Team Developer Migratory Path

Unify Corporation

Table of Contents

Abstract

Unify is aware there are a large number of 16-bit SQLWindows applications waiting to be converted to 32-bit Team Developer. Microsoft® no longer supports 16-bit operating systems and has stopped supporting 16-bit sub systems in 32-bit operating systems such as Windows® 2000/XP. Now is the time for all 16-bit applications to be migrated.

Migration difficulty depends on the features you have in the current application, some of which migrate easily and some not so easily. This paper will discuss the issues and features you need to consider while migrating 16-bit applications to 32bits

This paper uses SQLWindows to refer to 16-bit code and Team Developer to 32-bit code.

Migrating from SQLWindows to Team Developer

Win32 Architectural Changes

There are many changes Microsoft has done to the operating system, but we will discuss mainly the areas that affect SQLWindows applications.

Many data types that were 16-bit are now 32-bit (i.e. Window Handle, wParam, int, and BOOL). Elements of structure are now also 32-bit and long remain 32-bit. Shorts and WORDs are still 16-bit and abyte remains 8-bit. The 32-bit widening has affected various structure parameters. For example, in16-bit Number: WORD is commonly used for 16-bit unsigned integers and handles. In Team Developer this needs to be changed to Number: DWORD for API functions that use UNITs. Functions that return a BOOL rather than WORD also need to use DWORD.

As far as Win32 messages are concerned, for some messages wParam and lParam interpretations have changed. You need to refer to MSDN for more information on the specifics.

As for the direct Windows API calls; there are some changes made to the names of the DLLs and functions in Win32. Win32 provides ANSI and UNICODE character set support. For Team Developer applications that need to call API functions, call the ANSI version of the function. More details regarding external functions migration are provided in the Migration Issues section of this paper.

Other important changes that are to be considered are Asynchronous input sharing memory and communication support.

Asynchronous Input: Each thread can have its own message queue, andan API call like GetActiveWindow() can return NULL if the active window is in a different thread.

Sharing memory: Sharing memory is difficult in Win32, because each application lives in its own process. Memory space allocated to one process cannot be shared with another process. We recommend using memory mapped files or database to share data between applications.

Having each application run in its own memory space reduces program crashes. In the 16-bit world, all applications ran in the same shared memory space and if one application crashed, so did all the other applications.

Communication Support: OpenComm, CloseComm, ReadComm and WriteComm are now obsolete. Instead, you must use CreateFile, CloseHandle, ReadFile, WriteFile, etc. with COM1. Also WM_COMMNOTIFY is obsolete and you must use the WaitCommEvent API function.

Limitations removed: There is good news for people migrating to 32-bit Team Developer. SQLWindows was designed for 16-bit operating systems -even though it runs on 32-bit operating systems nowadays and has the same limitations as 16-bit operating systems. For example, SQLWindows creates a symbol table at compile stage that collects together all of the symbol information within an application. The maximum size of this symbol table is 64K and often very large applications run into this limitation. With the invention of 32-bit operating systems, this limitation is removed in Team Developer. Not only is the symbol table 64K limitation removed, but also the 64K limit of outline segment size. Some controls, like table window and list box, have limitations in the16-bit world and these limitations are removed in 32-bit.

Migration Issues

Many SQLWindows applications written using SAL code only will require nothing more than opening them up in Team Developer and saving them to run in the 32-bit environment. However you need to be aware of the following issues.

First of all before opening the application in Team Developer, we recommend that all the applications, including apls, be saved as text files (apt files).

Also, attention should be paid to the following items before you open the application in Team Developer:

• Following constants no longer exist and if you need them - you need to define them manually as global constants.
o For example,
• TYPE_QuestTableWindow = 0x00400000
• TYPE_QuestChildTableWindow = 0x00800000

Quest is not supported in Team Developer. Therefore we recommend converting all Quest Windows to table windows in SQLWindows. Remember to program manually

any functionality the Quest window may have provided. Once this is completed you can convert to Team Developer by opening the apt file.

