOdoubles = new double[3];
        myUDOdoubles[0] = myCursor.X;
        myUDOdoubles[1] = myCursor.Y;
        myUDOdoubles[2] = myCursor.Z;
        firstUDO.SetDoubles(myUDOdoubles);
        // add the udo to the display list manually
        // this will force the udo to display immediately
        theUFSession.Disp.AddItemToDisplay(firstUDO.Tag);
    }
        }
```

```
        catch (NXOpen.NXException ex)
        {/
            // ---- Enter your exception handling code here -----
            UI.GetUI().NXMessageBox.Show("Caught exception",
NXMessageBox.DialogType.Error, ex.Message);
        }
        return retValue;
    }
    //------------------------------------------------------------------
-------------

    // Startup
    // Entrypoint used when program is loaded automatically
    // as NX starts up. Note this application must be placed in a
    // special folder for NX to find and load it during startup.
    // Refer to the NX Open documentation for more details on how
    // NX finds and loads applications during startup.
    //------------------------------------------------------------------
-------------

    public static int Startup()
    {
        int retValue = 0;
        try
    {
        initUDO(false);
    }
    catch (NXOpen.NXException ex)
    {
        // ---- Enter your exception handling code here -----
        UI.GetUI().NXMessageBox.Show("Caught exception",
NXMessageBox.DialogType.Error, ex.Message);
    }
    return retValue;
    }

    //------------------------------------------------------------------
-------------

    // GetUnloadOption
    // Make sure you specify AtTermination for the unload option.
    // If you unload the library before the NX Session Terminates
    // bad things could happen when we try to execute a udo
    // callback that no longer exists in the session.
    //------------------------------------------------------------------
-------------

    public static int GetUnloadOption(string arg)
    {
```

```
        //Unloads the image when the NX session terminates
        return
System.Convert.ToInt32(Session.LibraryUnloadOption.AtTermination);
    }
}
```

C++

```
 //------------------------------------------------------------------------
----------
// This program creates a very simple UDO example.
//
// It demonstrates each of the following callbacks:
// * Display
// * Selection
// * Attention Point
// * Fit
// * Edit
// * Information
//
// In NX execute this program via: File -> Execute -> NX Open...
// This program begins by opening a new part (if there were no open
parts).
// Next it will prompt you to select a position on the screen.
// The screen position will be used as a reference point for the UDO.
// The UDO will display as a triangle on that point with a name
// "C++ UDO" next to the triangle.
//
// This UDO is selectable. If you go to Information->Object and select
// the UDO you will see custom information output to the listing window
from
// the function myInfoCB defined in this program.
//
// You can also edit the location of the UDO. Start by right-clicking
// on the UDO and selecting "Edit User Defined Object" from the MB3
popup menu.
// Editing the UDO will invoke the myEditCB function defined in this
program.
// You will be prompted to select a new screen position, and after you
make
// the selection the UDO will move to the new location.
//------------------------------------------------------------------------
---------
/* Include files */
#if ! defined ( __hp9000s800 ) && ! defined ( __sgi ) && ! defined (
__sun )
# include <strstream>
# include <iostream>
using std::ostrstream;
using std::endl;
using std::ends;
```

```
using std::cerr;
#else
# include <strstream.h>
# include iostream.h>
#endif
#include <uf.h>
#include <uf_ui.h>
#include <uf_exit.h>
#include <ufdisp.h>

#include <NXOpen/Session.hxx>
#include <NXOpen/Part.hxx>
#include <NXOpen/PartCollection.hxx>
#include <NXOpen/Callback.hxx>
#include <NXOpen/NXException.hxx>
#include <NXOpen/UI.hxx>
#include <NXOpen/Selection.hxx>
#include <NXOpen/LogFile.hxx>
#include <NXOpen/NXObjectManager.hxx>
#include <NXOpen/ListingWindow.hxx>
#include <NXOpen/View.hxx>

#include <NXOpen/UserDefinedObjects_UserDefinedClass.hxx>
#include <NXOpen/UserDefinedObjects_UserDefinedClassManager.hxx>
#include <NXOpen/UserDefinedObjects_UserDefinedObject.hxx>
#include <NXOpen/UserDefinedObjects_UserDefinedObjectManager.hxx>
#include <NXOpen/UserDefinedObjects_UserDefinedEvent.hxx>
#include <NXOpen/UserDefinedObjects_UserDefinedDisplayEvent.hxx>
#include <NXOpen/UserDefinedObjects_UserDefinedLinkEvent.hxx>
#include
<NXOpen/UserDefinedObjects_UserDefinedObjectDisplayContext.hxx>

using namespace NXOpen;
using namespace NXOpen::UserDefinedObjects;
//static variables
static NXOpen::Session* theSession = NULL;
static UI* theUI = NULL;
static UserDefinedClass* myUDOclass = NULL;

//------------------------------------------------------------------------
---------
// Callback Name: myDisplayCB
// This is a callback method associated with displaying a UDO.
// This same callback is registered for display, select, fit, and
attention point
//------------------------------------------------------------------------
---------
extern int myDisplayCB(UserDefinedDisplayEvent*  displayEvent)
{
```

```
    try
    {
        // Get the doubles used to define the selected screen position
for this UDO.
        std::vector myUDOdoubles = displayEvent->UserDefinedObject()-
>GetDoubles();

        // Use the doubles to define points of a triangle
        std::vector

        myPoints(4); myPoints[0].X = myUDOdoubles[0] + 0;
        myPoints[0].Y = myUDOdoubles[1] + 0;
        myPoints[0].Z = myUDOdoubles[2] + 0;

        myPoints[1].X = myUDOdoubles[0] + 100;
        myPoints[1].Y = myUDOdoubles[1] + 0;
        myPoints[1].Z = myUDOdoubles[2] + 0;

        myPoints[2].X = myUDOdoubles[0] + 0;
        myPoints[2].Y = myUDOdoubles[1] + 100;
        myPoints[2].Z = myUDOdoubles[2] + 0;

        myPoints[3].X = myUDOdoubles[0] + 0;
        myPoints[3].Y = myUDOdoubles[1] + 0;
        myPoints[3].Z = myUDOdoubles[2] + 0;

        // Display the triangle
        displayEvent->DisplayContext()->DisplayPolyline(myPoints);

        // Display the text next to the triangle
        Point3d myPt = Point3d(myUDOdoubles[0] + 100, myUDOdoubles[1],
myUDOdoubles[2]);
        displayEvent->DisplayContext()->DisplayText("C++ UDO", myPt,
UserDefinedObjectDisplayContext::TextRefBottomLeft);
    }
    catch (NXException ex)
    {
        // ---- Enter your exception handling code here -----
        cerr << "Caught exception: " << ex.Message() << endl;
    }
    return 0;
}
//----------------------------------------------------------------------
---------
// Callback Name: myEditCB
// This is a callback method associated with editing a UDO.
//----------------------------------------------------------------------
---------
extern int myEditCB(UserDefinedEvent* editEvent)
```

```
{
    try
    {
        // required for calls to legacy UF routines
        // such as UF_DISP_add_item_to_display
        UF_initialize();

        View* myView = NULL;
        Point3d myCursor(0,0,0);

        // highlight the current udo we are about to edit
        // this is helpful if multiple udo's were on the selection
        // list when the user decided to edit them
        editEvent->UserDefinedObject()->Highlight();

        // ask the user to select a new origin for this UDO
        Selection::DialogResponse myResponse = theUI-
>SelectionManager()->SelectScreenPosition("Select New Origin for C++
UDO", &myView, &myCursor);
        // we are done asking the user for input... unhighlight the udo
        editEvent->UserDefinedObject()->Unhighlight();

        // use the new screen position (if the user picked one)
        if( myResponse == Selection::DialogResponsePick )
        {
        std::vector myUDOdoubles(3);
        myUDOdoubles[0] = myCursor.X;
        myUDOdoubles[1] = myCursor.Y;
        myUDOdoubles[2] = myCursor.Z;
        // store the newly selected origin with the udo
        editEvent->UserDefinedObject()->SetDoubles(myUDOdoubles);
        // add the udo to the display list manually
        // this will force the udo display to regenerate
        // immediately and show the changes we just made
        UF_DISP_add_item_to_display(editEvent->UserDefinedObject()-
>GetTag());
        }
        UF_terminate();
    } catch (NXException ex)
    {
        // ---- Enter your exception handling code here -----
        cerr << "Caught exception: " << ex.Message() << endl;
    } return 0;
}
//----------------------------------------------------------------------
---------
// Callback Name: myInfoCB
// This is a callback method associated with querying information for a
UDO.
```

```
// The information is printed in the listing window.
//-------------------------------------------------------------------
---------
extern int myInfoCB(UserDefinedEvent* infoEvent)
{
    try
    {
        ListingWindow* theLW = theSession->ListingWindow();
        char msg[256];
        theLW->Open();
        theLW->WriteLine(" ");
        theLW->WriteLine("-----------------------------------------
---------------");
        theLW->WriteLine("Begin Custom Information");
        theLW->WriteLine(" ");
        sprintf( msg, "UDO Class Name: '%s'", infoEvent-
>UserDefinedObject()->UserDefinedClass()->ClassName() );
        theLW->WriteLine(msg);
        sprintf( msg, "UDO Friendly Name: '%s'", infoEvent-
>UserDefinedObject()->UserDefinedClass()->FriendlyName() );
        theLW->WriteLine(msg);
        std::vector myUDOdoubles = infoEvent->UserDefinedObject()-
>GetDoubles();
        sprintf( msg, "myUDOdoubles(0) = %f", myUDOdoubles[0] );
        theLW->WriteLine(msg);
        sprintf( msg, "myUDOdoubles(1) = %f", myUDOdoubles[1] );
        theLW->WriteLine(msg); sprintf( msg, "myUDOdoubles(2) = %f",
myUDOdoubles[2] );
        theLW->WriteLine(msg);
        theLW->WriteLine(" ");
        theLW->WriteLine("End Custom Information");
    }
    catch (NXException ex)
    {
    // ---- Enter your exception handling code here -----
    cerr << "Caught exception: " << ex.Message() << endl;
    }
    return 0;
}

//-------------------------------------------------------------------
---------
// initUDO
// Checks to see which (if any) of the application's static variables
are
// uninitialized, and sets them accordingly.
// Initializes the UDO class and registers all of its callback methods.
//-------------------------------------------------------------------
---------
```

```
static int initUDO( bool alertUser)
{
    try
      {
        if (theSession == NULL)
        {
        theSession = Session::GetSession();
        }
        if( theUI == NULL )
        {
        theUI = UI::GetUI();
        }
        if (myUDOclass == NULL)
        {
        if (alertUser)
        {
        ListingWindow*
        theLW = theSession->ListingWindow();
        theLW->Open();
        theLW->WriteLine("Registering C++ UDO Class");
        }
        // Define your custom UDO class
        myUDOclass = theSession->UserDefinedClassManager()-
>CreateUserDefinedObjectClass("Sample_Cpp_UDO", "Sample C++ UDO");
        // Setup properties on the custom UDO class
        myUDOclass-
>SetAllowQueryClassFromName(UserDefinedClass::AllowQueryClassOn);
        // Register callbacks for the UDO class
        myUDOclass->AddDisplayHandler(make_callback(&myDisplayCB));
        myUDOclass-
>AddAttentionPointHandler(make_callback(&myDisplayCB));
        myUDOclass->AddFitHandler(make_callback(&myDisplayCB));
        myUDOclass->AddSelectionHandler(make_callback(&myDisplayCB));
        myUDOclass->AddEditHandler(make_callback(&myEditCB));
        myUDOclass->AddInformationHandler(make_callback(&myInfoCB));
        // Add this class to the list of object types available for
selection in NX.
        // If you skip this step you won't be able to select UDO's of
this class,
        // even though you registered a selection callback.
        theUI->SelectionManager()-
>SetSelectionStatusOfUserDefinedClass(myUDOclass, true);
      }
    }
    catch (NXException ex)
    {
    // ---- Enter your exception handling code here -----
    cerr << "Caught exception: " << ex.Message() << endl;
    }
```

```
    return 0;
}

//------------------------------------------------------------------------
---------
// ufusr (Explicit Activation)
// This entry point is used to activate the application explicitly, as
in
// "File->Execute UG/Open->NX Open..."
//------------------------------------------------------------------------
---------
extern void ufusr( char *parm, int *returnCode, int rlen )
{
        try
        {
        // required for calls to legacy UF routines
        // such as UF_DISP_add_item_to_display
        UF_initialize();

        // initialize the UDO - if we didn't load this library at
        // startup, here is our second chance to load it
        initUDO(true);

        // if we don't have any parts open create one
        BasePart* myBasePart = theSession->Parts()->BaseDisplay();
        if( myBasePart == NULL)
        {
            myBasePart = theSession->Parts()-
>NewBaseDisplay("test_cpp_udo.prt", BasePart::UnitsMillimeters);
        }

        View* myView = NULL;
        Point3d myCursor(0,0,0);

        // ask the user to select an origin for this UDO
        Selection::DialogResponse myResponse = theUI-
>SelectionManager()->SelectScreenPosition("Select Origin of C++ UDO",
&myView, &myCursor);
        if( myResponse == Selection::DialogResponsePick )
        {
            // The user selected a point - go ahead and create the udo
            UserDefinedObjectManager* myUDOmanager = myBasePart-
>UserDefinedObjectManager();
            UserDefinedObject* firstUDO = myUDOmanager-
>CreateUserDefinedObject(myUDOclass);
            // set the color property of the udo - just for fun :)
            firstUDO->SetColor(36);
            // store the origin selected by the user with the udo
            std::vector myUDOdoubles(3);
```

```
            myUDOdoubles[0] = myCursor.X;
            myUDOdoubles[1] = myCursor.Y;
            myUDOdoubles[2] = myCursor.Z;
            firstUDO->SetDoubles(myUDOdoubles);
            // add the udo to the display list manually
            // this will force the udo to display immediately
            UF_DISP_add_item_to_display(firstUDO->GetTag());
        }
        UF_terminate();
    }
    catch (const NXOpen::NXException& ex)
    {
    cerr << "Caught exception: " << ex.Message() << endl;
    }
}

//------------------------------------------------------------------------
---------
// ufsta
// Entrypoint used when program is loaded automatically
// as NX starts up. Note this application must be placed in a
// special folder for NX to find and load it during startup.
// Refer to the NX Open documentation for more details on how
// NX finds and loads applications during startup.
//------------------------------------------------------------------------
---------
extern void ufsta( char *param, int *returnCode, int rlen )
{
    try
    {
        initUDO(false);
    }
    catch (const NXOpen::NXException& ex)
    {
    cerr << "Caught exception: " << ex.Message() << endl;
    }
} //------------------------------------------------------------------------
-----------
// ufusr_ask_unload
// Make sure you specify AtTermination for the unload option.
// If you unload the library before the NX Session Terminates
// bad things could happen when we try to execute a udo
// callback that no longer exists in the session.
//------------------------------------------------------------------------
---------
extern int ufusr_ask_unload( void )
{
//return (int)Session::LibraryUnloadOptionExplicitly;
// return (int)Session::LibraryUnloadOptionImmediately;
```

```
return (int)Session::LibraryUnloadOptionAtTermination;
}
```

JAVA

```
 //------------------------------------------------------------------------
----------
// This program creates a very simple UDO example.
//
// It demonstrates each of the following callback reasons:
// * Display
// * Selection
// * Attention Point
// * Fit
// * Edit
// * Information
//
// In NX execute this program via: File -> Execute -> NX Open...
// This program begins by opening a new part (if there were no open
parts).
// Next it will prompt you to select a position on the screen.
// The screen position will be used as a reference point for the UDO.
// The UDO will display as a triangle on that point with a name
// "JAVA UDO" next to the triangle.
//
// This UDO is selectable. If you go to Information->Object and select
// the UDO you will see custom information output to the listing window
from
// the function genericCallback defined in this program.
//
// You can also edit the location of the UDO. Start by right-clicking
// on the UDO and selecting "Edit User Defined Object" from the MB3
popup menu.
// Editing the UDO will invoke the genericCallback function defined in
this program.
// You will be prompted to select a new screen position, and after you
make
// the selection the UDO will move to the new location.
//------------------------------------------------------------------------
---------
import nxopen.*;
import nxopen.userdefinedobjects.*;
import java.io.*;

// SimpleJavaUDO class used to demo a UDO in the java language
public class SimpleJavaUDO implements
nxopen.userdefinedobjects.UserDefinedClass.DisplayCallback,
nxopen.userdefinedobjects.UserDefinedClass.GenericCallback
{
// class members
     public static Session theSession = null;
```

```
    public static UI theUI = null;
    public static UFSession theUFSession = null;
    public static UserDefinedClass myUDOclass = null;

    static SimpleJavaUDO theSimpleJavaUDO;

