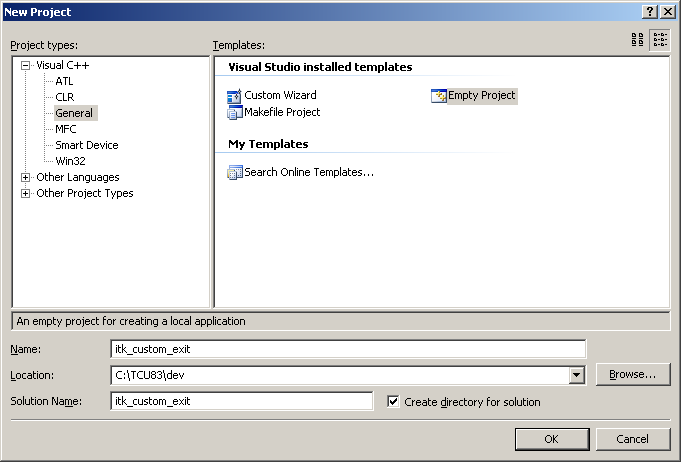
**Teamcenter - Configuring Visual Studio 2005 to build and debug ITK custom exits DLLs**

Currently, the Teamcenter ITK build tools are provided in the form of script files, such as compile.bat and link\_custom\_exit.bat. Unfortunately, the use of these script files don’t take advantage the many powerful features available in Visual Studio 2005 IDE, such as the Debugger, Symbol browsing, and project management. This document will take the reader through the steps needed to configure and build a debug and release versions of an ITK custom exit DLL and show how to use the debugger to debug the DLL.

Prerequisites:

* A Teamcenter 8.3 installation.
* Visual Studio 2005 w/SP1 or greater installed.

Start the Visual Studio IDE and create an empty Project called “itk\_custom\_exit.”



Create a new ‘itk\_custom\_exit.c’ file in the itk\_custom\_exit\itk\_custom\_exit folder and add the file to the project.

#include <ict\_userservice.h>

#include <tccore/custom.h>

#include <tccore/item\_msg.h>

#include <user\_exits/user\_exits.h>

extern DLLAPI int register\_userservices(int \*decision, va\_list list);

extern DLLAPI int itk\_custom\_exit\_register\_callbacks()

{

printf("itk\_custom\_exit\_register\_callbacks\n");

CUSTOM\_register\_exit("itk\_custom\_exit" /\* shared library name \*/,

"USERSERVICE\_register\_methods", /\* user\_exit name\*/

(CUSTOM\_EXIT\_ftn\_t)register\_userservices /\* custom function name \*/);

return ITK\_ok;

}

Next create file ‘item\_create\_post.c’ place it in the itk\_custom\_exit\itk\_custom\_exit folder as well, and add it to the project.

#include <ict\_userservice.h>

#include <tccore/custom.h>

#include <tccore/item\_msg.h>

#include <user\_exits/user\_exits.h>

#include <tccore/item.h>

typedef struct

{

const char\* item\_id;

const char\* item\_name;

const char\* type\_name;

const char\* rev\_id;

tag\_t\* new\_item;

tag\_t\* new\_rev;

tag\_t item\_master\_form;

tag\_t item\_rev\_master\_form;

} ITEM\_create\_va\_list;

extern DLLAPI int ItemCreatePost(METHOD\_message\_t\* msg, va\_list list)

{

ITEM\_create\_va\_list\* clist = (ITEM\_create\_va\_list\*)list;

printf("ItemId: %s, ItemName: %s, TypeName: %s, RevId: %s", clist->item\_id,

clist->item\_name, clist->type\_name, clist->rev\_id);

return ITK\_ok;

}

extern DLLAPI int register\_userservices(int \*decision, va\_list list)

{

int ec = ITK\_ok;

METHOD\_id\_t method;

\*decision = ALL\_CUSTOMIZATIONS;

ec = METHOD\_find\_method("Item", ITEM\_create\_msg, &method);

if(ec)

printf("Error %d - METHOD\_find\_method\n", ec);

if(method.id != NULLTAG)

{

ec = METHOD\_add\_action(method, METHOD\_post\_action\_type, (METHOD\_function\_t)ItemCreatePost, NULL);

if(ec != ITK\_ok)

printf("\tMETHOD\_add\_action error: %d!\n", ec);

else

printf("METHOD\_add\_action successful!\n");

}

else

printf("Method NOT found!\n", ec);

return ec;

}

At this point we have a basic ITK project created. Next we need to setup the project properties to compile and link the ITK sample code.

In Visual Studio the way the compiler and linker settings are managed are through Project Property Sheets. Below is a XML Project Property Sheets that contains the general properties to build both Debug and Release ITK custom exits.

Open your favorite Text editor, create a new file, and cut & paste the following XML text into the new file.