Named transactions are not supported and the functions SqlConnectTransaction, SqlSharedSet, SqlSharedAcquire and SqlSharedRelease are obsolete because they rely on the shared memory concept of 16-bit operating systems. With 32-bit applications running in separate process/memory space, same technique could not be applied. In 16-bit applications, named transaction was used within a single application or to share Sql handles across multiple applications. There is an alternate option that you can consider in place of named transactions within single application. New session functions introduced in 32-bit can be used to achieve the same. Functions are SqlCreateSession, SqlCommitSession, SqlCreateStatement, SqlFreeSession and SqlGetSessionHandle.

If your SQLWindows applications include any of the items listed below, migrating will require more than opening and saving the applications in Team Developer.

External functions and Direct calls to SDK functions
VBX
OLE1
Select From option in File Include

External functions and Direct calls to SDK functions:
With external functions declarations, all the 16-bit DLLs have to be replaced with their 32-bit equivalent. If the DLL was purchased from a third party, contact the vendor and ask for a 32-bit equivalent. If it is your DLL and you have the source code, then recompile the source in Visual C++ and create a new 32-bit DLL. You could also read Porting information from MSDN. Microsoft also ships a Port Tool with SDK.

If the DLL is a Microsoft supplied DLL replace it with 32-bit equivalent. Commonly used DLLs and their equivalents are as follows:

User.exe >>> user32.dll
Gdi.exe >>> gdi32.dll
Krnl386.exe >>> kernel32.dll

SDK Function names also may have changed. Please check Microsoft's documentation for any name changes and make sure to call the ANSI version of the functions.

Most of the 16-bit applications I have encountered used ordinal numbers specified for the external functions. Please note that the ordinal numbers are different in the 32-bit world. We ship a very handy tool called mapdll.exe with SQLWindows that gives you the list of functions available in any DLL along with their ordinal numbers. This is not available in 32-bit and you must use tools like DUMPBIN/EXPORTS from MS Visual C++ if you wish to use ordinal numbers.

Alternately, you can remove ordinal numbers all together and just use the function names. This allows for easier migration in the future if the ordinal numbers change again.

VBX:
VBX controls are 16-bit and will not work in 32-bit. These controls should be replaced with the 32-bit equivalent such as OCX or ActiveX.

OLE1:
OLE1 has been deprecated and is not part of Team Developer. OLE must be migrated to ActiveX containers.

Select From option in File Include:
In 16-bit SQLWindows, the Libraries section has two options to choose from; File Include and Select From. In Team Developer, the Select From option has been removed. This may lead to problems during migration, especially if the include files have duplicate names. In order to overcome this issue, you must separate the items included using Select From into a new apl and include the new apl using File Include.

Migration Strategies

Parallel Development
Migration can be done gradually while maintaining 16-bit applications. SQLWindows and Team Developer can co-exist happily on the same machine, but you should avoid installing them in the same directory. Try to keep only one sql.ini to avoid database connection errors. You can combine both 16-bit and 32-bit sql.ini entries in one sql.ini. Please note SQLWindows files can be opened in Team Developer but not the other way around.

Repository
If you are using Team Windows for project management you must use two repositories, but both repositories can be in the same database as the table owner and names have changed.

Concurrent Deployment
SQLWindows and Team Developer applications can be deployed concurrently on the same machine. You need to follow the same rules as defined above for parallel development. Remember to have only one sql.ini file and deploy them in separate directories.

Migration Process

Objective:
This will be an overview on how to do a simple 16-bit to 32-bit conversion; some tweaking of the source code might be needed to get most applications to work. We will first look at the steps at doing the initial conversion process. Then at some of the problems areas - like deprecated functions, Windows API function calls, and then some issues with migrating 16-bit Reports (QRP's).