    //------------------------------------------------------------------
-------------

    // Callback Name: displayCallback
    // This is a callback method associated with displaying a UDO.
    // This same callback is registered for display, select, fit, and
attention point
    //------------------------------------------------------------------
-------------

    public int
displayCallback(nxopen.userdefinedobjects.UserDefinedDisplayEvent e)
    {
        int retValue = 0;
        try
        {
            // all display reasons (DISPLAY, SELECTION, FIT, and
ATTENTION_POINT)
            // should do the same thing so we don't need to figure out
the
            // reason we're in this displayCallback

            //Get the doubles used to define the selected screen
position for this UDO.

            double[] myUDOdoubles = e.userDefinedObject().getDoubles();

            // Use the doubles to define points of a triangle
            Point3d[] myPoints = new Point3d[]
                                { new Point3d( myUDOdoubles[0] + 0,
                                               myUDOdoubles[1] + 0,
                                               myUDOdoubles[2] + 0),
                                  new Point3d( myUDOdoubles[0] + 100,
                                               myUDOdoubles[1] + 0,
                                               myUDOdoubles[2] + 0),
                                  new Point3d( myUDOdoubles[0] + 0,
                                               myUDOdoubles[1] + 100,
                                               myUDOdoubles[2] + 0),
                                               new
Point3d(myUDOdoubles[0] + 0,
                                               myUDOdoubles[1] + 0,
                                               myUDOdoubles[2] + 0)};
            // Display the triangle
```

```
            e.displayContext().displayPolyline(myPoints);

            // Display the text next to the triangle
            Point3d myPt = new Point3d();
            myPt.x = myUDOdoubles[0] + 100;
            myPt.y = myUDOdoubles[1] + 0;
            myPt.z = myUDOdoubles[2] + 0;
            e.displayContext().displayText("JAVA UDO", myPt,
UserDefinedObjectDisplayContext.TextRef.BOTTOM_LEFT);
        }
        catch(Exception ex)
        {
            System.out.println("Error Message");
            System.out.println(ex.getMessage());
        }
        return retValue;
    }
//------------------------------------------------------------------------
---------

  // Callback Name: genericCallback
  // This is a callback method associated with editing a UDO, or
querying the UDO
  // for information. Because one callback is used to do two completely
different
  // things (with different functionality for each thing) we must first
check the
  // reason stored in the event object to see which piece of
functionality we need
  // to execute.
  //------------------------------------------------------------------------
-----------
  public int genericCallback(nxopen.userdefinedobjects.UserDefinedEvent
e)
  {
   int retValue = 0;
   try
   {
    if(e.eventReason() ==
nxopen.userdefinedobjects.UserDefinedEvent.Reason.EDIT )
    {
     // highlight the current udo we are about to edit
     // this is helpful if multiple udo's were on the selection
     // list when the user decided to edit them
     e.userDefinedObject().highlight();

     // ask the user to select a new origin for this UDO
```

```
     Selection.SelectScreenPositionData mySelectionData =
theUI.selectionManager().selectScreenPosition("Select Origin of Java
UDO");
     // we are done asking the user for input... unhighlight the udo
     e.userDefinedObject().unhighlight();

     // use the new screen position (if the user picked one)
     if( mySelectionData.response == Selection.DialogResponse.PICK )
     {
      double[]
      myUDOdoubles = new double[3];
      myUDOdoubles[0] = mySelectionData.screenPosition.x;
      myUDOdoubles[1] = mySelectionData.screenPosition.y;
      myUDOdoubles[2] = mySelectionData.screenPosition.z;
      // store the newly selected origin with the udo
      e.userDefinedObject().setDoubles(myUDOdoubles);

      // add the udo to the display list manually
      // this will force the udo display to regenerate
      // immediately and show the changes we just made

theUFSession.disp().addItemToDisplay(e.userDefinedObject().tag());
     }
    }
    else if(e.eventReason() ==
nxopen.userdefinedobjects.UserDefinedEvent.Reason.INFO )
    {
     // print information for the udo here
     ListingWindow
     theLW = theSession.listingWindow();
     theLW.open(); theLW.writeLine(" ");
     theLW.writeLine("-----------------------------------------------
-----------");
     theLW.writeLine("Begin Custom Information");
     theLW.writeLine(" ");
     theLW.writeLine("UDO Class Name: '" +
e.userDefinedObject().userDefinedClass().className() + "'");
     theLW.writeLine("UDO Friendly Name: '" +
e.userDefinedObject().userDefinedClass().friendlyName() + "'");
     double[] myUDOdoubles = e.userDefinedObject().getDoubles();
     theLW.writeLine("myUDOdoubles(0) = " + myUDOdoubles[0]);
     theLW.writeLine("myUDOdoubles(1) = " + myUDOdoubles[1]);
     theLW.writeLine("myUDOdoubles(2) = " + myUDOdoubles[2]);
     theLW.writeLine(" "); theLW.writeLine("End Custom Information");
    }
   }
   catch(Exception ex)
   {
    System.out.println("Error Message");
```

```
    System.out.println(ex.getMessage());
   }
   return retValue;
  }
  //------------------------------------------------------------------------
-----------

  // initUDO
  // Checks to see which (if any) of the application's static variables
are
  // uninitialized, and sets them accordingly.
  // Initializes the UDO class and registers all of its callback
methods.
  //------------------------------------------------------------------------
-----------
  private void initUDO()
  {
   try
   {
    if (theSession == null)
    {
     theSession = (Session)SessionFactory.get("Session");
    }
    if( theUI == null )
    {
     theUI = (UI)SessionFactory.get("UI");
    }
    if( theUFSession == null )
    {
     theUFSession = (UFSession)SessionFactory.get("UFSession");
    }
    if (myUDOclass == null)
    {
     // Define your custom UDO class
     myUDOclass =
theSession.userDefinedClassManager().createUserDefinedObjectClass("Samp
le_Java_UDO", "Sample Java UDO" );
     // Setup properties on the custom UDO class
     myUDOclass.setAllowQueryClassFromName(
nxopen.userdefinedobjects.UserDefinedClass.AllowQueryClass.ON );
     // Register callbacks for the UDO class
     myUDOclass.addDisplayHandler(this);
     myUDOclass.addAttentionPointHandler(this);
     myUDOclass.addFitHandler(this);
     myUDOclass.addSelectionHandler(this);
     myUDOclass.addEditHandler(this);
     myUDOclass.addInformationHandler(this);
     // Add this class to the list of object types available for
selection in NX.
```

```
     // If you skip this step you won't be able to select UDO's of this
class,
     // even though you registered a selection callback.

theUI.selectionManager().setSelectionStatusOfUserDefinedClass(myUDOclas
s, true);
   }
            }
            catch(Exception ex)
            {
                    System.out.println("Error Message");
                    System.out.println(ex.getMessage());
            }
    }
    // constructor
    public SimpleJavaUDO()
    {
            try
            {
                    initUDO();
            }
            catch(Exception ex)
            {
                    System.out.println("Caught Exception");
                    System.out.println(ex.getMessage());
            }
    }

    //------------------------------------------------------------------
-------------

    // main (Explicit Activation)
    // This entry point is used to activate the application explicitly,
as in
    // "File->Execute UG/Open->NX Open..."
    //------------------------------------------------------------------
-------------

    public static void main(String[] args)
    {
            try
            {
                    theSimpleJavaUDO = new SimpleJavaUDO();
                    // if we don't have any parts open create one
                    BasePart myBasePart =
theSession.parts().baseDisplay();
                    if( myBasePart == null)
                    {
```

```
                        myBasePart =
theSession.parts().newBaseDisplay("test_java_udo.prt",
BasePart.Units.MILLIMETERS);
                }

                // ask the user to select an origin for this UDO
                Selection.SelectScreenPositionData mySelectionData =
theUI.selectionManager().selectScreenPosition("Select Origin of Java
UDO");
                if( mySelectionData.response ==
Selection.DialogResponse.PICK )
                {
                        // The user selected a point - go ahead and
create the udo
                        UserDefinedObjectManager myUDOmanager =
myBasePart.userDefinedObjectManager;
                        UserDefinedObject firstUDO =
myUDOmanager.createUserDefinedObject(myUDOclass);
                        // set the color property of the udo - just
for fun :)
                        firstUDO.setColor(36);
                        // store the origin selected by the user
with the udo
                        double[] myUDOdoubles = new double[3];
                        myUDOdoubles[0] =
mySelectionData.screenPosition.x;
                        myUDOdoubles[1] =
mySelectionData.screenPosition.y;
                        myUDOdoubles[2] =
mySelectionData.screenPosition.z;
                        firstUDO.setDoubles(myUDOdoubles);
                        // add the udo to the display list manually
                        // this will force the udo to display
immediately

    theUFSession.disp().addItemToDisplay(firstUDO.tag());
                }
            }
            catch(Exception ex)
            {
                System.out.println("Caught Exception");
                System.out.println(ex.getMessage());
            }
    }
    //------------------------------------------------------------------
-------------

    // startup
    // Entrypoint used when program is loaded automatically
```

```
    // as NX starts up. Note this application must be placed in a
    // special folder for NX to find and load it during startup.
    // Refer to the NX Open documentation for more details on how
    // NX finds and loads applications during startup.
    //------------------------------------------------------------
-------------
    public static void startup (String [] args)throws NXException,
java.rmi.RemoteException
    {
            try
            {
                    theSimpleJavaUDO = new SimpleJavaUDO();
    }
    catch(Exception ex)
    {
            System.out.println("Caught Exception");
            System.out.println(ex.getMessage());
    }
    }
    //------------------------------------------------------------
-------------

    // getUnloadOption
    // Make sure you specify AtTermination for the unload option.
    // If you unload the library before the NX Session Terminates
    // bad things could happen when we try to execute a udo
    // callback that no longer exists in the session.
    //------------------------------------------------------------
-------------

    public static int getUnloadOption()
    {
            return BaseSession.LibraryUnloadOption.AT_TERMINATION;
    }
```

UDO Name

Before creating a user defined object in your part, you must first define the class for the UDO. The class must contain a class name and a "user friendly" name. The class name should not match the name of any NX object. A class name should be specified as unique to a particular application to avoid conflicts with class names from other applications. The user friendly name doesn't have to be unique, but using a descriptive name is advised to avoid confusing the end users.

The method to create a UDO class and establish the class name is CreateUserDefinedObjectClass. This method can be found in the UserDefinedClassManager.

Examples:

	Code (create class)

VB	`myUDOclass = theSession.UserDefinedClassManager.CreateUserDefinedObjectClass("Sample_VB_UDO", "Sample VB UDO")`
C#	`myUDOclass = theSession.UserDefinedClassManager.CreateUserDefinedObjectClass("Sample_CSharp_UDO", "Sample C# UDO");`
C++	`myUDOclass = theSession->UserDefinedClassManager()->CreateUserDefinedObjectClass("Sample_Cpp_UDO", "Sample C++ UDO");`
Java	`myUDOclass = theSession.userDefinedClassManager().createUserDefinedObjectClass("Sample_Java_UDO", "Sample Java UDO" );`

Free Form Data

There are predefined areas for data types (integers, doubles, and strings) that are controlled exclusively by the user (NX has no knowledge of the content or use of these areas and does not perform any validation on these areas). These areas are arrays that can contain any number of valid data element types.

There are functions for adding, editing, deleting, and querying the area for each data type.

Examples:

	Code (get doubles)
VB	`Dim myUDOdoubles() As Double = displayEvent.UserDefinedObject.GetDoubles()`
C#	`double[] myUDOdoubles = displayEvent.UserDefinedObject.GetDoubles();`
C++	`std::vector<double> myUDOdoubles = displayEvent→UserDefinedObject()→GetDoubles();`
Java	`double[] myUDOdoubles = e.userDefinedObject().getDoubles();`
	Code (set doubles)

VB	`Dim myUDOdoubles(3) As Double`
	`myUDOdoubles(0) = myCursor.X`
	`myUDOdoubles(1) = myCursor.Y`
	`myUDOdoubles(2) = myCursor.Z`
	`' store the newly selected origin with the udo`
	`editEvent.UserDefinedObject.SetDoubles(myUDOdoubles)`

C#	`double[] myUDOdoubles = new double[3]; myUDOdoubles[0] = myCursor.X; myUDOdoubles[1] = myCursor.Y; myUDOdoubles[2] = myCursor.Z; // store the newly selected origin with the udo editEvent.UserDefinedObject.SetDoubles(myUDOdoubles);`

C++	`std::vector<double> myUDOdoubles(3);`
	`myUDOdoubles[0] = myCursor.X;`
	`myUDOdoubles[1] = myCursor.Y;`
	`myUDOdoubles[2] = myCursor.Z;`
	`// store the newly selected origin with the udo`
	`editEvent->UserDefinedObject()->SetDoubles(myUDOdoubles);`

Java

```
double[] myUDOdoubles = new double[3];
myUDOdoubles[0] = mySelectionData.screenPosition.x;
myUDOdoubles[1] = mySelectionData.screenPosition.y;
myUDOdoubles[2] = mySelectionData.screenPosition.z;
// store the newly selected origin with the udo
e.userDefinedObject().setDoubles(myUDOdoubles);
```

Convertible Data

There are data areas (unlike the free form areas) that are converted automatically from English to Metric or vice versa when the part file is converted. These convertible data areas are arrays of elements which represent:

- Lengths — an array that can contain any number of doubles where each double represents a unit of length (e.g. centimeters).
- Areas — an array that can contain any number of doubles where each double represents a unit of area (e.g. square meters).
- Volumes — an array that can contain any number of doubles where each double represents a unit of volume (e.g. cubic meters).

There are functions for adding, editing, deleting, and querying each one of the convertible data areas.

Links to NX Objects

You can link a UDO to other NX objects using any one of five different linking mechanisms. Each of the five link mechanisms has features that affect the behavior of NX during update and delete events.

If:	*UDO is Deleted*		*Associated Object is Deleted*		*UDO is Updated*	*Associated Object is Updated*
What happens to:	*Link*	*Associated Object*	*Link*	*UDO*	*Associated Object*	*UDO*
Link type 1	Removed	Nothing	Removed	Deleted	Nothing*	Updated
Link type 2	Removed	Deleted	Nothing	Nothing	Nothing*	Nothing
Link type 3	Removed	Nothing	Removed	Updated	Nothing*	Updated
Link type 4	Removed	Nothing	Removed	Nothing	Nothing*	Nothing
Owning Link	Removed	Nothing (if AssocObj is a solid body) /	NA-The AssocObj cannot be deleted		Nothing*	Nothing

		Deleted (otherwise)	directly		

*However, you can cause the associated object to update by registering an update function to the UDO class.

Note:

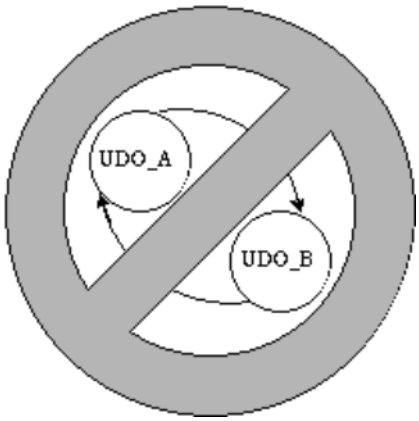
Not every object can be linked to a UDO with every available link type. To check if a given object may be linked to a UDO with a given link type use the IsObjectLinkable method on the UserDefinedObjectManager class.

Known restrictions for Link Type 2

Link Type 2 can not be used to link User Defined Objects to features, solid faces, or solid edges

Note:

Cyclical links (regardless of type) are not allowed. For example, if UDO_A is linked to UDO_B, then UDO_B should not be linked back to UDO_A.



Cyclical links are not allowed

Example VB Link Program:

```
 '-------------------------------------------------------------------------
---------
' This program allows you to test all of the various link types.
'
' Open a part that contains multiple smart points and execute
' this program. It will prompt you to select one of the points,
' and will then ask you to enter a number (0-4) to indicate
' the link type you wish to test.
'
' 0 = Owning link
' 1 = Type 1 link
' 2 = Type 2 link
' 3 = Type 3 link
' 4 = Type 4 link
'
' After selecting the point an link type, you will see a new
```

```
' UDO in the part that links to the selected point via the
' specified link type. The UDO is displayed as a Circle.
' The color and font of the UDO will vary depending on the
' link type you chose, it will also have text near the center
' to indicated what link type is being tested by that UDO.
'
' You may now edit the point (to see if the UDO goes through
' update when the point updates) or try to delete the point
' to see what happens to the linked UDO.
'
' After testing one link type for a while, you may decide to
' create a new UDO with a different link type to test, or
' you have the option of editing an existing UDO to use a
' new link type. If you right click on a UDO in the screen
' it will give you the option to "Edit User Defined Object".
' If you choose to edit the UDO, it will prompt you to choose
' a new link type for your UDO.
'
' NOTE: owning links have special properties for selection,
' so you won't be able to right click on the owning link UDO
' and choose edit. However, you can still edit the owning link
' UDOs. Go to the menu at the top and choose
' Edit->User Defined Object. This will bring up the selection
' dialog and allow you to choose the UDO(s) you wish to edit
' including the owning link UDOs.
'
'------------------------------------------------------------------------
--------