<?xml version="1.0" encoding="Windows-1252"?>

<VisualStudioPropertySheet

ProjectType="Visual C++"

Version="8.00"

Name="itk\_general\_exit (8.3)"

>

<Tool

Name="VCCLCompilerTool"

AdditionalIncludeDirectories="C:\TCU83\TC\_ROOT\include"

PreprocessorDefinitions="NT40;POMDLL;CRTAPI1=\_cdecl;CRTAPI2=cdecl;\_WIN32;WIN32;WIN32\_LEAN\_AND\_MEAN;WNT;BYPASS\_FLEX;\_INTEL=1;IPLIB=none"

CompileAs="1"

WarningLevel="1"

/>

<Tool

Name="VCLinkerTool"

AdditionalDependencies="libict.lib libsyss.lib libpom.lib libae.lib libappr.lib libarchive.lib libbackup.lib libbom.lib libcfm.lib libcxpom.lib libdmi.lib libecm.lib libeint.lib libepm.lib libfclasses.lib libform.lib libgrm.lib libics.lib libtcinit.lib libtc.lib libtccore.lib libtctrushape.lib libinternal\_exits.lib libvalidation.lib libitk.lib liblov.lib libme.lib libmechatronics.lib libmgc.lib libobjio.lib libpie.lib libtie.lib libproperty.lib libps.lib libpublication.lib libqry.lib libres.lib librdv.lib libsa.lib libss.lib libsub\_mgr.lib libtccoreext.lib libtextsrv.lib libnxmgr\_im.lib libbase\_utils.lib xerces270.lib"

DelayLoadDLLs="libuser\_exits.dll;libict.dll;libae.dll;libappr.dll;libarchive.dll;libbackup.dll;libbom.dll;libcfm.dll;libdmi.dll;libecm.dll;libeint.dll;libepm.dll;libform.dll;libgrm.dll;libics.dll;libtccore.dll;libtctrushape.dll;libinternal\_exits.dll;libvalidation.dll;libitk.dll;liblov.dll;libme.dll;libmechatronics.dll;libmgc.dll;libobjio.dll;libpie.dll;libtie.dll;libproperty.dll;libps.dll;libpublication.dll;libqry.dll;libres.dll;libsub\_mgr.dll;libtccoreext.dll;libnxmgr\_im.dll"

AdditionalLibraryDirectories="C:\TCU83\TC\_ROOT\lib"

SubSystem="1"

TargetMachine="1"

/>

</VisualStudioPropertySheet>

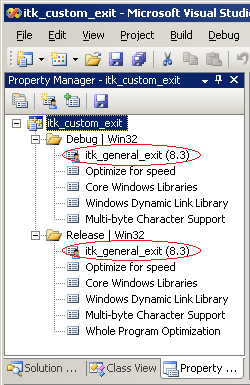
Once the XML file is in your editor:

* Verify no line wrapping is occurring; each property and value must be on one line. For example AdditionalDependencies="…” must be all on the same line.
* Verify “spaces” between library names are preserved.
* Update “AdditionalIncludeDirectories” and “AdditionalLibraryDirectories” nodes to reflect your environment paths.

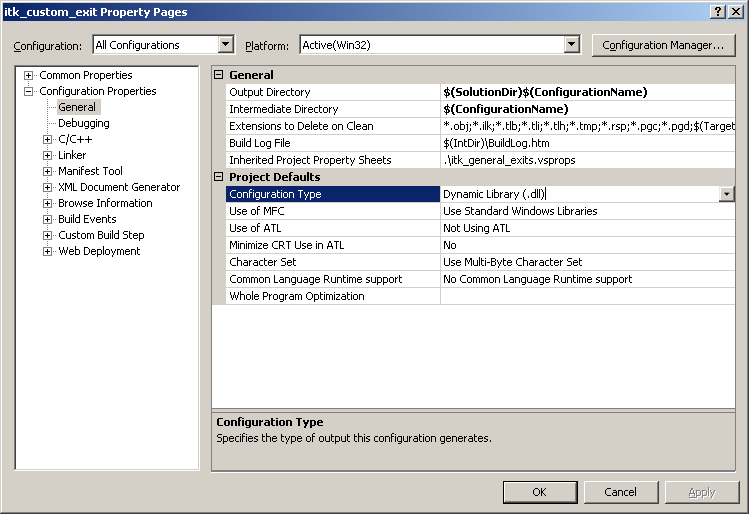
Save the file as “itk\_general\_exits.vsprops” in the itk\_custom\_exit\itk\_custom\_exit folder.

In the Visual Studio IDE open the Property Manager view. On the “Debug | Win32” node, right click, “Add Existing Property Sheet…”, and select “itk\_general\_exit.vsprops”. Likewise do the same for “Release | Win32”.