SAL only Code:
SQLWindows applications that are using only SAL code and did not have direct calls to the WinAPI or included external DLLs, will be fairly easy. In such cases, all you will need to do is convert the application with the Conversion Process (listing in this document) and it should compile without any problems. (Notes: wParam is now 32-bit & SQL routers are also 32-bit)

Conversion Process:

- Copy the original source into a conversion directory.
- Establish a build environment for the 16-bit applications, ensuring you can build and run them as a 16-bit application. If the application does not run in the build environment (16-bit), it will not run after converting it to 32-bit.
- Convert the Outlines to a Team Developer readable format. This can be done in batch mode with CBCVT426.Exe or from with-in Team Developer. Note: if you have a large number of apl's or app's, it is recommended that you use a batch process with cbcvt426. To do a single file from the command line prompt, it would be as follows: cbcvt426 programname.app. Or to do this in a batch file, run a display a list of applications into a file (i.e. "dir *.ap? /b >convert.bat"), then edit this file putting the CBCVT426 in front of each application. Then run this batch file from the command line prompt.
- Check for deprecated functions, making a note of what files have these functions. Each of these files will need to be edited with Team Developer to make the appropriate changes. OLE 1.0 and VBX (16-bit functions) are not available in the 32-bit environment (removed by Microsoft). Here is a list of deprecated functions:
    - OLE - SalEditCanInsertObject, SalEditPasteSpecial, SalEditInsertObject, SalOLEAnyLinked, SalOLEFileInsert, SalOLEGetVerbs, SalOLEServerInsert, SalEditCanPasteLink, SalEditCanPasteSpecial, SalEditCanPasteSpecial, SalEditPasteLink, SalOLEAnyActive, SalOLEDoVerb, SalOLEGetServers, SalOLELinkProperties, SalOLEUpdateActive
    - VBX – SalVBXAddItem, SalVBXGetError, SalVBXGetProp, SalVBXOpenProp, SalVBXRemoveItem, SalVBXSetProp, SalVBXCloseChannel, SalVBXGetNumParam, SalVBXGetStringParam, SalVBXRefresh, SalVBXSetPicture
    - Shared SQL Handles – SqlSharedAcquire, SqlSharedRelease, SqlSSTStatus, SqlSharedSet, SqlSSTGetID

- QuestWindows – 16-bit QuestWindows should be converted to Tables Windows. Resulting in the following changes/removals; removed SalOutlineDataTypeOfQuestCol, SalQuestColumnGetFieldName, SalQuestTblApplyEdits, SalQuestTblDiscardEdits, SalQuestTblPopulate, and SalQuestTblReset functions, rename from SAM_QuestDoDetails to SAM_TBLDoDetails, removed TYPE_QuestTableWindow & TYPE_QuestChildTables

- Compile the application: At this point we should be able to compile the applications in Team Developer. Some minor tweaking might be needed to get it working, but it should work correctly at this point.

Migrating from Team Developer 1.5.x to 3.1 or above

This section is intended to assist programmers who are faced with the challenge of migrating Team Developer 1.5 applications to TD Developer 3.1 or above. For a complete list of changes since Team Developer 1.5, please refer to your Team Developer release notes.

Converting OLE2 objects to ActiveX within Team Developer 3.1 or above does not work

For a workaround on how to convert OLE2 objects to ActiveX, please refer to the section “Converting data in OLE2 format to ActiveX format”.

External libraries will be named differently than they were named in Team Developer 1.5

For example, the tooltip library, which was previously cbtti15.dll, is now cbtti31.dll. Install Team Developer 3.1 in a separate directory (e.g. C:\Program Files\UNIFY 3.1) and make a copy of all your sources in a new directory structure (e.g. C:\Inventory Programs 3.1 …) so as not to overwrite your current code. After opening one of the applications, go to Tools | Preferences | Directories menu item, specify the correct path where the Team Developer 3.1 library files can be found, and change the library names in the application. Avoid opening Team Developer applications by double-clicking on the .APP file. Instead always start Team Developer first and then use the File | Open menu item or the toolbar to open the application.