Option Strict Off
Imports System
Imports NXOpen
Imports NXOpen.UF Module

Module1
    Dim myUDOclass As UserDefinedObjects.UserDefinedClass = Nothing
    Dim theSession As Session = Nothing
    '------------------------------------------------------------------
--------
    ' Callback: myDisplayCB ' This is a callback method associated with
displaying a UDO.
    ' This same callback is registered for display, select, fit, and
    ' attention point.
    '------------------------------------------------------------------
--------
    Public Function myDisplayCB(ByVal displayEvent As
UserDefinedObjects.UserDefinedDisplayEvent) As Integer
            Try
                    Dim myUDODisplayString As String = ""
```

```
                    Dim myUDOLinks() As UserDefinedObjects.
UserDefinedObject.LinkDefinition = Nothing

                    ' What type of link did we store with this udo?
                    Dim myUDOints() As Integer =
displayEvent.UserDefinedObject.GetIntegers

                    ' Find the point we stored with the specified link
type
                    If myUDOints(0) = 0 Then
                            myUDOLinks =
displayEvent.UserDefinedObject.GetLinks
(UserDefinedObjects.UserDefinedObject.LinkType.Owning)
                            myUDODisplayString = "Own"
                    ElseIf myUDOints(0) = 1 Then
                            myUDOLinks =
displayEvent.UserDefinedObject.GetLinks
(UserDefinedObjects.UserDefinedObject.LinkType.Type1)
                            myUDODisplayString = "1"
                    ElseIf myUDOints(0) = 2 Then
                            myUDOLinks =
displayEvent.UserDefinedObject.GetLinks
(UserDefinedObjects.UserDefinedObject.LinkType.Type2)
                            myUDODisplayString = "2"
                    ElseIf myUDOints(0) = 3 Then
                            myUDOLinks =
displayEvent.UserDefinedObject.GetLinks
(UserDefinedObjects.UserDefinedObject.LinkType.Type3)
                            myUDODisplayString = "3"
                    ElseIf myUDOints(0) = 4 Then
                            myUDOLinks =
displayEvent.UserDefinedObject.GetLinks
(UserDefinedObjects.UserDefinedObject.LinkType.Type4)
                            myUDODisplayString = "4"
                    Else
                            ' There was no link type defined...
                            ' We should never get here, but if we do,
stop trying
                            ' to display this UDO.
                            Return 0
                    End If
                    If myUDOLinks.Length = 0 Then
                            ' The linked point was missing (the point
may have been deleted)
                            ' Don't bother to display any UDO's without
linked points.
                            Return 0
                    End If
```

```
                    ' The point linked to the UDO will define the
location of
                            ' the UDO we're about to display
                            Dim myPoint As Point =
myUDOLinks(0).AssociatedObject
                            Dim myPointCoordinates As Point3d =
myPoint.Coordinates
                            ' Draw some text at this location to
indicate the link type in use

    displayEvent.DisplayContext.DisplayText(myUDODisplayString,
myPointCoordinates, 0)
                            ' Draw a circle around the linked point in
the X-Z plane
                            ' First we must define a matrix to describe
the transform from
                            ' Absolute coordinates into the X-Z plane
this matrix is the
                            ' "rotation" matrix for our circle.
                            Dim myMatrix As Matrix3x3
                            myMatrix.Xx = 1
                            myMatrix.Xy = 0
                            myMatrix.Xz = 0
                            myMatrix.Yx = 0
                            myMatrix.Yy = 0
                            myMatrix.Yz = 1
                            myMatrix.Zx = 0
                            myMatrix.Zy = -1
                            myMatrix.Zz = 0
                            ' Now we must transform the origin of the
circle from Absolute coordinates
                            ' to the coordinates of the circle (ie apply
the rotation transform).
                            Dim xformedPoint As Point3d =
theSession.MathUtils.Multiply(myMatrix, myPointCoordinates)
                            ' Draw the circle now

    displayEvent.DisplayContext.DisplayCircle(xformedPoint, myMatrix,
20, False)
                    Catch ex As NXException
                            ' the display/selection/fit/attention
callback is called so many times
                            ' that it's best to print this error
handling stuff in the syslog
                            ' any interactive messages in the UI would
drive the user crazy ;)
                            Dim theUfSession As UFSession =
UFSession.GetUFSession()
```

```
                        theUfSession.UF.PrintSyslog("Caught
Exception in myDisplayCB: '" & ex.Message() & "'" & vbCrLf, False) End
Try myDisplayCB = 0
            End Function
            '------------------------------------------------------------
----------------
            ' Callback: myEditCB
            ' This is a callback method associated with editing a UDO.
            '------------------------------------------------------------
----------------
            Public Function myEditCB(ByVal editEvent As
UserDefinedObjects.UserDefinedEvent) As Integer
                  Try
                        Dim theUI As UI = UI.GetUI()
                        Dim myView As NXOpen.View = Nothing
                        Dim myCursor As Point3d
                        myCursor.X = 0
                        myCursor.Y = 0
                        myCursor.Z = 0

                        ' highlight the current udo we are about to
edit
                        ' this is helpful if multiple udo's were on
the selection
                        ' list when the user decided to edit them
                        editEvent.UserDefinedObject.Highlight()
                        ' Prompt the user to specify a new link type
                        Dim returnVal As Integer = 0
                        returnVal = InputBox("Enter a number between
0 and 4", "Select Link Type", returnVal)

                        ' We're done asking the user for input,
                        ' it is safe to unhighlight the udo now

                        editEvent.UserDefinedObject.Unhighlight()
                        ' Validate the user's input
                        If returnVal < 0 Then
                              MsgBox("Invalid input - UDO was not
edited", MsgBoxStyle.OkOnly)
                              Return 0
                        End If
                        If returnVal > 4 Then
                              MsgBox("Invalid input - UDO was not
edited", MsgBoxStyle.OkOnly)
                              Return 0
                        End If
                        ' Verify that the user selected something
different from the original value
```

```
                            Dim myUDOints() As Integer =
editEvent.UserDefinedObject.GetIntegers
                            If myUDOints(0) = returnVal Then
                                    MsgBox("UDO link type (" &
myUDOints(0) & ") was unchanged", MsgBoxStyle.OkOnly)
                                    Return 0
                            End If
                            ' We have a new link type, so remove the old
linked objects from the UDO
                            Dim myLinks() As
UserDefinedObjects.UserDefinedObject.LinkDefinition = Nothing
                            If myUDOints(0) = 0 Then
                                    myLinks =
editEvent.UserDefinedObject.PopLinks(UserDefinedObjects.UserDefinedObje
ct.LinkType.Owning, 1)
                            ElseIf myUDOints(0) = 1 Then
                                    myLinks =
editEvent.UserDefinedObject.PopLinks(UserDefinedObjects.UserDefinedObje
ct.LinkType.Type1, 1)
                            ElseIf myUDOints(0) = 2 Then
                                    myLinks =
editEvent.UserDefinedObject.PopLinks(UserDefinedObjects.UserDefinedObje
ct.LinkType.Type2, 1)
                            ElseIf myUDOints(0) = 3 Then
                                    myLinks =
editEvent.UserDefinedObject.PopLinks(UserDefinedObjects.UserDefinedObje
ct.LinkType.Type3, 1)
                            ElseIf myUDOints(0) = 4 Then
                                    myLinks =
editEvent.UserDefinedObject.PopLinks(UserDefinedObjects.UserDefinedObje
ct.LinkType.Type4, 1)
                            End If
                            ' Update the status of the link object we
just removed
                            myLinks(0).Status =
UserDefinedObjects.UserDefinedObject.LinkStatus.UpToDate

                            ' Store the integer selected by the user
with the udo.
                            ' This integer will indicate the link type
we are testing
                            ' for this given UDO
                            myUDOints(0) = returnVal

    editEvent.UserDefinedObject.SetIntegers(myUDOints)

                            ' Set the display properties so users can
"see" what
```

```
                            ' link type is used for the UDO and add the
old link objects
                            ' back onto the udo with the new link type
                            If returnVal = 0 Then
                                    editEvent.UserDefinedObject.SetLinks
(UserDefinedObjects.UserDefinedObject.LinkType.Owning, myLinks)
                            editEvent.UserDefinedObject.LineFont =
DisplayableObject.ObjectFont.Solid
                            editEvent.UserDefinedObject.LineWidth =
DisplayableObject.ObjectWidth.Normal
                            editEvent.UserDefinedObject.Color = 186
                        ElseIf returnVal = 1 Then
                            editEvent.UserDefinedObject.SetLinks
(UserDefinedObjects.UserDefinedObject.LinkType.Type1, myLinks)
                            editEvent.UserDefinedObject.LineFont =
DisplayableObject.ObjectFont.Dashed
                            editEvent.UserDefinedObject.LineWidth =
DisplayableObject.ObjectWidth.Normal
                            editEvent.UserDefinedObject.Color = 36
                        ElseIf returnVal = 2 Then
                            editEvent.UserDefinedObject.SetLinks
(UserDefinedObjects.UserDefinedObject.LinkType.Type2, myLinks)
                            editEvent.UserDefinedObject.LineFont =
DisplayableObject.ObjectFont.Dotted
                            editEvent.UserDefinedObject.LineWidth =
DisplayableObject.ObjectWidth.Normal
                            editEvent.UserDefinedObject.Color = 36
                        ElseIf returnVal = 3 Then
                            editEvent.UserDefinedObject.SetLinks
(UserDefinedObjects.UserDefinedObject.LinkType.Type3, myLinks)
                            editEvent.UserDefinedObject.LineFont =
DisplayableObject.ObjectFont.Dashed
                            editEvent.UserDefinedObject.LineWidth =
DisplayableObject.ObjectWidth.Thick
                            editEvent.UserDefinedObject.Color = 211
                        ElseIf returnVal = 4 Then
                            editEvent.UserDefinedObject.SetLinks
(UserDefinedObjects.UserDefinedObject.LinkType.Type4, myLinks)
                            editEvent.UserDefinedObject.LineFont =
DisplayableObject.ObjectFont.Dotted
                            editEvent.UserDefinedObject.LineWidth =
DisplayableObject.ObjectWidth.Thick
                            editEvent.UserDefinedObject.Color = 211
                        End If

                        ' Add the udo to the display list manually.
                        ' This will force the udo to display immediately
                        Dim theUfSession As UFSession =
UFSession.GetUFSession()
```

```
    theUfSession.Disp.AddItemToDisplay(editEvent.UserDefinedObject.Tag(
))

            Catch ex As NXException
                    Dim theLW As ListingWindow =
theSession.ListingWindow
                    theLW.Open()
                    theLW.WriteLine("Caught Exception in myEditCB: '" &
ex.Message() & "'")
            End Try
            myEditCB = 0
    End Function

            '------------------------------------------------------------
----------------
            ' Callback: myUpdateCB
            ' This is a callback method allows you to define the update
            ' behavior of your UDO. It is only used for specific link
types.
            '
            ' If your associated object goes through update and it was
linked via:
            ' * owning link - this callback is NOT used
            ' * type 1 link - this callback is executed
            ' * type 2 link - this callback is NOT used
            ' * type 3 link - this callback is executed
            ' * type 4 link - this callback is NOT used
            '------------------------------------------------------------
----------------
            Public Function myUpdateCB(ByVal updateEvent As
UserDefinedObjects.UserDefinedLinkEvent) As Integer
                    Dim theLW As ListingWindow =
theSession.ListingWindow
                    theLW.Open()
                    Try
                            ' Print information to the listing window
about what
                            ' triggered this call to update the udo
                            Dim theUfSession As UFSession =
UFSession.GetUFSession()
                            If updateEvent.AssociatedObject Is Nothing
Then
                                    theLW.WriteLine("Inside Update
Callback for UDO (" & updateEvent.UserDefinedObject.Tag() & ") with
linktype " & updateEvent.LinkType & " to NULL object")
                            Else
                                    Dim assocObjStatus As Integer =
theUfSession.Obj.AskStatus(updateEvent.AssociatedObject.Tag())
```

```
                                    If assocObjStatus = 0 Then
                                            theLW.WriteLine("Inside
Update Callback for UDO (" & updateEvent.UserDefinedObject.Tag() & ")
with linktype " & updateEvent.LinkType & " to DELETED object (" &
updateEvent.AssociatedObject.Tag() & ")")
                                    ElseIf assocObjStatus = 1 Then
                                            theLW.WriteLine("Inside
Update Callback for UDO (" & updateEvent.UserDefinedObject.Tag() & ")
with linktype " & updateEvent.LinkType & " to TEMPORARY object (" &
updateEvent.AssociatedObject.Tag() & ")")
                                    ElseIf assocObjStatus = 2 Then
                                            theLW.WriteLine("Inside
Update Callback for UDO (" & updateEvent.UserDefinedObject.Tag() & ")
with linktype " & updateEvent.LinkType & " to CONDEMNED object (" &
updateEvent.AssociatedObject.Tag() & ")")
                                    Else
                                            theLW.WriteLine("Inside
Update Callback for UDO (" & updateEvent.UserDefinedObject.Tag() & ")
with linktype " & updateEvent.LinkType & " to ALIVE object (" &
updateEvent.AssociatedObject.Tag() & ")")
                                    End If
                            End If theLW.WriteLine(" ")
                    Catch ex As NXException
                            theLW.WriteLine("Caught Exception in
myUpdateCB: '" & ex.Message() & "'")
                    End Try
                    myUpdateCB = 0
            End Function

            '-------------------------------------------------------------
----------------
            ' Callback: myDeleteCB
            ' This is callback is invoked whenever your linked objects
get deleted.
            ' When the linked object is deleted different things can
happen
            ' depending on the link type used.
            '
            ' If your associated object goes through delete and it was
linked via:
            ' * owning link - Not Available - you can not delete an
object
            ' when it's owned by a UDO (unless that object is
            ' a solid body, but this example links to points
            ' so we don't have to worry about that case).
            ' * type 1 link - After executing this callback the UDO
itself is deleted
            ' along with the point.
```

```
            ' * type 2 link - The linked object isn't really deleted
it's marked as
            ' "condemned" and then this callback is invoked.
            ' * type 3 link - After executing this callback the point
is deleted, the
            ' link is removed from the UDO, and UDO itself is updated.
            ' * type 4 link - After executing this callback the point
is deleted, and
            ' the link is removed from the UDO.
            '
            ' NOTE: With the way this particular UDO's display callback
was designed,
            ' if the links are removed from the UDO, it will no longer
be visible.
            ' Just because you don't see it anymore, does not necessary
mean
            ' the UDO has been deleted (as what happens with both link
type 3
            ' and link type 4). UDO's with link type 1 are the only
UDO's to
            ' commit suicide (delete themselves) whenever the linked
object gets ' deleted.
            '-------------------------------------------------------
----------------
            Public Function myDeleteCB(ByVal updateEvent As
UserDefinedObjects.UserDefinedLinkEvent) As Integer
                    Dim theLW As ListingWindow =
theSession.ListingWindow
                    theLW.Open()
                    Try
                            ' Print information to the listing window
about what
                            ' triggered this call to the delete callback
of the udo
                            If updateEvent.AssociatedObject Is Nothing
Then
                                    theLW.WriteLine("Inside Delete
Callback for UDO (" & updateEvent.UserDefinedObject.Tag() & ") with
linktype " & updateEvent.LinkType & " and NULL link object")
                            Else
                                    Dim theUfSession As UFSession =
NXOpen.UF.UFSession.GetUFSession()
                                    Dim assocObjStatus As Integer =
theUfSession.Obj.AskStatus(updateEvent.AssociatedObject.Tag())
                                    If assocObjStatus = 0 Then
                                            theLW.WriteLine("Inside
Delete Callback for UDO (" & updateEvent.UserDefinedObject.Tag() & ")
with linktype " & updateEvent.LinkType & " to DELETED object (" &
updateEvent.AssociatedObject.Tag() & ")")
```