Non-editable datafields in Team Developer 3.1 will receive focus
This is a feature that has become a Windows standard - not a bug. The disabled datafield will still be protected from any input. However, by allowing the focus to enter these fields, the data can be copied from the disabled datafield and then pasted somewhere else. If the non-standard behavior is required, a functional class could be called for each window’s Sam_CreateComplete to loop through the child datafields

and multiline text fields to disable them if they are found to be non-editable. Here's an example:

```
Set hWndChild = p_hWndParent
Set hWndChild = SalGetFirstChild( hWndChild, TYPE_DataField
TYPE_MultilineText)
Loop
If hWndChild = hWndNULL
Break
Set nStyle = GetWindowLongA( hWndChild, GWL_STYLE )
!
Set nType = SalGetType ( hWndChild )
If ( nType = TYPE_DataField ) OR ( nType = TYPE_MultilineText )
If (nStyle & ES_READONLY)
Call SalEnableWindow( hWndChild )
Call SalDisableWindow( hWndChild )
!
Set hWndChild = SalGetNextChild( hWndChild, TYPE_DataField
TYPE_MultilineText )
```

Other Issues

Assuming that the Team Developer 3.1 migration also includes moving to Windows 2000 / NT and new COM+ components, any remaining migration issues are likely to be functionality "outside" of SAL, such as: custom controls and changed object libraries.

For example, if the Team Developer 1.5 application was developed using the Calendar Control 8.0 and now you are using the newer version Calendar Control 9.0, you will experience some compile errors.

In the Calendar control, the first problem that needs to be resolved is "Undefined symbol …….. " ; Create an instance of the new AX_MSACAL_Calendar control. This should be done by using the Controls tool in layout mode. Click on the ActiveX icon, select the Calendar Control 9.0, and drag and drop it on the form. You will want to keep the old control MSACAL_Calendar until you have applied all the code in the Message Actions to the new control. Apply the message actions to the new control (don't cut and paste, use the Coding Assistant). In Team Developer 1.5 and the initial release of Team Developer 2.0, a problem was identified when accessing the ActiveX control in the outline. Do not comment-out or delete the old control directly in the outline; instead, when the old control is no longer needed, delete it in layout mode. Now recompile and you will notice that the "Unresolved symbol" problem has now been fixed. The remaining compile errors will identify the need to resolve "Undefined functions …". Take a look at the Calendar Control 9.0 Class Definitions and you will see that the object AX_MSACAL_Calendar is derived from a COM Proxy class, which is derived from a Functional Class (interface).

All access to the object should be through the interface (Functional Class: MSACAL_ICalendar). If we search the interface we do not find the functions called

SetValue or GetValue; instead we find functions called PropGetValue and PropSetValue. The translation is not always that obvious; however, in this case it was easy to determine that we must use PropSetValue and PropGetValue. Notice that the functions now use Variants as parameters in order to make them more flexible (anything can go in a Variant). However, we must set them correctly using date.SetDate, string.SetString etc.

The following code is an example of using PropSetValue and PropGetValue:

```
Dialog Box: dlgCalendar
Description:
Tool Bar
Contents
Pushbutton: pbOk
Message Actions
On SAM_Click
Call axCalendar.PropGetValue( vDate )
Call vDate.GetDate( dtTmp )
Set p_dDate = dtTmp
Call SalEndDialog( hWndForm, TRUE )

AX_MSACAL_Calendar: axCalendar
Message Actions

On DblClick
Parameters
Actions
Call SalSendMsg(pbOk, SAM_Click, 0, 0)
Functions
Window Parameters
Window Variables
Message Actions
On SAM_CreateComplete
If p_dDate = DATETIME_Null
Set p_dDate = SalDateCurrent( )
Call vDate.SetDate( p_dDate )
Call axCalendar.PropSetValue( vDate )
```

If the Team Developer 1.5 application was developed using Microsoft's Office 97 Object libraries or later versions (e.g. Microsoft Word Object Library 8.0) and now you are using Office 2000 (Microsoft Word Object Library 9.0), you will notice the impact resulting from Microsoft's implementation of COM+. To resolve this, start by re-generating the library. Use the Tools | ActiveX Explorer then select the library (e.g. Microsoft Word 9.0 Object Library), select the required classes and Generate using the Generate Deep option.

Note: When re-generating an object library, it is a good practice to remove the included library from the application and delete the "Centura" APL (e.g. Microsoft Word 9.0 Object Library.APL), from the "Centura" AXLibs directory, before re-generating from the Type Library, this will ensure that none of the code from the previous library will remain in the application.

ActiveX controls have changed. The objects now consist of COM Proxy Classes and the standard COM functions. To create the object you now need to access the Create or CreateEx functions of the COM Proxy Class.

Code example:

```
Set bReturn = oWord.Create( )
If bReturn
Call oWord.PropSetVisible( TRUE )
Else
Set bReturn = FALSE
Call SalMessageBox( "Create Word Object failed", 'Error', MB_Ok )
```

The previous method of using Detach to destroy the object is now replaced with the Release function.

Code example:

```
Call oWord.Release( )
```

As was shown with the Calendar Control, all interactions with the object should be done by accessing the functions exposed by the interface (Functional Class). To get information as to how to call these functions refer to the documentation provided by the vendor of the object library.

Converting data in OLE2 format to ActiveX format

Converting objects from OLE2 to ActiveX does not work

QuickOLE functions SWinSetObjectBits and SwinGetObjectBits are not supported in Team Developer 3.1, and data stored in OLE2 format cannot be loaded into ActiveX containers.

Solution:

Team Developer 1.5.01 PTF6 contains changes to allow conversion of OLE2 data to ActiveX format. These changes consist of a change to function SwinSetObjectBits and the addition of function SalActiveXPaste.

SwinSetObjectBits is modified so that when it loads the contents of a string that was previously created using SwinGetObjectBits into a QuickOLE2 container, it also loads the contents of this string into the clipboard. Function SalActiveXPaste loads the contents of the clipboard into an ActiveX container.

The process of converting OLE2 data to ActiveX format must be carried out under Team Developer 1.5.01 PTF6 and is as follows:

The application in which the conversion is to be carried out must have both a QuickOLE20 class container and an ActiveX container.

Read data stored in files in OLE2 format using Sal File functions into a string.
Use QuickOLE function SWinSetObjectBits to load the data from the string into the OLE2 container. This also copies the data into the Windows clipboard.
Call function SalActiveXPaste to paste the contents of the clipboard into an ActiveX container. This function is described below and must be used in combination with the SwinSetObjectBits function call.

Call function SalActiveXGetData to copy the contents of the ActiveX container to a string. This string may then be saved to a file using Sal File functions.

SalActiveXPaste function description

BOk = SalActiveXPaste(hwndCntrCtrl)

Pastes the contents of the Windows clipboard into an ActiveX container.

Parameters
HwndCntrlCtrl Window Handle. The ActiveX container control

Return value
BOk is TRUE if the function succeeds and FALSE if it fails

Example
Long String: lstrObject
CQuickOle20: pic1
ActiveX: ax1

Call SWinSetObjectBits(pic1, lstrObjet) ! loads data into OLE2 container and Windows clipboard
bOk = SalActiveXPaste(ax1) ! copies data from Windows clipboard into ActiveX container

Notes:
SalActiveXPaste () must be used in combination with SwinSetObjectBits ()

SalActiveXPaste () is present in Team Developer 1.5.01 PTF6 **only**

Sample application OLE2_ActiveX_conversion.apt illustrates this process. This application is included, in a zip file, with the development download of PTF 6 for Team Developer 1.5.1.

Problems Related to Migrating ActiveX apps

Description:
SalActiveXGetActiveObject() returns TRUE intermittently even when Excel is not running..
Solution:
The following steps will solve the problem

Re-Create ActiveX UDV variables in Team Developer 3.1
SalActiveXGetActiveObject()/CreateObject against the interface class (example-Excel__Application) rather than coclass (e.g. Excel_Appplication)

Here is a code sample that WORKS all the time using interface class
If Not SalActiveXGetActiveObject(ExcelApp,'Excel Application')
Call ExcelApp Create Object('Excel Application')
Call ExcelApp.PropSetVisible(TRUE)
Where ExcelApp is defined as Excel__Application: ExcelApp

Here is a code sample that DOES NOT WORK all the time using interface class.

If Not SalActiveXGetActiveObject(ExcelApp,'Excel Application')
Call ExcelApp Create Object('Excel Application')
Call ExcelApp.PropSetVisible(TRUE)
Where ExcelApp is defined as Excel_Application: ExcelApp

Migration Issues with COM and ActiveX

Team Developer 3.1 has made improvements in the handling of COM objects. A new class called the COM Proxy Class has facilitated this. There are some issues this creates when migrating your application from 1.5 to 3.1; there are fewer issues if the original application was using the ActiveX versions of the COM servers. This paper points out issues that you may encounter. Not all topics covered are necessarily applicable to every application.