```
                                    ElseIf assocObjStatus = 1 Then
                                            theLW.WriteLine("Inside
Delete Callback for UDO (" & updateEvent.UserDefinedObject.Tag() & ")
with linktype " & updateEvent.LinkType & " to TEMPORARY object (" &
updateEvent.AssociatedObject.Tag() & ")")
                                    ElseIf assocObjStatus = 2 Then
                                            theLW.WriteLine("Inside
Delete Callback for UDO (" & updateEvent.UserDefinedObject.Tag() & ")
with linktype " & updateEvent.LinkType & " to CONDEMNED object (" &
updateEvent.AssociatedObject.Tag() & ")")
                                    Else
                                            theLW.WriteLine("Inside
Delete Callback for UDO (" & updateEvent.UserDefinedObject.Tag() & ")
with linktype " & updateEvent.LinkType & " to ALIVE object (" &
updateEvent.AssociatedObject.Tag() & ")")
                                    End If
                            End If
                            If updateEvent.LinkType = 0 Then
                                    theLW.WriteLine("This should not be
possible - you can't delete owned objects.")
                                    theLW.WriteLine("Note: owned solid
bodies can be deleted through modeling, but this demo links to points
not bodies.")
                            ElseIf updateEvent.LinkType = 1 Then
                                    theLW.WriteLine("The UDO will now
delete itself.")
                            ElseIf updateEvent.LinkType = 2 Then
theLW.WriteLine("The point was not deleted it is now condemned (ie
invisible, but still exists in the part).")
                            ElseIf updateEvent.LinkType = 3 Then
                                    theLW.WriteLine("We will soon break
the link in the UDO and force the UDO to go through update")
                                    theLW.WriteLine("Without the linked
point, this UDO's display callback won't do anything.")
                                    theLW.WriteLine("You will no longer
be able to see this UDO on the screen, but it still exists in the
part.")
                            ElseIf updateEvent.LinkType = 4 Then
                                    theLW.WriteLine("We will soon break
the link in the UDO.") theLW.WriteLine("Without the linked point, this
UDO's display callback won't do anything.")
                                    theLW.WriteLine("You will no longer
be able to see this UDO on the screen, but it still exists in the
part.")
                                    theLW.WriteLine("Also note you may
need to regenerate the display before the UDO disappears.")
                                    theLW.WriteLine("Go to View->Layout-
>Regenerate.")
                            End If
```

```
                                theLW.WriteLine(" ")
                Catch ex As NXException
                                theLW.WriteLine("Caught Exception in
myDeleteCB: '" & ex.Message() & "'")
                End Try
                myDeleteCB = 0
        End Function

    '-------------------------------------------------------------------
--------
    ' initUDO ' Checks to see which (if any) of the application's
static
    ' variables are uninitialized, and sets them accordingly.
    ' Initializes the UDO class and registers all of its
    ' callback methods.
    '-------------------------------------------------------------------
--------
    Public Function initUDO(ByVal alertUser As Boolean) As Integer
            Try
                If theSession Is Nothing Then
                        theSession = Session.GetSession()
                End If
                If myUDOclass Is Nothing Then
                        If alertUser = True Then
                                MsgBox("Registering VB UDO Class",
MsgBoxStyle.OkOnly)
                        End If
                        myUDOclass =
theSession.UserDefinedClassManager.CreateUserDefinedObjectClass("VB_Lin
k_Test_UDO", "VB Link Test UDO")
                        myUDOclass.AllowQueryClassFromName =
UserDefinedObjects.UserDefinedClass.AllowQueryClass.On
                        myUDOclass.AllowOwnedObjectSelectionOption =
UserDefinedObjects.UserDefinedClass.AllowOwnedObjectSelection.On
                        myUDOclass.AddDisplayHandler(AddressOf
myDisplayCB)

    myUDOclass.AddAttentionPointHandler(AddressOf myDisplayCB)
                        myUDOclass.AddFitHandler(AddressOf
myDisplayCB)
                        myUDOclass.AddSelectionHandler(AddressOf
myDisplayCB)
                        myUDOclass.AddEditHandler(AddressOf
myEditCB)
                        myUDOclass.AddUpdateHandler(AddressOf
myUpdateCB)
                        myUDOclass.AddDeleteHandler(AddressOf
myDeleteCB)
                        Dim theUI As UI = UI.GetUI()
```

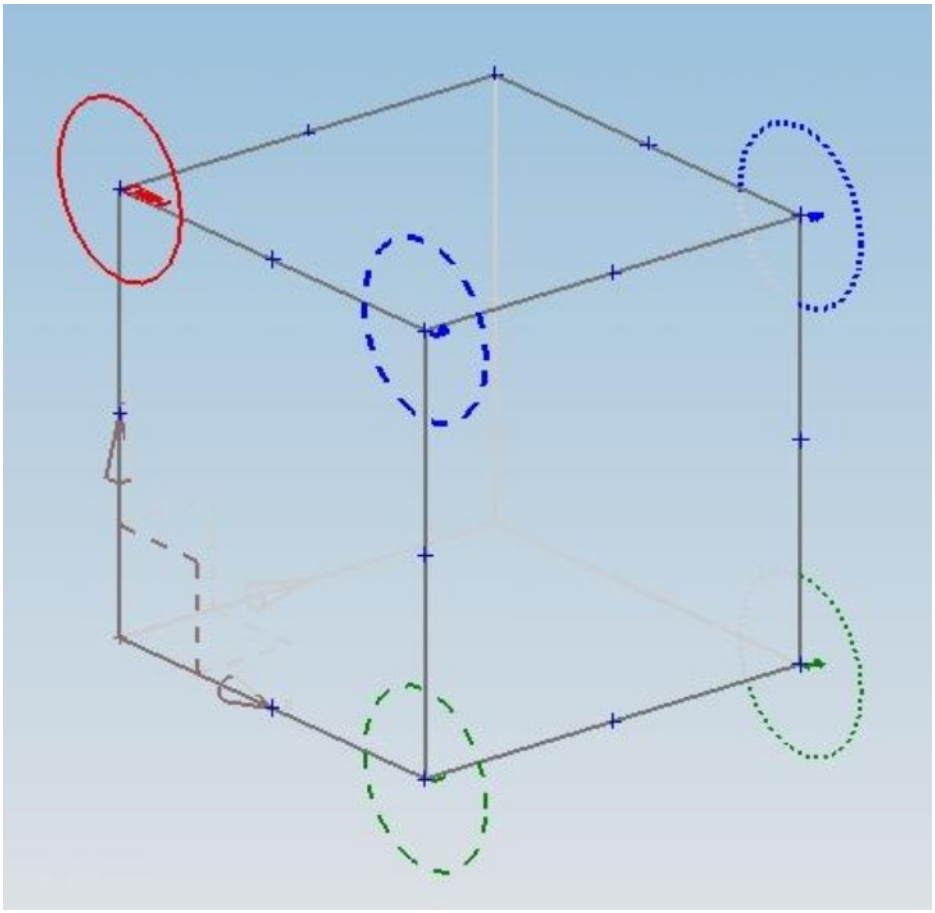
```
    theUI.SelectionManager.SetSelectionStatusOfUserDefinedClass(myUDOcl
ass, True)
                    End If
            Catch ex As NXException
                    ' We may be initializing the UDO class during NX
Startup
                    ' Print any error messages directly to the syslog
                    Dim theUfSession As UFSession =
UFSession.GetUFSession()
                    theUfSession.UF.PrintSyslog("Caught Exception in
initUDO: '" & ex.Message() & "'" & vbCrLf, False)
            End Try
            initUDO = 0
    End Function
    '------------------------------------------------------------------
--------
    ' NX Startup
    ' Startup entrypoint used when program is loaded automatically
    ' as NX starts up. Note this application must be placed in a
    ' special folder for NX to find and load it during startup.
    ' Refer to the NX Open documentation for more details on how
    ' NX finds and loads applications during startup.
    '------------------------------------------------------------------
--------
    Public Function Startup() As Integer
            initUDO(False)
            Startup = 0
    End Function ' Startup ends
    '------------------------------------------------------------------
--------
    ' Explicit Activation
    ' This entry point is used to activate the application explicitly
    ' via File->Execute->NX Open...
    '------------------------------------------------------------------
--------
    Sub Main()
            Try
                    ' in case we didn't load this dll at startup...
                    ' attempt to initialize the UDO class
                    initUDO(True)

                    Dim theUI As UI = UI.GetUI()
                    Dim theUfSession As UFSession =
UFSession.GetUFSession()

                    ' if we don't have any parts open create one
                    Dim myBasePart As BasePart =
theSession.Parts.BaseDisplay
```

```
                    If myBasePart Is Nothing Then
                            myBasePart =
theSession.Parts.NewBaseDisplay("test_vb_udo.prt",
BasePart.Units.Millimeters)
                    End If

                    Dim myView As NXOpen.View = Nothing
                    Dim myCursor As Point3d
                    myCursor.X = 0
                    myCursor.Y = 0
                    myCursor.Z = 0

                    ' Prompt user to select a point
                    Dim selectedPoint As NXObject = Nothing
                    Dim mask(0) As Selection.MaskTriple
                    mask(0).Type = NXOpen.UF.UFConstants.UF_point_type
                    mask(0).Subtype =
NXOpen.UF.UFConstants.UF_point_subtype
                    theUI.SelectionManager.SelectObject("Select point to
link to UDO", "Select point",
                            Selection.SelectionScope.WorkPart, _
Selection.SelectionAction.ClearAndEnableSpecific, False, False, mask, _
                                    selectedPoint, myCursor)
                            If selectedPoint Is Nothing Then Return
                            ' Prompt user to select a link type to test
                            ' (Owning, Type 1, Type 2, Type 3, or Type
4)
                            Dim returnVal As Integer
                            returnVal = 0 returnVal = InputBox("Enter a
number between 0 and 4", "Select Link Type", returnVal)
                            ' Validate the input
                            If returnVal < 0 Then
                                    Return
                            End If
                            If returnVal > 4 Then
                                    Return
                            End If
                            ' The user selected a valid point and link
type
                            ' go ahead and create the udo
                            Dim myUDOmanager As
UserDefinedObjects.UserDefinedObjectManager =
myBasePart.UserDefinedObjectManager
                            Dim firstUDO As
UserDefinedObjects.UserDefinedObject =
myUDOmanager.CreateUserDefinedObject(myUDOclass)

                            ' set the display properties so users can
"see" what
```

```
                        ' link type is used for the udo and add the
old link objects
                        ' back onto the udo with the new link type
                        Dim myLinks(0) As
UserDefinedObjects.UserDefinedObject.LinkDefinition
                        myLinks(0).AssociatedObject = selectedPoint
                        myLinks(0).Status =
UserDefinedObjects.UserDefinedObject.LinkStatus.UpToDate
                        If returnVal = 0 Then
                                firstUDO.LineFont =
DisplayableObject.ObjectFont.Solid
                                firstUDO.LineWidth =
DisplayableObject.ObjectWidth.Normal
                                firstUDO.Color = 186
                                firstUDO.SetLinks
(UserDefinedObjects.UserDefinedObject.LinkType.Owning, myLinks)
                        ElseIf returnVal = 1 Then
                                firstUDO.LineFont =
DisplayableObject.ObjectFont.Dashed
                                firstUDO.LineWidth =
DisplayableObject.ObjectWidth.Normal
                                firstUDO.Color = 36 firstUDO.SetLinks
(UserDefinedObjects.UserDefinedObject.LinkType.Type1, myLinks)
                        ElseIf returnVal = 2 Then
                                firstUDO.LineFont =
DisplayableObject.ObjectFont.Dotted
                                firstUDO.LineWidth =
DisplayableObject.ObjectWidth.Normal
                                firstUDO.Color = 36
                                firstUDO.SetLinks
(UserDefinedObjects.UserDefinedObject.LinkType.Type2, myLinks)
                        ElseIf returnVal = 3 Then
                                firstUDO.LineFont =
DisplayableObject.ObjectFont.Dashed
                                firstUDO.LineWidth =
DisplayableObject.ObjectWidth.Thick
                                firstUDO.Color = 211
                                firstUDO.SetLinks
(UserDefinedObjects.UserDefinedObject.LinkType.Type3, myLinks)
                        Else
                                firstUDO.LineFont =
DisplayableObject.ObjectFont.Dotted
                                firstUDO.LineWidth =
DisplayableObject.ObjectWidth.Thick
                                firstUDO.Color = 211
                                firstUDO.SetLinks
(UserDefinedObjects.UserDefinedObject.LinkType.Type4, myLinks)
                        End If
```

```
                            ' store the integer selected by the user
with the udo
                            ' this integer will indicate the link type
we are testing
                            ' for this given udo
                            Dim myUDOints(0) As Integer
                            myUDOints(0) = returnVal
                            firstUDO.SetIntegers(myUDOints)

                            ' add the udo to the display list manually
                            ' this will force the udo to display
immediately
                            theUfSession =
NXOpen.UF.UFSession.GetUFSession()

    theUfSession.Disp.AddItemToDisplay(firstUDO.Tag())
                    Catch ex As NXException
                            Dim theLW As ListingWindow =
theSession.ListingWindow
                            theLW.Open()
                            theLW.WriteLine("Caught Exception in Main:
'" & ex.Message() & "'")
                    End Try
            End Sub

            '------------------------------------------------------------
----------------
    ' GetUnloadOption
    ' Tells NX when to unload this application.
    ' This MUST return AtTermination because we have UDO callbacks
    ' defined in this program. Otherwise NX could try to call
    ' one of the UDO callbacks, after it had already unloaded
    ' the application.
    '------------------------------------------------------------------
--------
    Public Function GetUnloadOption(ByVal dummy As String) As Integer