AUTOMATION.APL Object Class
The Class Object contains the following changes that affect migration:
Create method: In 3.1 the Object.Create has changed to CreateObject. The parameter signature is the same and the effect is the same. Calls to Create with this signature need to be changed to CreateObject. There is also an Object.CreateObjectEx that is new to 3.1; it takes an additional host parameter.

GetInterface has been removed from the Object class, GetInterface is now a function of the COM Proxy Class that is new to Team Developer 3.1. The parameter signature has changed as well.

ActiveX Library Generation (1.5)
Key points: a COM Server's implementation of a Createable "Object" is called a CoClass. CoClasses implement or contain "Interfaces." In SAL Interfaces are represented as a Functional Class derived from the Automation.apl class called "Object." Createable Objects (CoClasses) are represented as either Functional Classes derived from Interface Classes (non-visual) or ActiveX classes derived from Interface Classes (visual.)
"Createable Objects" expose a method called "Init" which calls the inherited method Create.

ActiveX Library Generation (3.1)
Key points: a COM Server's implementation of a Createable "Object" is called a CoClass. CoClasses implement or contain "Interfaces." In SAL Interfaces are represented as a Functional Class derived from the Automation.apl class called "Object." Createable Objects (CoClasses) are represented as either COM Proxy Classes derived from Interface Classes (non-visual) or ActiveX classes derived from Interface Classes (visual.)

ActiveX Library Generation (Key Differences)

- In Team Developer 1.5, the maximum "name" length of anything was 32 Characters. This has been removed in 3.1.
- In Team Developer 1.5, the function Init is exposed for "Creatable" objects. In 3.1, for "Creatable" objects a COM Proxy Class exposes "Create" and "CreateEx."
- In Team Developer 1.5, the function "Detach" is exposed for releasing a reference. In 3.1, COM Proxy Classes expose a method called "Release." Interface Classes expose Detach or can be assigned to OBJ_Null to release their reference.
- There is a subtle difference in the naming convention of Enumerations between 1.5 and 3.1.
- There is a subtle difference in the naming convention of Visual ActiveX Objects between 1.5 and 3.1.
- In the Outline the COM Proxy Class is a different class type than a Functional Class. Likewise an Instance of a COM Proxy is a different outline item type than an instance of a functional class.
- The COM Proxy exposes a GetInterface function with a different signature in 3.1 than in 1.5.

Steps for Migration

All Team Developer 1.5 APLs in the AXLIBS subfolder need to be removed
Note: you may want to back up your Team Developer 1.5 AXLIBS APLs before regenerating those APLs in version 3.1. To see whether you need to do this, read the section below named "Enumerations may need to have their names changed."

If you are not getting an error message when you first open your existing application in Team Developer 3.1, you need to make sure that you already generated the APL in

your new environment. Make sure you are not picking up the old library. If the libraries were merged under Team Developer 1.5 you will need to remove the classes manually.

AXLIBS APLs need to be "regenerated" with the Team Developer 3.1 ActiveX Explorer.
Clarification: Go to Menu item Tools, ActiveX Explorer to generate the APLs for the desired object. The default location for ActiveX libraries is the AXLIBS subfolder of the Team Developer program folder. If you are attempting to generate a library for an object and are having trouble getting a successful generation you may need to use the Browse option in ActiveX Explorer and find the appropriate .TLB or .OLB file in the install folder of the object for which you are trying to generate the library.

Example: If you are attempting to build the Excel library and it has not been registered in Team Developer's ActiveX Explorer, click the Browse option, navigate to the MS Office install folder and select EXCEL9.OLB, then generate the library.
Some class definitions need to be changed from Functional Classes to COM Proxy Classes. This applies to all classes in 1.5 that were Functional Classes and were derived from ActiveX Functional Classes, and now the equivalent ActiveX parent class in 3.1 is a COM Proxy Class.