            'Unloads the image when the NX session terminates
            GetUnloadOption =
NXOpen.Session.LibraryUnloadOption.AtTermination
    End Function
End Module
```

This program lets you link to existing points in a part with any of the 5 link types. Then by editing and deleting the points you can force the UDO's update and delete callbacks to get invoked. This will help you learn how the various link types work in NX.



The image above shows a part with a block and many points. UDO's with links to some of the points were created via the example VB link program above. The red UDO with the solid font on the left has an owning link to the point at the center of the circle. The green dashed UDO at the bottom center of the image uses a link type 1. The green dotted UDO at the bottom right uses a link type 2. The blue dashed UDO at the top center uses a type 3 link. Last the blue dotted UDO on the top right uses a link type 4. NOTE: the actual colors may vary when you run the program depending on the color palette you have loaded in your session.

UDO Owning Link

Selection with Owning Links:

A UDO Owning Link links a UDO with an NX object. A UDO with owning links has some unique interactive selection qualities. Interactively, when you select an NX object, that is referenced to a UDO by an owning link, you are directly selecting BOTH the associated UDO and the NX object itself. For example, if you perform an Information Object on an point that is referenced by an UDO owning link, then the information you receive is from both the associated UDO and the point.

There are times where you may not want to select both the owned object, and the owning UDO. For example, if your UDO computes the knot points of a spline, and then creates the spline as an owned object, you may want to allow selection of the spline so that it can be used in modeling to create an extrusion. In this case you would want to set your UDO class to allow the selection of

owned objects. This option is controlled by the AllowOwnedObjectSelectionOption property on the UserDefinedClass.

If AllowOwnedObjectSelectionOption is set to "On" and you select the owned object, if the owning UDO is also selectable, the Quick Pick dialog will pop up and allow you to select either the owned object or the UDO. Now if you select the UDO, you will in effect select both the owned object and the owning UDO (assuming the class selection filters allow their selections). If the UDO is not eligible for selection, picking the owned object will select the owned object only, and the Quick Pick dialog will not show the UDO in the list of selectable objects at that location. Notice that the class selection filter can still be used to filter out selections. If the filter is set to filter out the type of the owned object, then picking the UDO will result in the selection of only the UDO and the owned object will not be selected.

Additional Rules for Owning Links:

The UDO owning links have similar properties to link types 1-4 which are as follows:

- If the UDO is deleted, then the link between the UDO and the associated object is removed and the associated object itself is deleted, if it is not a solid body. If the owned object is a solid, then only the link is broken, and the solid body is left alone.
- The associated object cannot be deleted directly. However if the owned object is a solid, then the solid may still be selected through modeling selection where the feature list is presented to the user. For example, it may be deleted through EditFeatureDelete Feature. This is because ultimately modeling is in control of all solid bodies.
- If the UDO is updated, then the associated object is unaffected. However, you can cause the associated object to update by registering an update function to the UDO class. Then you can make changes to the associated object, in the UDO update method.
- If an associated object is updated, the UDO is unaffected.

A UDO can have more than one owning link. For example, UDO_A can have owning links to the following NX objects: NX_obj1, NX_obj2, and NX_obj3. Selection of UDO_A will result in the selection of all three objects, provided three are all selectable. Although a UDO can have owning links to more than one NX object, the converse is not true. An NX object can only be owned by one UDO.

UDOs can own other UDOs, so you can have the chain:

UDO1 owns UDO2 owns UDO3 owns point1

Events for UDOs

UDOs can participate in certain NX events through registered callback functions which are referred to as methods. The methods are function pointers which are arguments to the callback registration functions.

There are many events that trigger callbacks for UDO participation:

- Display - the three different display events include: self display (including plotting and CGM file creation), attention point assignment, and fit to view.
- Selection - the selection callback should be the same as the display callback.
- Update - allows you to keep objects up to date as the model changes.

- Delete - allows you to realign data in the free form and convertible data areas as associated objects are removed from the data model.
- Edit - allows custom code to execute when the user attempts to edit the UDO.
- Information - allows custom code to execute when the user attempts to query information from the UDO.

Display

NX has no knowledge of how or where a UDO should be displayed. Therefore if you want something visible in the screen to indicate the presence of a UDO, you must define and register your own display callback. If any of the display callbacks (self display, attention point, and fit) are to be used, then all of them should be used and the same callback may be registered for all three events.

Examples:

	Code (register display methods)
VB	`myUDOclass.AddDisplayHandler(AddressOf myDisplayCB)` `myUDOclass.AddAttentionPointHandler(AddressOf myDisplayCB) myUDOclass.AddFitHandler(AddressOf myDisplayCB)`

C#	`myUDOclass.AddDisplayHandler(new UserDefinedClass.DisplayCallback(Program.myDisplayCB));`
	`myUDOclass.AddAttentionPointHandler(new UserDefinedClass.DisplayCallback(Program.myDisplayCB));`
	`myUDOclass.AddFitHandler(new UserDefinedClass.DisplayCallback(Program.myDisplayCB));`

C++	`myUDOclass->AddDisplayHandler(make_callback(&myDisplayCB));`
	`myUDOclass->AddAttentionPointHandler(make_callback(&myDisplayCB));`
	`myUDOclass->AddFitHandler(make_callback(&myDisplayCB));`

Java	`// be sure to declare your class so that it knows about the callback implementation`
	`public class SimpleJavaUDO implements nxopen.userdefinedobjects.UserDefinedClass.DisplayCallback, nxopen.userdefinedobjects.UserDefinedClass.GenericCallback`
	`// then use code like this to actually register the class with the callbacks`
	`myUDOclass.addDisplayHandler(this);`
	`myUDOclass.addAttentionPointHandler(this);`
	`myUDOclass.addFitHandler(this);`

The actual display is processed as a result of a series of primitive display routines within the context of the callback. The input data for the primitive display functions is with respect to the absolute coordinate system. The only exceptions to this rule are the Arc and Circle. The Arc and Circle display function take in a matrix to define the orientation of the plane containing the Arc/Circle (use the identity matrix if you want the Arc/Circle displayed in the XY plane). The centerpoint of the circle is the original point in absolute coordinates, trasformed by the rotation matrix.

The primitive display functions live on the UserDefinedObjectDisplayContext object and include:

- DisplayArc
- DisplayCircle
- DisplayPolyline

- DisplayPoints
- DisplayPolygon
- DisplayText
- DisplayFacets

Examples:

	Code (display circle)
VB	```' Draw a circle around the linked point in the X-Z plane ' First we must define a matrix to describe the transform from ' Absolute coordinates into the X-Z plane this matrix is the ' "rotation" matrix for our circle. Dim myMatrix As Matrix3x3 myMatrix.Xx = 1 myMatrix.Xy = 0 myMatrix.Xz = 0 myMatrix.Yx = 0 myMatrix.Yy = 0 myMatrix.Yz = 1 myMatrix.Zx = 0 myMatrix.Zy = -1 myMatrix.Zz = 0 ' Now we must transform the origin of the circle from Absolute coordinates ' to the coordinates of the circle (ie apply the rotation transform). Dim xformedPoint As Point3d = theSession.MathUtils.Multiply(myMatrix, myPointCoordinates) ' Draw the circle now displayEvent.DisplayContext.DisplayCircle(xformedPoint, myMatrix, 20, False)```
	Code (display polyline)
VB	```Dim myPoints(3) As Point3d myPoints(0).X = myUDOdoubles(0) + 0 myPoints(0).Y = myUDOdoubles(1) + 0 myPoints(0).Z = myUDOdoubles(2) + 0 myPoints(1).X = myUDOdoubles(0) + 100 myPoints(1).Y = myUDOdoubles(1) + 0 myPoints(1).Z = myUDOdoubles(2) + 0 myPoints(2).X = myUDOdoubles(0) + 0 myPoints(2).Y = myUDOdoubles(1) + 100 myPoints(2).Z = myUDOdoubles(2) + 0 myPoints(3).X = myUDOdoubles(0) + 0 myPoints(3).Y = myUDOdoubles(1) + 0 myPoints(3).Z = myUDOdoubles(2) + 0 ' Display the triangle displayEvent.DisplayContext.DisplayPolyline(myPoints)```

C#	```Point3d[] myPoints = new Point3d[4];
	myPoints[0].X = myUDOdoubles[0] + 0;
	myPoints[0].Y = myUDOdoubles[1] + 0;
	myPoints[0].Z = myUDOdoubles[2] + 0;
	
	myPoints[1].X = myUDOdoubles[0] + 100;```

```
myPoints[1].Y = myUDOdoubles[1] + 0;
myPoints[1].Z = myUDOdoubles[2] + 0;

myPoints[2].X = myUDOdoubles[0] + 0;
myPoints[2].Y = myUDOdoubles[1] + 100;
myPoints[2].Z = myUDOdoubles[2] + 0;

myPoints[3].X = myUDOdoubles[0] + 0;
myPoints[3].Y = myUDOdoubles[1] + 0;
myPoints[3].Z = myUDOdoubles[2] + 0;

// Display the triangle
displayEvent.DisplayContext.DisplayPolyline(myPoints);
```

C++

```
std::vector<Point3d> myPoints(4);
myPoints[0].X = myUDOdoubles[0] + 0;
myPoints[0].Y = myUDOdoubles[1] + 0;
myPoints[0].Z = myUDOdoubles[2] + 0;

myPoints[1].X = myUDOdoubles[0] + 100;
myPoints[1].Y = myUDOdoubles[1] + 0;
myPoints[1].Z = myUDOdoubles[2] + 0;

myPoints[2].X = myUDOdoubles[0] + 0;
myPoints[2].Y = myUDOdoubles[1] + 100;
myPoints[2].Z = myUDOdoubles[2] + 0;

myPoints[3].X = myUDOdoubles[0] + 0;
myPoints[3].Y = myUDOdoubles[1] + 0;
myPoints[3].Z = myUDOdoubles[2] + 0;

// Display the triangle
displayEvent->DisplayContext()->DisplayPolyline(myPoints);
```

Java

```
Point3d[] myPoints = new Point3d[] {new Point3d(myUDOdoubles[0] +
0,
myUDOdoubles[1] + 0,
myUDOdoubles[2] + 0),
new Point3d(myUDOdoubles[0] + 100,
myUDOdoubles[1] + 0,
myUDOdoubles[2] + 0),
new Point3d(myUDOdoubles[0] + 0,
myUDOdoubles[1] + 100,
myUDOdoubles[2] + 0),
new Point3d(myUDOdoubles[0] + 0,
myUDOdoubles[1] + 0,
myUDOdoubles[2] + 0)};
// Display the triangle
e.displayContext().displayPolyline(myPoints);
```

Code (display text)

VB

```
Dim myPt As Point3d
myPt.X = myUDOdoubles(0) + 100
myPt.Y = myUDOdoubles(1) + 0
myPt.Z = myUDOdoubles(2) + 0
displayEvent.DisplayContext.DisplayText("VB .Net UDO", myPt, 0)
```

C#

```
Point3d myPt = new Point3d();
myPt.X = myUDOdoubles[0] + 100;
myPt.Y = myUDOdoubles[1] + 0;
myPt.Z = myUDOdoubles[2] + 0;
displayEvent.DisplayContext.DisplayText("C# UDO", myPt, 0);
```

C++

```
Point3d myPt = Point3d(myUDOdoubles[0] + 100, myUDOdoubles[1],
```

	myUDOdoubles[2]); displayEvent->DisplayContext()->DisplayText("C++ UDO", myPt, UserDefinedObjectDisplayContext::TextRefBottomLeft);

Java	Point3d myPt = new Point3d();
	myPt.x = myUDOdoubles[0] + 100;
	myPt.y = myUDOdoubles[1] + 0;
	myPt.z = myUDOdoubles[2] + 0;
	e.displayContext().displayText("JAVA UDO", myPt,
	UserDefinedObjectDisplayContext.TextRef.BOTTOM_LEFT);

Note:

The display callbacks should not perform any operations other than the primitive display functions listed above. In particular, you should never use the following routines in any display callback function:

- Any routine that Regenerates the Display
- Any routine that Queries or Sets the Work or Displayed Part
- Any routine that alters data in the part.

The same function can be used for all three methods because they all have the same basic code structure, and in most cases the same area that is occupied by the UDO on the screen for display is the same area you want defined for the fit and attention point operations. If you have a special circumstance where you really want different behavior for the three methods, the event reason can be used to determine the actual cause of the callback.

Selection

NX has no knowledge of how to select a UDO, or what set of points define the UDO (ie where must you place your mouse on the screen to select this UDO). Therefore if you want to be able to select your UDO, you must define and register your own selection callback. *The selection callback should be the same function you registered as your display callback.* If you use different functions for display and selection, then the set of points used to display your UDO may not be the same as the set of points your mouse must land on to select the UDO (this would lead to mass confusion of anyone trying to use your UDO).
Examples:

	Code (register selection method)
VB	myUDOclass.AddSelectionHandler(AddressOf myDisplayCB)
C#	myUDOclass.AddSelectionHandler(new UserDefinedClass.DisplayCallback(Program.myDisplayCB));
C++	myUDOclass->AddSelectionHandler(make_callback(&myDisplayCB));
Java	// be sure to declare your class so that it knows about the callback implementation public class SimpleJavaUDO implements nxopen.userdefinedobjects.UserDefinedClass.DisplayCallback,