Example: I have a Functional Class, CSCWord_Application that is derived from Word_Application. In Team Developer 1.5x Word_Application was a functional class. However in Team Developer 3.1 Word_Application is a COM Proxy Class. Therefore I need to declare CSCWord_Application a COM Proxy Class.

Clarification: Word_Application in Team Developer 1.5 is the generated Functional Class that will in Team Developer 3.1 become a COM Proxy Class.
Word_Application is derived from Word__Application, which is derived from Object. The class hierarchy remains the same for both Team Developer versions and reflects the hierarchy of the ActiveX object. The difference being, in Team Developer 1.5 it will be generated as a Functional Class - in Team Developer 3.1 it will be a COM Proxy Class.

All instances of Functional Classes that should now be instances of COM Proxy Classes need to be removed and reentered (this causes the correct Outline Item Type to be used)
Example: I have a form that needs to invoke MS Word. On that form I have a variable declared as CSCWord_Application: oWordApp. Since the original application outline was created using Team Developer 1.5, the outline has stored the appropriate information for a Functional Class. However, in Team Developer 3.1, it is no longer a Functional Class. The oWordApp variable is now a COM Proxy Class and it really needs COM Proxy Class information. To force the changes to the outline, simply enter the exact same declaration as the original variable and delete the old one.

Important note: *simply commenting and un-commenting the item will not make the necessary outline changes.*

All calls made to the Init function of a COM Proxy object need to be changed to Create
Example: Call oWordApp.Init() becomes Call oWordApp.Create()

Clarification: When the new outline is generated the class function Init() no longer exists, it has been deprecated. This function used to be generated for the functional classes that now are represented by COM Proxy Classes. COM Proxy Classes have four new functions generated; they are CreateEx(), Create(), GetInterface(), and Release(). The Init() function is replaced by the Create() function in the COM Proxy Class. It has the same parameter list. It is also recommended that the Release() function be called when you are finished with the Object. In your old code this was probably done with a Call oWordApp.Detach(). The
CreateEx() call is to create the object on another machine and takes the host name as a parameter (DCOM).

All Calls made to the Detach() function of a COM Proxy object need to be changed to Release()
Clarification: In Team Developer 1.5 when you were finished with the COM object you called Object.Detach(), in Team Developer 3.1 you need to call the COM Proxy Class.Release( ). The Detach() function is still valid for interface classes and essentially is the same as setting an interface object equal to OBJ_Null.

Example: In Team Developer 1.5 after doing an oWordApp.Init( ) to create the object after you had completed using the object you would call oWordApp.Detach( ), this needs to be changed to oWordApp.Release( ). If you have called oWordApp.GetInterface(iDocs, Word_Documents) a call to iDocs.Detach( ) or Set iDocs = OBJ_Null will accomplish the same thing.

Any Object name that is now longer than 32 Characters needs to be accounted for.
Clarification: If an Object was registered in Team Developer 1.5 with a name over 32 characters long, the name was truncated to 32 characters. Iin Team Developer 3.1 it will not be truncated; it will have the same name that it was registered with.

Enumerations may need to have their names changed.
Clarification: In 1.5, it was possible for multiple ActiveX object libraries to expose multiple enumerations with the same name, but different values. That's because in 1.5, defined constants have the same 32-character issue as object names, with the additional challenge of a prefix being added. In Team Developer 3.1 this problem was fixed. Thus Team Developer 3.1 will produce different names for ActiveX APL constants than the names that were produced under Team Developer 1.5. This will affect your existing application written under Team Developer 1.5 if that application used references to the constant names within the application itself (not the APL). If

you regenerate the APL using ActiveX Explorer in Team Developer 3.1, those references will now point to names that don't exist in the new APL.

There are two ways to handle this. One way is to open the old APL generated under Team Developer 1.5, and cut and paste the portion of that APL that covered the constant definitions for enumerations. In essence you would now have the same constant defined twice in your application: once with the new name, from the new APL generated by version 3.1 of the ActiveX Explorer, and once with the old (1.5) name from the code that you pasted into your .APP file from the old 1.5 APL.