```
nxopen.userdefinedobjects.UserDefinedClass.GenericCallback
// then use code like this to actually register the class with the
callbacks
myUDOclass.addSelectionHandler(this);
```

In addition to the registered selection method, you need to tell NX to enable selection for the UDO class. For example if you go to to Information Object the class selection dialog will launch. Choose the option to filter by types in this dialog, and you will see a list of all of the types available for selection. If you don't enable selection for this UDO class, then the UDO class will never appear in that list of available types, and UDO's of this class will never be selectable. Note the User Friendly name is the class name listed for a UDO in the filter by types dialog.

Examples:

	Code (enable selection)
VB	`theUI.SelectionManager.SetSelectionStatusOfUserDefinedClass(myUDOclass, True)`
C#	`theUI.SelectionManager.SetSelectionStatusOfUserDefinedClass(myUDOclass, true);`
C++	`theUI->SelectionManager()->SetSelectionStatusOfUserDefinedClass(myUDOclass, true);`
Java	`theUI.selectionManager().setSelectionStatusOfUserDefinedClass(myUDOclass, true);`

Note:

Owning links have special selection properties. If the UDO class is enabled for selection, and all of the UDO's in that class own NX objects, you can select the owned NX object, you can select the owned object and both the UDO and owned objects will be selected. Therefore if you used owned objects, the UDO can be selected even without having registered a selection callback.

Update

The mechanism within NX that keeps all of the objects up to date as the model changes is called the update mechanism.

As objects change in the data model, other objects need to be notified about those changes so they keep up to date with respect to the model. For example, if a dimension is assigned to the height of a block and the block's height changes, the dimension needs to be notified so that it can reflect the new height. As objects are involved in the update mechanism, they are examined for associations with other objects (such as UDOs linked via LinkType.Type1 and LinkType.Type3) that require those linked objects to be updated in addition to the original object. If any such associations are found, the associated objects are added to the list of items to be updated, and the process continues. (This is why cyclic relationships are not allowed as mentioned earlier in the Links section.) Each item on the update list is then accessed and processed during update. The update method available with UDOs is called when an object associated to a UDO with either a link type 1 or a link type 3 association passed through update. By definition associated

objects going through update that are linked to a UDO, with link type 2, 4, or with an owning link do not add the UDO to the update list. Therefore, an associated object with link type 2, or 4, or owning link does not invoke the update method.

As with any UDO callback method the update method must be registered in the NX session.

Examples:

	Code (register update method)
VB	`myUDOclass.AddUpdateHandler(AddressOf myUpdateCB)`
C#	`myUDOclass.AddUpdateHandler(new UserDefinedClass.LinkCallback(Program.myUpdateCB));`
C++	`myUDOclass->AddUpdateHandler(make_callback(&myUpdateCB));`
Java	`// be sure to declare your class so that it knows about the callback implementation public class MyJavaUDO implements nxopen.userdefinedobjects.UserDefinedClass.LinkCallback // then use code like this to actually register the class with the callbacks myUDOclass.addUpdateHandler(this);`

The update method has a UserDefinedLinkEvent input object to give you information about the reason the update method was called.

Refer to the Example VB Link Program above to see the implementation of an update callback named myUpdateCB.
If a UDO is linked to a number of NX objects using link types 1 or 3 and has an update callback registered, an update of the UDO will be caused by changes to any one (or more) of the linked NX objects. The update callback will be called only after all of the associated objects are updated. In other words, the UDO's registered callback is called only once per update cycle, regardless of how many associated objects are modified. The AssociatedObject property of the update callback's eventObject (update cause) argument contains only one of the object tags which may have changed. The callback should assume that everything that the UDO depends on has already changed and update itself accordingly.

While within the update callback, you may freely query the data model. You may also edit the free form data areas in the UDO. Additionally, you may display a dialog (if running an internal - non batch - program) to inform the NX user of the affect the edit may have on the UDO. If you display any dialogs be sure you use UI's LockAccess and UnlockAccess methods around the call to launch your dialog.
There are restrictions on the types of actions that can be performed during the context of this callback. These restrictions are necessary to keep the context of the system correct and because the update mechanism can not cope with recursive invocations. The restrictions are:

- Never explicitly invoke the update process again. This may occur by using Update.DoUpdate, or by creating a new feature.
- Never change the work part.

- Delete
-

- As objects associated to the UDO are deleted from the model, it may be necessary to realign data in the free form data areas or in the convertible data areas to reflect the removal of the associated objects from the data model. The notification during the delete callback should enable this realignment. Note that the associated objects may not be alive when this method is called. If you have UDO with a link type 2 and attempt to delete the associated object, when the delete callback is executed the associated object in the link event is already in a condemned state.
- Examples:

	Code (register delete method)
VB	`myUDOclass.AddDeleteHandler(AddressOf myDeleteCB)`
C#	`myUDOclass.AddDeleteHandler(new UserDefinedClass.LinkCallback(Program.myDeleteCB));`
C++	`myUDOclass->AddDeleteHandler(make_callback(&myDeleteCB));`
Java	`// be sure to declare your class so that it knows about the callback implementation public class MyJavaUDO implements nxopen.userdefinedobjects.UserDefinedClass.LinkCallback // then use code like this to actually register the class with the callbacks myUDOclass.addDeleteHandler(this);`

-
- Refer to the Example VB Link Program above to see the implementation of an delete callback named myDeleteCB.

Edit

-
- The edit callback allows you to define what it means to Edit your UDO. If you register an edit callback for your UDO, when you right-click on your UDO in the screen you will have the option to "Edit User Defined Object". If you do not register an edit callback for your class you will not have the option to "Edit User Defined Object" when you right click on your UDO. If you select "Edit User Defined Object" from the popup menu, your custom edit callback is executed. The edit callback may also be executed when the user goes to Edit→User Defined Object from the pull down menu and then selects a UDO of your class. If you launch any dialogs (other then the NX Object Selection dialogs) from your edit callback don't forget to use UI's LockAccess and UnlockAccess methods around the call to launch your dialog.
- Examples:

	Code (register edit method)
VB	`myUDOclass.AddEditHandler(AddressOf myEditCB)`
C#	`myUDOclass.AddEditHandler(new UserDefinedClass.GenericCallback(Program.myEditCB));`
C++	`myUDOclass->AddEditHandler(make_callback(&myEditCB));`
Java	`// be sure to declare your class so that it knows about the callback implementation public class SimpleJavaUDO implements nxopen.userdefinedobjects.UserDefinedClass.DisplayCallback, nxopen.userdefinedobjects.UserDefinedClass.GenericCallback // then use code like this to actually register the class with the callbacks`

	`myUDOclass.addEditHandler(this);`

Information

When the user selects Information→Object from the pull down menu default information listing all of the data stored with the UDO will display in the listing window. However if you define and register your own custom information callback, you can replace a portion of the default data with whatever information you would like to display about your UDO.

Examples:

	Code (register information method)
VB	`myUDOclass.AddInformationHandler(AddressOf myInfoCB)`
C#	`myUDOclass.AddInformationHandler(new UserDefinedClass.GenericCallback(Program.myInfoCB));`
C++	`myUDOclass-> AddInformationHandler(make_callback(&myInfoCB));`

Java	`// be sure to declare your class so that it knows about the callback implementation`
	`public class SimpleJavaUDO implements nxopen.userdefinedobjects.UserDefinedClass.DisplayCallback, nxopen.userdefinedobjects.UserDefinedClass.GenericCallback`
	`// then use code like this to actually register the class with the callbacks`
	`myUDOclass.addInformationHandler(this);`

UDO Status

An integer status value (1-7) is set for each UDO by NX in the absence of user defined methods. You can obtain the status with the query method GetUserDefinedObjectStatus on the UserDefinedObject. The description for each status value is given in the following table.

Status Value	UDO is out of date	Due to Addition or Deletion of Links to the UDO	Due to Update being performed on Associated Objects in the Absence of a UDO Method	Due to Deletion of Associated Objects in the Absence of a UDO Method
0	No			
1	Yes	Yes		
2	Yes		Yes	
3	Yes	Yes	Yes	
4	Yes			Yes
5	Yes	Yes		Yes
6	Yes		Yes	Yes
7	Yes	Yes	Yes	Yes

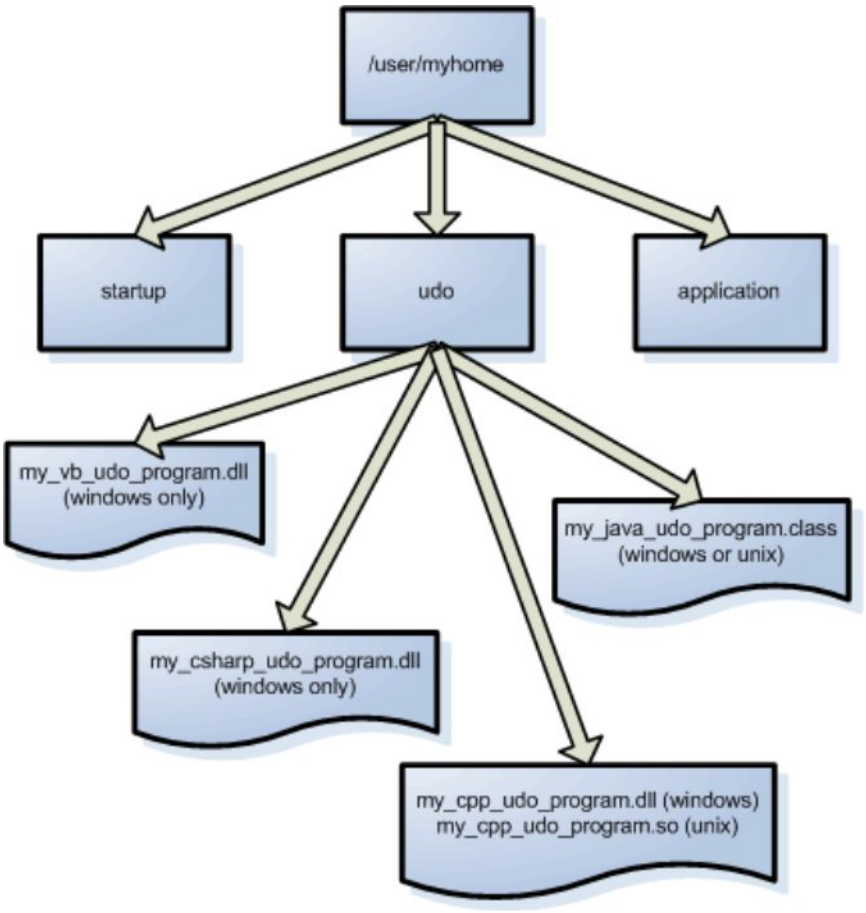
Note:

If user defined methods are used, then only status values 0 and 1 can occur.

Automatic Loading at Startup

Classes and methods can be automatically loaded into NX during system initialization for both an interactive session and an external NX Open program. To load the classes and methods automatically create a user directory with startup, application, and udo subdirectories. Once the user directory structure is created, add the base directory to the custom_dirs.dat file found in the $UGII_BASE_DIR/ugii/ugmenu. Then, place the libraries in the "udo" directory.

For example, if you add the directory "/user/myhome" to custom_dirs.dat, then you place your libraries in the directory "/user/myhome/udo".



Warning:

If you place a shared library in the udo directory for both internal (interactive) NX and for external (batch) programs, then your shared library cannot contain any calls to the internal only routines.

The directories present in the custom_dirs.dat file are processed in order, and the shared libraries found are loaded. In order to load the shared library, the library must contain the proper startup entry point.

Examples:

	Code (sample startup entry points)
VB	`Public Function Startup() As Integer`
C#	`public static int Startup()`
C++	`extern void ufsta( char *param, int *returnCode, int rlen )`
Java	`public static void startup (String [] args)throws NXException, java.rmi.RemoteException`

Additionally, your shared library must have the correct file extension for a particular platform as follows:

- Windows - .dll (or .class for java)
- Non—Windows .class for java)

Loading takes place during NX initialization. Thus, the loading of a shared library occurs before any parts are loaded, and (if in interactive mode) before a user interface is present.

Consequently, you should not:

- Place code within the ufsta function that attempts to access a part or the user interface.
- Attempt to load or create a part within the call to ufsta.

The creation of UDO classes and the registration of the methods is all we recommend during this period of the system's initialization.

Since classes can be created at an time during an NX session, you are not required to use this mechanism. However, this mechanism guarantees that classes and methods are present for and applied to all parts loaded within a session.

UDO Features

UDO's can also be features so that they can be time-stamped and updated in order with respect to other features in the model. Use the UserDefinedObjectFeatureBuilder to create a UDO feature.

The UDO links of type 1 and type 3 define parents of the UDO feature while the owned objects define its children. It is important to ensure that the UDO data model is correct with respect to timestamp creation. The primary difference between a UDO and a UDO feature is that the UDO updates last. For example, if you create links to modeling objects that were created after the UDO Feature you will get errors during update.
As you create features, the system assigns a time stamp for each feature. When a feature is modified, the update is controlled by the ordering of the timestamps.

Features are listed in the order in which they were created, as indicated by the time stamp (the number in parenthesis at the end of the name). The time stamp also indicates the order in which features will be evaluated when the model is updated.

For example

1. create feature BLOCK(0)
2. create feature TAPER(1)
3. create UDO

The UDO updates after the last feature to update, in this example, after TAPER(1). If we create feature HOLLOW(2), then the UDO updates after HOLLOW(2).

If the UDO references the edge of the block then it updates via the final state of this edge. That is, after the taper has been applied.

UDO features update in feature order.

For example,

1. create feature BLOCK(0)
2. create feature UDO(1)
3. create feature TAPER(2)

The UDO now updates right after BLOCK(0) and right before TAPER(2). If the UDO referenced the edge of a block, then it would update via the intermediate state of the edge (before the taper was applied). In this example, you would most likely have wanted to create a UDO and not a UDO feature since the final state of the model is probably what you want the UDO to reference.

However, if any of the owned objects of the UDO are going to be referenced by other features, then a UDO feature must be created or else the feature update will be incorrect.

For example:

1. create feature BLOCK(0)
2. create feature UDO(1)
3. create feature SWEEP(2)

In general, UDOs update after feature update and UDO features update during feature update.

If you wish to, you may use UDO callbacks. The UDO feature does not place any limitations of the implementation of callbacks.

Interactively, the UDO feature appears in the following feature menus:

- Info
- Edit Parameters
- Reorder
- Delete
- Suppress
- Unsuppress
- Suppress by Expression

If you choose to Edit Parameters for a UDO feature your custom edit callback is executed. If you do not have an edit callback registered, then an default edit parameters dialog is launched that contains any expressions linked to by the UDO.

Handling Errors and Help

Error Handling
UNDO

Error Handling

NX Open is designed to trap and report errors which prevent API methods from completing successfully. Errors may result from various reasons including the following:

- invalid input parameters
- request to generate invalid geometric models
- unexpected calling sequence (API methods called in wrong order)
- unavailable system resources such as memory or file access

In most cases each NX Open method is designed to return NX to a valid/complete state. However, there are logical groups of methods which must be used together. For example, after modifying expressions, model update must be called to rebuild the model using the new expression values. In case a valid logical group of API methods fail to complete successfully then NX Open provides several methods for responding to errors. The following outlines NX Open methods for detecting, reporting, handling and recovering from errors.

Note:

Accessing output parameters or calling NX Open methods after an API methods fails can result in undefined system behavior, including session or part corruption. For example, consider an NX Open method designed to return a pointer to an NX object. If the method fails the pointer will be invalid. Using the pointer could result in memory being overwritten with invalid values (essentially a random value could be written to a random memory location). System behavior will then depend on the location and value that was changed by mistake.

Error Detection

The first step in error handling is to detect and trap an error. Most NX Open methods are designed to return status and/or exceptions. Methods that do not return a specific status return an object or value which can be validated. It is very important for any exception or unexpected value to be trapped as soon as possible. Error detection is handled in two different ways depending on which NX Open API is being used.

Open C (User Function)

Almost all Open C functions are designed to return an integer value. By convention a returned value of zero indicates that the function completed successfully. If the return value is not zero then the function did not complete successfully and error reporting and recovery is required. It is important to test every return value for success or failure and to provide code to handle failures.

NX Open .NET, Java or C++ (Common API)

NX Open methods are designed to use the Try/Catch constructs. This method of error detection has the advantage of not requiring the programmer to test a return value after every method called. A logical block of code can be included in a Try block. If any method detects an error the Try block will immediately terminate and the Catch block is executed. The Catch block can then determine the type of exception and determine how to respond to the exception. Every logical block of code making NX Open method calls should be imbedded in a Try/Catch construct to ensure that all exceptions are handled appropriately.

Error Reporting

The second step in error handling is to report the error or at least log the error in a log file. It is a good idea to encapsulate the following error reporting steps into an error reporting method that is appropriate for the custom application. Typically programmers will want to create an error reporting method that is reused by many applications.

Error Reporting Steps

1. Obtain NX Open error message.
2. Build complete error report.
3. In most cases report the error to the user.
4. Always report the error to the log file.

Obtaining an Error Message

Obtaining a human-readable error message from NX Open is the first step to error reporting. NX Open provides methods to take a return code or an exception and produce a human-readable error message.

Open C (User Function)

For Open C , UF_get_fail_message(...) takes an error code and returns a human redable error message. To translate the message in native language supported by NX localization, use UF_TEXT_translate_string(..)

NX Open .NET, Java or C++ (Common API)

(NXException) contains a Message property that contatins a human readable error message.

Building an error report

The programmer should add information to the error report that will help determine the cause of the error. For instance, information about where the error occurred in the custom application (e.g. a function name) and specifically what the application was attempting to do when the error was detected. If error recovery is attempted and requires input from the user, then the user should be told specifically what they did wrong and what they can do to correct the problem.

Note:

It may also be necessary to build two types of error reports. One report could include only the information required by the user. Another report could contain debugging information specifically for the programmer.

Reporting an Error

Once the appropriate information has been gathered, the programmer must decide how to report the information. If the error is unexpected or critical, the user must be informed and given the option to exit the custom application without corrupting the NX session or the part. If the error is expected (see Error Handling and Error Recovery), then it may be appropriate to just log the error in the log file. It is suggested that all errors should be written to the log file even when the errors are reported directly to the user. This gives the programmer a record of the event. It is also possible for the programmer to design a custom error reporting dialog or to create a log file specifically for the custom application.

Error Recovery

Error recovery is a special case of error handling. If an unexpected error occurs then error handling typically consists of reporting the error, returning the internal state to a valid condition and then exiting the custom application to return control to NX. However, there are many types of errors and exceptions which should be anticipated by the programmer. A very simple example is asking the user to provide a file name. The programmer should anticipate that the user may make a mistake and that the file may not be found. The programmer should give the user the opportunity to enter another file name.

Another example of a failure that can be anticipated is a geometric modeling error. It may be that the user has specified design parameters that are invalid. For example, a blend radius that exceeds the limits of the faces being blended. When the blend operation fails it may be appropriate to continue the program by trying a different blend radius.

Error recovery is one of the most important system design challenges for any application development. Close attention must be given to maintaining a valid state for all data models.

NX Open supports error recovery in three ways:

1. Methods are designed to return exceptions to enable the programmer to detect failures.
2. When NX Open detects an error the NX Open methods are designed to return the NX session and part to a valid state.
3. The UNDO methods are provided to let the programmer easily return the NX session and part to a previous valid state.

Error and Exception Codes

In most cases, once a human-readable error message is obtained, error handling and recovery is the same for all types of errors from a given method. However, in some cases special error handling and recovery will depend on the specific type or reason for the error. The error code value or other information contain within an exception object is typically used to determine the specific type of error.

Error Handling - Language Specific Details

NX Open for C++
NX Open for .NET
NX Open Java

NX Open for C++

```
   int status = UF_MODL_create_block1( UF_NULLSIGN, corner_pts[i],
edge_lens,  &features[i]);
    //check for return value
    if (status != 0)
    {
        //get the human readable error message
                    UF_get_fail_message(status, ugErrorText);

        //report error to the user
        UF_UI_message_dialog("Dialog", UF_UI_MESSAGE_ERROR,
&ugErrorText, 1, .....  );

        //report error to syslog
```

```
        UF_print_syslog("Failed to create block\n", false);
        return 1;
    }
    status = UF_MODL_ask_feat_body(features[i], &blocks[i]);
    if (status != 0)
    {
       UF_print_syslog("Failed to get body from block\n", false);
       return;
    }
```

NX Open for .NET

```
Try
    Dim theSession As Session = Session.GetSession()
    Dim workPart As Part = theSession.Parts.Work

    Dim nullFeature As Features.Feature

    Dim blockBuilder As Features.BlockFeatureBuilder

    blockFeatureBuilder1 =
workPart.Features.CreateBlockFeatureBuilder(nullFeature)

    isDisposeCalled = False


Catch ex As NXException
   ' ---- Enter your exception handling code here -----

   ' ---- report the error in syslog, Message propery on the exception
object
          already has the human readable message ---

   theSession.LogFile.WriteLine("Failed to Create Block",  )

End Try
```

NX Open for Java

```
try

 {

    Session theSession =(Session)SessionFactory.get("Session");

    Part workPart = theSession.parts().work();

    nxopen.features.Feature nullFeatures_Feature = null;

    nxopen.features.BlockFeatureBuilder blockFeatureBuilder1;
```

```
    blockFeatureBuilder1 =
workPart.features().createBlockFeatureBuilder(nullFeatures_Feature);

 }

 catch (Exception e)
 {
    //report error to syslog, Message property on exception object
already has human
    //readable message
    theSession.Logfile().WriteLine("Failed to create block", +
e.getMessage());

 }
```

UNDO

The UNDO methods provide a very easy and powerful way to ensure that the NX session and parts are returned to a valid state. Conceptually using the UNDO methods are very simple. To use the UNDO methods the programmer first creates an UNDO Mark. This saves the current state of NX. If the program executes without error, the programmers calls the method to delete the undo mark. (unless the UNDO Mark is visible and being provided to let the user UNDO the operations of the custom application). If the program encounters an error and needs to recover, the programmer can return the NX state to that defined by the UNDO Mark.

Visible vs. Invisible UNDO Marks

The choice of visible or invisible UNDO mark depends on whether the programmer wants the user to be able to Undo the operations of an NXOpen program or not. Irrespective of whether the Undo mark is visible or invisible, the program should always return to the the Undo mark and delete the mark (i.e. return NX to state before the program execution) if an error is encountered. Invisible Undo marks should always be deleted.

Visible Undo Mark

A visible Undo mark should be created if user has to be able to undo the operations of an NXOpen program. If the program executes successfully, the visible Undo mark should not be deleted. NX will make visible undo marks available to the user under Edit -→Undo List.

Invisible Undo Mark

If the Undo mark is used for ONLY internal error recovery and the programmer doesn't want to expose it to the user, an invisible Undo mark should be used. Invisible Undo marks should always be deleted even when the program executes successfully.

Usage Considerations

When using UNDO Marks the following should be considered.

1. It is recommended that every custom application should create at least one UNDO Mark which is used for error recovery.
2. Undo marks should be maintained (created and deleted after the logical block) for any block of code that is modifying (creating/deleting/editing) the NX session or part data. This includes helper-objects like builders.
3. Do not overuse UNDO Marks, it takes time and space to save NX states. Only define UNDO Marks for significant or critical blocks of operations. The number of active UNDO marks in NX is limited to 200. Once that limit is reached, older marks are reused.
4. Making an UNDO Mark Visible exposes it to the user in the UNDO menu list. Do not expose UNDO Marks to the user that are only used for internal error recovery.
5. Visible UNDO Marks should only be used if you want the user to be able to undo the changes made by the custom application after the application has completed its function and returned control to NX.
6. Take care to maintain a one to one mapping of UNDO Mark Creation and either deletion or returning to the UNDO Mark.
7. There are special cases where an UNDO mark may be invalid. For instance, if an UNDO mark is created and then the Part is Saved or another Part is Opened, NX will not be able to return to the UNDO mark. Another example is trying to return to an UNDO mark that was created before a sketch was entered. To validate that an UNDO mark is still available use UF_UNDO_ask_mark_exist() for Open C applications or DoesUndoMarkExist() method for common API.

UNDO - Language Specific Details

NX Open for C++
NX Open for .NET
NX Open for Java

NXOpen C++

```
UF_UNDO_mark_id_t myMark;

UF_UNDO_set_mark(UF_UNDO_invisible, myMarkName, myMark);

status = ........

if ( status !=  0 )
{
    /* UNDO to the mark */
    status = UF_UNDO_undo_to_mark( myMark, myMarkName);
}

/* Delete the invisible undo mark */
UF_UNDO_delete_mark( myMark, myMarkName );
```

.NET

```
// Create UNDO Mark
Session.UndoMark myMark;
```

```
myMark = theSession.SetUndoMark(Session.MarkVisibility.Invisible,
markName);
try
{
  ..........
   // Success  Remove the UNDO Mark (because it is invisible )
    theSession.DeleteUndoMark(myMark, markName);
}
catch (NXException exceptionObject)
{
   // Error Reporting and Recovery

      try
    {
       // Return NX to valid state and remove the UNDO Mark
       theSession.UndoToMark(myMark);
    }
    catch
    {
        // UNDO Failed  let user know that the part may be in an
invalid state

    }
    theSession.DeleteUndoMark(myMark, markName);
}
```

Java

```
Session theSession = (Session)SessionFactory.get("Session");

 int myMark;
 myMark=
theSession.setUndoMark(nxopen.Session.MarkVisibility.INVISIBLE,
markName);

try
{
   ..............
   theSession.deleteUndoMark( myMark, markName);
}
catch (Exception e )
{
    try
    {
      //Return NX to valid state. Add the error note to syslog

      theSession.logFile().writeLine("Error: " + e.getMessage() );
      theSession.undoToMark( myMark );
    }
    catch
    {
```

```
            //UNDO failed, report to user
        }

        //UNDO successful, delete the undo mark
        theSession.deleteUndoMark( myMark, markName );
    }
```

Debugging

Debugging NX Open Applications

Debugging NX Open Applications

Debugging NX Open applications with visual studio is no different then debugging any other application.

Debugging with Visual Studio

1. Compile and Link NX Open application using visual studio - See Compiling and Linking
2. Start the NX process from visual studio - Project→Properties→Debug→Start External Program <Path to NX executable>

If you want to debug NX Open application while in Teamcenter Integration mode:

Pass in the command line argument " -pim=yes -u<teamcenter user name> -p=<teamcenter password>. This can be done in Project→Properties→Debug→Command Line Arguments

Debugging Java Applications

1. Compile and link NX Open Java application - See Compiling and Linking
2. To attach the debugger to the JVM:

Before starting NX, set the UGII_CLASSPATH_PRELOAD environment variable so that it contains the classpath for the program you want to debug.

Next, add the following to the UGII_JVM_OPTIONS environment variable:

```
-agentlib:jdwp=transport=dt_shmem,address=jdbconn,server=y,suspend=n
```

You can use dt_socket instead of dt_shmem if it is available for your JRE, as dt_shmem is not available on all platforms. For more information, see Sun Microsystems documentation for jdb.

Finally, attach to the JVM. However, you cannot attach to the JVM until it has been started, and NX does not start a JVM until it runs a Java program. You can run any Java application from within NX to start the JVM. For example, you could run a Hello World application. After you start the JVM, connect to it using the following:

```
jdb -attach jdbconn
```

License Checking

Signing Process

An executable for any application must be "signed" before it can be executed by anyone who does not have an NX Open Author license. This section describes the signing process. This process is typically performed when an application is distributed to the user base and is used to verify that application have been developed using a valid NX Open Author license.

The following two steps define the general signing process.

1. A resource file must be added to the source files. The resource must be compiled and linked with the executable. This step is not required for Java applications.
2. Run the signing utility to add an encrypted string to the executable.

When running without an NX Open Author licenses NX will check for this encrypted string when the application is loaded. If NX does not find an NX Open Author license or signature it will not load the executable, or in Batch mode the Common API will fail to initialize.

- The signing utility may only be executed if an NX Open Author license if available.
- The signing utility also provides a verify option that will display a message confirming whether the file has been correctly signed or not.
- Running the signing utility multiple times on the same executable does no harm, the results are the same as running it once.
- Journals and GRIP programs do not need to be signed. All other NX Open applications must be signed.

Signing Process - Language Specific Details

NX Open for C++

The C++ resource file and signing utility are found in *<NX install directory>\UGOPEN*

Resource File	NXSigningResource.cpp
Signing Utility	nxsign

Note:

NXSigningResource.cpp does not require a C++ compiler. You may need to change the file extension to match the requirements of your compiler.

To *embed* the resource file compile and link it with the executable.

To *sign* an executable run *nxsign* at a command line prompt and provide the name of the executable. For example:

nxsign myApplication.exe

To *verify* that an executable has been signed use the *-verify* option. For example:

nxsign -verify myApplication.exe

Valid file extensions are: dll, so, sl and exe

NX Open for .NET

The .NET resource file is found in *<NX install directory>\UGOPEN*

The .NET signing utility is found in *<NX install directory>\UGII*

Resource File	NXSigningResource.res
Signing Utility	SignLibrary

To *embed* the resource file from a command line use the /resource compiler switch. For example:

VB Example

vbc /libpath:<NX install dir>\UGII\managed /t:library /r:NXOpen.dll /r:NXOpen.Utilities.dll /resource:<NX install dir>\UGOPEN\NXSigningResource.res myApplication.vb.

C# Example

csc /resource:<NX install dir>\UGOPEN\NXSigningResource.res /t:library myApplication.cs

For *Visual Studio* add NXSigningResouce.res as a resource to the project and set the Compile property on resource to: Embedded Resource.

To *sign* an executable run *SignLibrary* at a command line prompt and provide the name of the executable. For example:

SignLibrary myApplication.dll

To *verify* that an executable has been signed use the *-verify* option. For example:

SignLibrary -verify myApplication.dll

Valid file extensions are: dll and .exe

NX Open for Java

The Java signing utility is found in *<NX install directory>\UGOPEN*

Resource File	not required
Signing Utility	SignJar

To sign a java application, a jar file must be used. Class files cannot be signed. This means that class files cannot be distributed to users who do not have NX Open Author licenses.

No resource file is required for Java applications.

To *sign* a jar file run *SignJar* at a command line prompt and provide the name of the jar file. For example:

SignJar myApplication.jar

To *verify* that a jar file has been signed use the *-verify* option. For example:

SignJar myApplication.jar -verify

Valid file extensions are: jar

Feature based license checking

At runtime the Common API uses Feature Based license checking. This means that an application will only run if the NX licenses are available for the set of capabilities that are used by the application. For example, if the application generates geometry then a modeling license is required. If the application generates drafting dimensions then a drafting license is required. The fundamental idea is that an application should require the same set of NX licenses that would be required to produce the same results interactively using NX without the application.

The language reference guides define the feature licenses that are required for each property and method in the Common API. If an applications attempts to access a property or execute a method without the required license a dialog is displayed showing a license error. The name of the missing feature license is logged in the NX log file. The application will then exit.

If NX has previously reserved a license the application will not check out a new license, it will continue to use the license that is active for the NX session. When an application is unloaded (see the section on Unload Options in Development Cycle Considerations) all unused licenses are released.
It is also possible for the application to programmatically reserve and release specific feature license (See License Management).

License Management

The Common API contains a License Manager class that can be used to programmatically reserve and release feature licenses. The License Manager class contains three methods. The following discusses when to use these methods.

Reserve - use the reserve method to checked out a feature license for the application. This will ensure that the license is available when required by the application. To guarantee that an application will run without encountering license errors, it is recommended that an application reserve all required licenses during it's startup process. If a reserve fails the user can be warned before the application produces partial results.

Release - use the release method to inform NX that the application no longer requires the licenses. If the license is not in use by any other application it will be returned to the open pool of

NX licenses on the system. This method can be used to optimize license usage. However, it is up to the programmer to guarantee that the licenses is no longer required by the application.

IsReserved - use this method to check to see if a license is already reserved by the NX session. This method does not check out or check in any license.

Note:

To guarantee that licenses are available it is recommend that you use the Reserve method and then check for failures. Using the IsReserved method followed by calling the Reserve method for licenses that are not reserved will not prevent previously reserved licenses from being released before the application requires the license.

These methods take a text string as input to define the target feature license. The various text strings can be found in the language reference guides. Each property and method provides the specific test string used to reference the license that is required to access the property or execute the method.

License Management - Language Specific Details

NX Open for C++
Open for .NET
Open for Java

NX Open for C++

The syntax to reserve and release the solid modeling license in C++ is:

```
NXOpen::Session *theSession = NXOpen::Session::GetSession();

  theSession->LicenseManager.Reserve("solid_modeling");

    << your application >>

  theSession->LicenseManager.Release("solid_modeling");
```

NX Open for .NET

The syntax to reserve and release the solid modeling license in Visual Basic .NET is:

```
Dim theSession As Session = Session.GetSession()

  theSession.LicenseManager.Reserve("solid_modeling")

    << your application >>

  theSession.LicenseManager.Release("solid_modeling")
```

NX Open for Java

The syntax to reserve and release the solid modeling license in Java is:

```
  Session theSession = (Session)SessionFactory.get("Session");

  theSession.LicenseManager.reserve("solid_modeling");

    << your application >>
```

```
theSession.LicenseManager.release("solid_modeling");
```

Additional Topics

Additional Topics – Java

Additional Topics – Java

This section discusses additional topics related to NX Open for Java.

JVM Parameter

When the Java Virtual Machine (JVM) is run inside NX, it uses the following parameters:

Parameter	Description
UGII_JVM_LIBRARY_DIR	The directory where the jvm shared library is located.
UGII_JVM_OPTIONS	These options are just like the options that you can use on the command line with the JRE's Java executable, except the classpath must be set using UGII_CLASSPATH. Some common options are: –Dproperty=value –Xmsn
UGII_CLASSPATH	The classpath. The NX Open jars do not need to be added here, since they are added automatically. Use this parameter only when running a class file, not a jar file. This method is similar to running a jar file through a java exectuble. You must define a classpath for a jar file in the jar file's manifest file. Requirements of the classpath: • Paths must be relative. • Separate each path listed in the classpath with a space. These requirements are Java-specific and are not imposed by NX. For more information about this topic, see Sun's documentation on Jar files.

NX obtains initial settings for these parameters from ugii_env (ugii_env.dat on Windows). You can edit ugii_env before starting NX to change these settings.

After NX starts, you can change these parameters using the File → Execute →Override Java Variables command. However, you can only change UGII_CLASSPATH after the JVM starts. The fact that you cannot shut down the JVM and then restart it in the same OS process is due to a limitation of Java itself.

Creating GUIs with Java

When running a program from within NX, System.exit terminates NX immediately, without prompting you to save your changes. Do not use System.exit or anything that calls System.exit, unless you want to terminate NX. In particular, in Java Swing, NX shuts down when your GUI is closed if you use JFrame.EXIT_ON_CLOSE in JFrame.setDefaultCloseOperation(). You can use DISPOSE_ON_CLOSE instead. On Windows, debugging a Java GUI that you run from within NX

can be difficult because Windows discards System.err because NX is not a console application. For a solution, see the following information on redirecting System.err.

On Windows, debugging a multi-threaded application that you run from within NX is more difficult than on Unix because the stack trace for uncaught exceptions in threads other than the main thread is written to System.err and Windows discards System.err because NX is not a console application. (This is not an issue on Unix. On Unix, System.err gets printed to the console in which you started NX.) To see the stack trace, you can redirect System.err. For example, to redirect System.err to the syslog, you can use the following code:

```
static class SyslogOutputStream extends OutputStream
    {
        private LogFile m_log;
        private StringWriter m_buf;
        public SyslogOutputStream(LogFile log)
        {
            super();
            m_log = log;
            m_buf = new StringWriter();
        }
        public void write(int c)
        {
            if ( c != '\n' )
                m_buf.write(c);
            else
            {
                try
                {
                    m_log.writeLine(m_buf.toString());
                }
                catch (Exception e)
                {
                    ;
                }
                m_buf.getBuffer().setLength(0);
        }
    }
    public static void main(String[] args) throws Exception
    {
        Session theSession = (Session)SessionFactory.get("Session");
        if (
System.getProperty("os.name").toLowerCase().startsWith("windows") )
        {
            PrintStream newerr = new PrintStream(new
SyslogOutputStream(theSession.logFile()));
            System.setErr(newerr);
        }
    ...
    }
```

To redirect System.err to a scrollable dialog, you can use this class:

```
   // TODO: This class was created for debugging purposes
   // If this class is used for non-debugging purposes,
   // some work should be done on this class to optimize its
performance.
   public class DialogOutputStream extends java.io.OutputStream
   {
       private JTextArea textArea;
       public void write(int c)
       {
           if ( textArea == null )
               buildDialog();