The second way is to correct the references by searching for the portion of the constant name that does not include the prefix. However, because of the 32-character limitation in Team Developer 1.5, you may not wish to search on the entire portion. See the example below to understand how the trailing characters of a constant name may be truncated in Team Developer 1.5. In such a case, you would want to search on only the truncated string.

Example: Enumeration constants are generated with a prefix. In Team Developer 1.5 this was necessarily short because of the 32-character limit for names. For example, wd was the prefix for Word, mso was the prefix for MS Office etc. On the other hand, Team Developer 3.1 gives the full origin of the enumeration. Word_WdWordDialogTab_wd is the Microsoft prefix for a Word dialog constant. Thus a constant defined in Team Developer 1.5 as wdDialogToolsAutoCorrectEx-ceptio = 1400000 becomes Word_WdWordDialogTab_wdDialogToolsAutoCorrectExceptionsTabFirstLetter = 1400000 in Team Developer 3.1.

All ActiveX Window Objects need to have their name changed appropriately
Clarification: The new naming convention for the visual ActiveX Window Objects has AX_ pre-pended to the class name.

Example: In Team Developer 1.5, the ActiveX class for a Microsoft Word object is Word_Document. In Team Developer 3.1 that same class is generated as AX_ Word_Document.

ActiveX Differences
When developers start to move their application development environment from Team Developer 1.5.1 to Team Developer 3.1 they will see a few notable changes in the behavior of applications that use certain ActiveX objects.

Most ActiveX objects have a type library associated with them. The Controls Palette and the ActiveX Explorer in Team Developer read the type library to determine which methods and events are exposed by the object, and they build a library (.APL) file from this information. But there are some ActiveX objects that either do not have type libraries, or have type libraries that are incomplete for purposes of automation.

Examples are the “Image Document” and “Bitmap Image” objects. With Team Developer 1.5.1 or earlier, one could drag and drop an ActiveX object onto a form and Team Developer would generate the required library (.APL) file that contains wrapper functions for all the methods exposed in the object’s type library. In the case of missing or incomplete type libraries, Team Developer 1.5.1 generated a library (.APL) file containing a visual component class of the ActiveX object derived from the dispatch interface object. The ActiveX class was not fully functional and did not contain the “Init” function required to instantiate an object of that class.

Beginning with Team Developer 3.0 this behavior was altered, preventing Team Developer from generating the library (.APL) file if the object didn’t expose sufficient functionality. Instead Team Developer throws an error stating, “This object does not support OLE Automation. You may need to re-install the component, or use it within the ActiveX container.”

When you encounter any such error during your application development using Team Developer 3.1, try inserting the object inside an ActiveX container. You can find more details on how to do this in the online book “Developing with SQLWindows”, in chapter 19, in the section titled “Using the Controls Palette”.

For migration of existing applications (developed using Team Developer 1.5.1 or earlier versions) that use ActiveX objects without a type library, these will continue to work in Team Developer 3.1 if you still have the generated library (.APL) file created by the earlier version of Team Developer.

About Unify

Unify is a global provider of software development technology and solutions that helps IT customers participate in Service-Oriented Architecture (SOA). Unify's productive and re-liable development tools, migration solutions and databases enable organizations to build and modernize business essential applications for SOA. Composer for Lotus Notes offers a complete, like for like, production to production migration solution for Lotus Notes applications. Unify's award-winning NXJ Developer enables IT teams to be extremely productive, learn new technologies fast and deliver Web services-based applications on time and on budget. The Team Developer, SQLBase, DataServer, ACCELL and VISION product families enable cross-platform rapid development on Java/J2EE, Linux or Windows. Unify has a rich heritage in delivering rich, cost-effective technologies to its thousands of IT customers and ISV, VAR and distributor partners. Unify is headquartered in Sacramento, Calif., and can be reached at (916) 928-6400 or by visiting www.unify.com.

Unify Corporation
2101 Arena Blvd., Suite 100
Sacramento, CA 95834
USA
Phone: 1.916.928.6400
Toll Free: 1.800.468.6439
Fax: 1.916.928.6404
Munich: +49 8 115 55430
United Kingdom: +44 (0)1753 245 510
France: +33 (0)1 34 58 28 30