           textArea.append("" + (char)c);
       }
       private void buildDialog()
       {
           JFrame frame = new JFrame("err");
           frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
           textArea = new JTextArea(20, 80);
           JScrollPane pane = new JScrollPane(textArea);
           frame.getContentPane().add(pane, BorderLayout.CENTER);
           frame.pack();
           frame.setVisible(true);
       }
   }
```

Private Classes and Methods

Do not use implementation private classes and methods directly. In short, do not directly use NX Open for Java classes ending in "_impl" or methods starting with an underscore. If you compile your programs using the instructions in this guide, you do not need to worry about accidentally using the "_impl" classes. Why shouldn't I use these classes and methods? The public API of NX Open for Java is remotable interfaces. The classes that implement these interfaces are in separate jars and have names ending in "_impl". Do not use these implementation classes directly; always make calls using the remotable interfaces in the public API instead. Using the "_impl" classes directly 5-2 NX Open for Java Programmer's Guide Contents will make it difficult to convert an application into a remote application. The "_impl" classes are private to the implementation of NX Open and are subject to change at any time without notice. There are a few methods that should only be used by NX Open internally but which need to appear in the public interface due to the remoting architecture. These methods start with an underscore and are marked as deprecated in the API Reference Guide. Do not use these methods. These methods are private to the implementation of NX Open and are subject to change at any time without notice.

Release Upgrades

A primary goal of NX Open is to maintain your automation investments. This is done by adopting policies which minimizes the amount of code changes which are required by you to migrate your applications to new releases of NX. This section discusses these policies and how they impact your ability to support the users of your applications. Also, discussed are the steps you should be taking to successfully move your application to new releases of NX.

NX Open API Change Policy

NX maintains the following three primary policies to protect your investments.

1. API changes should be designed to minimize any changes to your source code. As an example, if the capabilities of a method are expanded which require new parameters then NX may maintain the original method and add a new method which includes the new capabilities. In this way existing applications do not require code changes unless they want to take advantage of the new capabilities.
2. If an API change does require you to make source code changes, if at all possible, you are given a one release warning. For example, if a method is going to be replaced by a new method, the original method will be maintained for at least one release before it is removed. The Release Notes and if possible compile time warnings are used to warn you of changes made in the current release and changes coming in the next release (see the next section on Finding New and Obsolete Methods and Properties).
3. Libraries should be upward compatible for all minor releases. This means that if you compile and link your application with the libraries shipped with a major release of NX (e.g. NX 5.0.0), then your application should continue to run with all future minor releases (e.g. NX 5.0.2) without having to be recompiled and relinked. Which means that you do not need to reship your applications to customers running various minor releases of NX.

Note:

NX development makes every effort to follow the above policies. However, there are time when the policies must be violated due to the type of changes that are required. NX maintains and publishes stability metrics which shows that for at least the last 10 years the stability of the NX Open API is above 96% while the average is above 99%.

Finding New and Obsolete Methods and Properties

NX Open changes and planned changes are published in the following section of the Release Notes.

Release Notes → Caveats and Product Notes → NX Open → Product Notes

This section contains detailed lists of changes organized by the NX Open libraries. For each library deleted methods and properties are listed as well as methods and properties that are marked as obsolete. Obsolete methods and properties are still available in the release but may be deleted in the next release. The differences listed also contain information about other forms of API changes, such as changes to: types, enums, class definitions, fields and method signatures.

In addition to the library difference list published in the release notes, obsolete methods are also marked as deprecated. Which means that you will get a compile warning when these functions are used in your application. This is a warning that these functions are still valid in the current NX release but may be deleted in the next release. In this way you will know which of your source files will need to change to stay current with NX Open and you are given a one release warning.

Note:

Open C API changes are listed in the same NX Open Product Notes section of the Release Notes under: "New Open C routines", "Newly retired functions" and "Deleted Open C routines" (maybe denoted as "Obsolete" instead of "Deleted" in earlier versions of NX). Also, function declarations that are newly retired (obsolete/deprecated) are moved to uf_retiring.h file, which contains a complete list of Open C functions which could be deleted in the next NX release.

Your Expected Release Upgrade Process

For each major release of NX you should perform the following steps to migrate your application to the release.

1. Review the NX Open Product Notes for the release to understand what changes are being made in the release and what changes are planned for the next release.
2. For any methods or properties that have been deleted and for those which you have not already replaced in your code, implement the replacement code.
3. Recompile your entire source base using the current compile time files and setting for the given release.
4. For any methods or properties that produce deprecation warnings, decide if you are going to replace this code now or in the next release. Implement the replacement code for any deprecated methods you want to replace now and recompile.
5. Link your application with the appropriate NX Open libraries.
6. Perform a full suite of application testing in a stable NX environment.
7. Distribute your applications to your user base.

You can find information about compiling and linking with the current release of NX Open in the Compiling and Linking section of this manual.

Wizard Setup

NX Open Visual Basic (VB) wizard

Creating a Visual Basic project using the NX Open VB wizard

The following topic describes how to develop an NX Open Visual Basic application using Microsoft Visual Studio. It is recommended that all NX Open Visual Basic program development on Windows be done from a Visual Basic project

Starting Visual Studio

You can start Visual Studio by typing "devenv.exe" from an NX command prompt window. Studio will pick up the required NX environment variables necessary to compile and link a project.

Creating a Visual Basic project

Use the Visual Basic wizard whenever you need to create a new NX Open automation program in Visual Studio with the Visual Basic language. The wizard automatically adds the required references to NX Open libraries in the new project.

Using the NX Open VB wizard

To use the NX Open VB wizard:

Step 1. Select the **File→New→Project** menu item to activate the **New** dialog box.

Step 2. Under **Project Types**, expand **Other languages** and select **Visual Basic**.

Step 3. Select **NX5_VB** from the list of **Templates**.

Step 4. Enter a project name into the "Project name:" dynamic input box. By default this becomes the name portion of the program being built. For instance, a project named "MyNXOpenApp" produces either a "MyNXOpenApp.dll" or "MyNXOpenApp.exe". You can override this later, if necessary.

Step 5. Click **OK** and follow the on-screen instructions to create your Visual Basic project.

NX Open C# wizard

Creating a C# project using the NX Open C# wizard

The following topic describes how to develop an NX Open C# application using Microsoft Visual Studio. It is recommended that all NX Open C# program development on Windows be done from a Visual C# project

Starting Visual Studio

You can start Visual Studio by typing devenv.exefrom the NX Command Prompt window. Visual Studio uses the required NX environment variables to compile and link a project.

Creating a Visual C# project

Use the C# wizard whenever you need to create a new NX Open automation program in Visual Studio with the C# language. The wizard automatically adds the required references to NX Open libraries in the new project.

Using the NX Open C# wizard

To use the NX Open C# wizard:

Step 1. Choose **File→New→Project** menu item to activate the **New** dialog box.

Step 2. Under **Project Types**, expand **Other languages** and select **Visual C#**.

Step 3. From the **Templates list.** Select **NX5_VCS** .

Step In Project Name, enter a project name.

4. By default this becomes the name portion of the program being built. For example, a project named MyNXOpenApp produces either a MyNXOpenApp.dll or MyNXOpenApp.exe. You can override this later, if necessary.

Step 5. Click **OK** and follow the on-screen instructions to create your C# project.

Visual Studio Application Wizard Setup

Microsoft Visual Studio can be used to compile, link and debug programs on the Windows platform. There are currently three NX Wizards that have been integrated into Visual Studio and available for use with the Common API. They are:

NX Open Wizard for use with C and C++ programs (found under the VC directory)

NX Open VB Wizard for Visual Basic programs (found under the VB directory)

NX Open C# Wizard for C# programs (found under the VC# directory)

If Visual Studio has been installed on a workstation and then NX is installed locally (on the same workstation), the available NX Wizards will be installed automatically. Otherwise the following steps need to be taken to set up each of the NX Wizards on the local workstation.

- Make sure the appropriate version Microsoft Visual Studio has been installed
- Copy all files from the NX kit to the corresponding directories of the local Visual Studio installation. The kit is located in:

  %UGII_BASE_DIR%\ugopen\vs_files\

To determine the correct location of the Visual Studio installation, look at the productdir registry key for each language. The key for C and C++ is HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\version#\Setup\VC/productdir (where version# would be 7.1, 8.0, etc). For other languages replace the VC with either VB or VC#.

- For each wizard, copy any files and/or folders under each subdirectory from the kit to the Visual Studio installation. This example is for the C language so substitute VB or VC# and the appropriate subdirectory for the other languages. Copy:

  %UGII_BASE_DIR%\UGOPEN\vs_files\VC\VCWizards\

  to

  C:/program Files\Microsoft Visual Studio 8\VC\VCWizards\

  and

  %UGII_BASE_DIR%\UGOPEN\vs_files\VC\vcprojects\

  to

  C:/program Files\Microsoft Visual Studio 8\VC\vcprojects\

Repeat for each language.

Appendices

NX Open System Requirements

Supported Platforms

For supported platforms certified to run NX6, see Release Notes

NX Open for .NET

Running NX Open for .NET Executables and DLLs

Running NX Open for .NET executables and .DLLs requires that you have the following installed:

- The Microsoft .NET Framework 2.0, which comes with Visual Studio. You can also download it from the Microsoft .NET Framework Version 2.0 Redistributable Package page.

Creating NX Open for .NET Executables and DLLs

Creating NX Open for .NET executables and .DLLs requires that you have the following installed:

- Microsoft Visual Studio .NET 2005 SP1
- The Microsoft .NET Framework 2.0, which comes with Visual Studio.

Adjusting intranet zone security

By default, the .NET platform is set to a high security configuration. This means that the journal replay mechanism checks the intranet security status before it attempts to execute a journal. By default, this high level security does not allow journals to execute from network-mounted drives. If your security settings are too strict, you may receive this warning:

Insufficient permission to load library. If your installation is on a network drive you may need to adjust .Net security settings. See the Release Notes for more information.

In order for journals to access libraries that are mounted on *internal* network drives, you must reset the default security level. You can accomplish this using the user interface or a command line. The following sections explain each method.

Using the user interface

Note:

This option may not display on systems in which the .NET SDK is not installed. If the option for Microsoft .NET Framework 2.0 Configuration is not available on the control panel, you can still adjust intranet security using a command line.

1. Click **Start** → **Settings** → **Control Panel**.

2. Select **Administrative Tools → Microsoft .NET Framework 2.0 Configuration**. This displays the Microsoft .NET Framework 2.0 Configuration page.
3. Click **Runtime Security Policy**.
4. Select **Adjust Zone Security**. This displays a wizard that asks if you want to configure for the computer or current user. The default is **Make changes to this computer**.
5. Accept the default and click **Next**.
6. Select **Local Intranet** and adjust its trust level to **Full Trust**.
7. Click **Next**.
8. Click **Finish**.
9. Close the Microsoft .NET Framework 2.0 Configuration page.

Using a command line

To adjust intranet zone security using a command line, type the following at a command line prompt and press Enter:

```
caspol -q -m -cg LocalIntranet_Zone FullTrust
```

Note:

You can set the security in two areas. The user interface exposes only one. On some systems, the one exposed in the user interface is the 32-bit security and the 64-bit security is not exposed. If you change the zone security, but cannot play a Visual Basic file due to the **Insufficient** permission, try this command:

```
$SystemRoot\Microsoft.NET\Framework64\v2.0.50727\caspol.EXE -q -m
-cg LocalIntranet_Zone FullTrust
```

NX Open for Java

NX Open for Java is designed to be used with Java version 1.6 or higher. You must use the 64-bit version of Java on Linux and the 64-bit version of Windows.

Runtime Environment

The Java runtime environment distributed and installed with NX does not include the Software Development Kit (SDK) tools for compiling and debugging Java programs. You can obtain these tools for Linux, WNTI32, and WNTX64 platforms from Java SDK.

Terms and Acronyms

API — Application Programming Interface. An NX Open API is defined by a set of functions/methods which provide access to the NX run time data model and user interface. NX Open includes several different APIs (see What is NX Open and Available Toolkits).
Application — This document uses the term "application" to refer to automation solutions that provide general solutions to a given problem domain. An application lets a user create objects based on general inputs from a user. The inputs may be in the form of data files or an interactive user interface. The objects created may be reports, data files, geometric models, NC programs, CAE models or just about anything imaginable. A Journal is used to refer to a single source file

as it is recorded by NX. A Journal may be turned into an application by adding a user interface to generalize the user inputs (see Turning Journals Into Applications).

Common API — A common API is a programming interface that supports multiple language binding where each language supports the same set of classes, methods and properties. The NX architecture now supports a Common API that NX developers are required to define when adding new capabilities. There is then an automated process which produces the NX Open language binding for C++, .NET and Java. In this way all NX Open language bindings have the same set of capabilities defined by a common API. There are many advantages to the common API approach (see Available Toolkits).

.dlg — The file extension used for UI Styler Dialog definition files. See (UI Styler Guide).

.dlx — The file extension used for Block Styler Dialog XML definition files. See (Block Styler Guide).

Entry Point — To execute a program it must first be loaded into memory and then execution control must be passed to the program. An entry point is the function or method within the program that control is first passed to for it's execution. An entry point may also be a standard way of accessing a capability that the application may provide. NX Open defines several different entry points. For instance, the Ask Unload Options entry point (see Unload Options) is a method that may be used to instruct NX when to unload a program when the program returns control to NX. See Entry Points for a summary of all entry points available to NX Open programs.

External Program — An execution mode for an NX Open programs, also called Batch Mode. When an NX Open program is executed in an external mode, a session of NX is running without the NX user interface (see Execution Modes).

IDE — Integrated Development Environment. An IDE contains all of the tools that are required for applications development integrated into a single system. For instance, an IDE will typically contain a code editor, a compiler, a linker, a debugger and access to user documentation. A application developer can code and test their applications all within a single environment. An example IDE is Microsoft's Visual Studio.

Internal — An execution mode for NX Open programs, sometime called Interactive Mode. When an NX Open program is executed in an internal mode, a session of NX is running with the full NX user interface. The user is interacting with NX and using NX to invoke the NX Open program (see Execution Modes).

Journals — A Journal is source code which is generated interactively by NX during the normal execution of an NX session (see Journals).

.men — the file extension used for Menu definition files. See (MenuScript Guide).

NX Data Model — The NX Data Model is defined by all of the information that is maintained by NX during an active session. The data model is used to maintain the current state of all open parts including the assembly relationship, the analysis configuration, the manufacturing information and many other product definition data. The data model in the active session also maintains the current state of the user interface options. NX Open provides access to a significant portion of the NX Data Model.

NX Part — An NX Part is all of the information that is stored in a single part file and maintained during an active NX session as a loaded part.

Object Model — An object model is defined by the relationship between the classes for a given systems library and more importantly, by the relationships between the objects within the classes. The object relationships define how the methods and properties of those objects are used to

create a useful application. A common API is defined by multiple language bindings that all share a common Object Model.

.prt — The file extension used for NX Part files.

Part Corruption — A corrupt NX Part is typically a part which cannot be loaded by NX, which means the data in the part file is lost. Coding errors in an NX Open application may produce corrupt parts. NX attempts to detect and provide warning when a corrupt part is saved. However, there are some types of errors which cannot be detected until NX attempts to load a part. To ensure that corrupt parts are not produced, NX Open applications must take steps to detect, report and correct unexpected behaviors (See Error Handling)

Programs — Any set of compute instruction that are executed by a compute. This document discusses Journals and Applications, which are both compute programs.

Session Corruption — A corrupt session is an NX session which is in an unexpected state. A corrupt session may product unexpected results including: corrupted parts, a hung session (does not respond) or a session exit. Coding errors in an NX Open application may result in a corrupted NX session. To ensure that corrupt sessions are not produced, NX Open applications must take steps to detect, report and correct unexpected behaviors (See Error Handling).

Signature — A signature is the formal set of parameters that are used to define the inputs and outputs of a function or method. An entry point is defined by the function or method's name and it's signature. It is possible to define different signature for the same method. The signature used by the calling program is used to find the entry point which is defined by the matching signature.

.tbr — The file extension used for Tool Bar definition files. See (MenuScript Guide).

Unload Options — Options set by an NX Open program which tell NX when to unload the program after the program exits (see Unload Options)

Example Programs

There are various example programs supplied with NX. The programs are in UGII_BASE_DIR\ugopen\SampleNXOpenApplications

There is a folder for each language for e.g. .NET examples are in

UGII_BASE_DIR\ugopen\SampleNXOpenApplications\.NET, C++ examples are in

UGII_BASE_DIR\ugopen\SampleNXOpenApplications\C++ and Java examples are in

UGII_BASE_DIR\ugopen\SampleNXOpenApplications\Java

Most examples have a Readme.txt which explains what the example is supposed to do and how to use it.

Entry Points

The following table shows entry points for various user exits in all supported languages. For details on how to use the entry points, see User Exits

NX Event	User Exit Environment Variable	C/C++ Entry Point	VB Entry Point	Java Entry Point
Execute Custom Program (initialization)	N/A	ufusr	Main	main
Execute Custom Program (after init)	N/A	ufusr_ask_unload	GetUnloadOption	getUnloadOption
Unload Program	N/A	ufusr_clean_up	UnloadLibrary	onUnload
Open Part	USER_RETRIEVE	ufget	ufget	ufget
New Part	USER_CREATE	ufcre	ufcre	ufcre
Save Part	USER_FILE	ufput	ufput	ufput
Save Part As	USER_SAVEAS	ufsvas	ufsvas	ufsvas
Import Part	USER_MERGE	ufmrg	ufmrg	ufmrg
Execute GRIP Program	USER_GRIP	ufgrp	ufgrp	ufgrp
Add Existing Part	USER_RCOMP	ufrcp	ufrcp	ufrcp
Export Part	USER_FCOMP	uffcp	uffcp	uffcp
Component Where-used	USER_WHERE_USED	ufusd	ufusd	ufusd
Plot File	USER_PLOT	ufplt	ufplt	ufplt
2D Analysis Using Curves	USER_AREAPROPCRV	uf2da	uf2da	uf2da
User Defined Symbols	USER_UDSYMBOL	ufuds	ufuds	ufuds
Open CLSF	USER_CLS_OPEN	ufclso	ufclso	ufclso
Save CLSF	USER_CLS_SAVE	ufclss	ufclss	ufclss
Rename CLSF	USER_CLS_RENAME	ufclsr	ufclsr	ufclsr
Generate CLF	USER_CL_GEN	ufclg	ufclg	ufclg
Postprocess CLSF	USER_POST	ufpost	ufpost	ufpost
Create Component	USER_CCOMP	ufccp	ufccp	ufccp
Change Displayed Part	USER_CDISP	ufcdp	ufcdp	ufcdp
Change Work Part	USER_CWORK	ufcwp	ufcwp	ufcwp

Remove Component	USER_DCOMP	ufdcp	ufdcp	ufdcp
Reposition Component	USER_MCOMP	ufmcp	ufmcp	ufmcp
Substitute Component Out	USER_SCOMP1	ufscpo	ufscpo	ufscpo
Substitute Component In	USER_SCOMP2	ufscpi	ufscpi	ufscpi
Open Spreadsheet	USER_SPRD_OPN	ufspop	ufspop	ufspop
Close Spreadsheet	USER_SPRD_CLO	ufspcl	ufspcl	ufspcl
Update Spreadsheet	USER_SPRD_UPD	ufspup	ufspup	ufspup
Finish Updating Spreadsheet	USER_SPRD_UPF	ufspuf	ufspuf	ufspuf
Replace Reference Set	USER_RRSET	ufrrs	ufrrs	ufrrs
Rename Component	USER_NCOMP	ufncp	ufncp	ufncp
NX Startup	USER_STARTUP	ufsta	Startup	startup
Access Genius Library Management System	USER_GENIUS	ufgen	ufgen	ufgen
Execute Debug GRIP	USER_GRIPDEBUG	ufgrpd	ufgrpd	ufgrpd
Execute User Function	USER_UFUNC	ufufun	ufufun	ufufun
Initialize new operation	USER_CREATE_OPER	ufnopr	ufnopr	ufnopr
CAM Startup	USER_CAM_STARTUP	ufcams	ufcams	ufcams

Signatures

C/C++	**VB.NET**	**Java**
void xyz(char *s, int *retcode, int rlen)	Function xyz(s As String()) As Integer	int xyz (String[])