Your Property Manager View should look something like this:

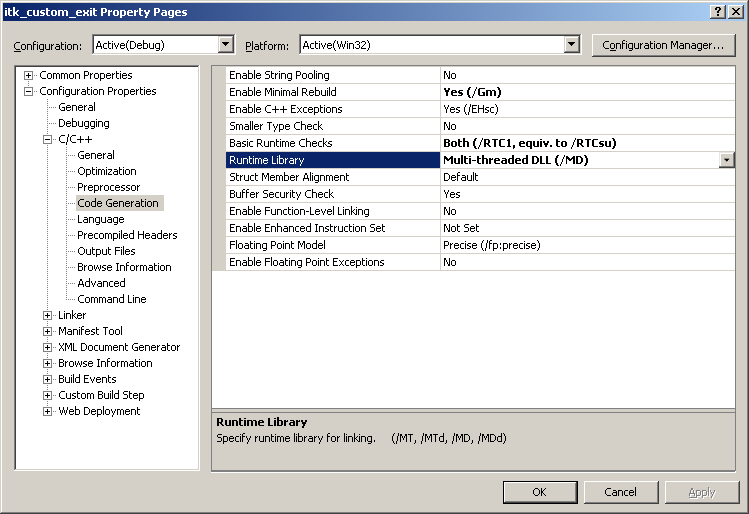


Open the project property pages and select for Configuration, “All Configurations”, select “General” under Configuration Properties, and on the right for “Configuration Type” set to “Dynamic Library (.dll)”.



Because Teamcenter only provides release ITK libraries to link, it is best to use the Microsoft Release runtime libraries when building the Debug target. This step will help eliminate some linker warnings.

Set the Configuration to “Debug” and under Code Generation use the /MD option for Runtime Library.



Click Ok and save the project.

Now the project should be ready to build the ITK custom exit DLL. Press F7 to build the current target which should be Debug, if not, set the target to Debug. After compiling and linking there should be no errors in the output window and a DLL should have been created. Note there maybe some linker warnings which in most cases can be ignored.

Note: To resolve the linker warning you’ll have to optimize the itk\_general\_exits.vsprops to only use the libraries specific to your project. Project property sheet itk\_general\_exits.vsprops is designed to be all inclusive.

**Configure Teamcenter to load custom exit DLLs**

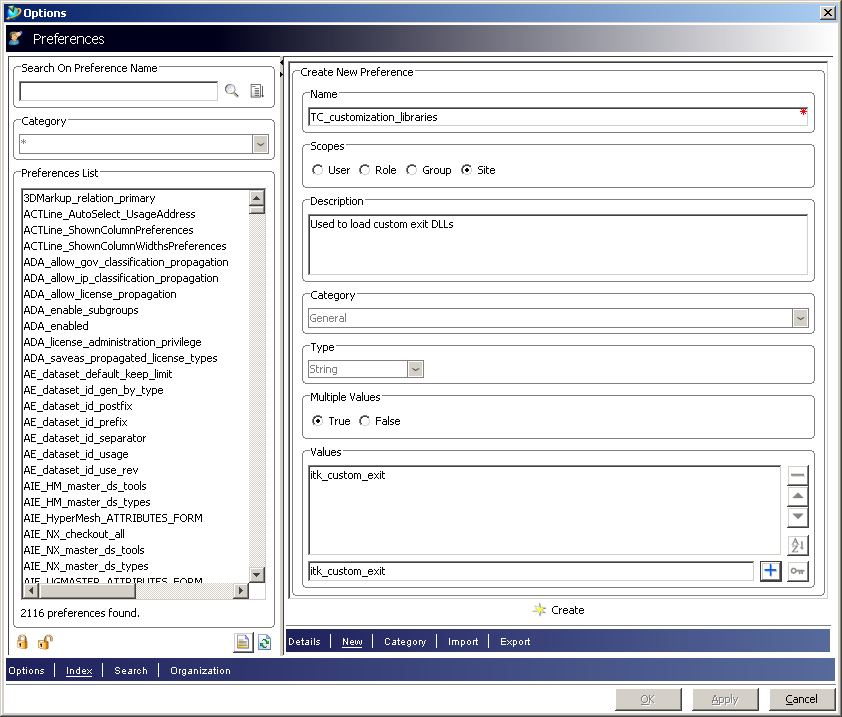
Now that we have our custom exit DLL built we need to configure Teamcenter to recognize and load our custom exit DLL.

First we need to set Teamcenter environment variable TC\_USER\_LIB to point to the location of our custom exit DLL. For example:

set TC\_USER\_LIB= C:\TCU83\dev\itk\_custom\_exit\debug

Set this environment variable in the %TC\_DATA%\tc\_profilevars.bat file.

Next we need to create or set Teamcenter TC\_customization\_libraries preference option. The following dialog show how the TC\_customization\_libraries is created.



Once created or updated, shut down the RAC client and TAO window.

To test the new custom exit DLL, start the RAC client, login, and create an Item. This should invoke the custom exit DLL and all custom output should be sent to the TAO window. Review the window for output.

**Using the Visual Studio Debugger to debug the Custom exit DLL.**

Before we get start launching the debugger to debug our custom exit DLL it is important first to understand how the Teamcenter IIOP server starts a tcserver.exe process. This important because the tcserver is the process that actually loads the custom exit DLL and understanding how this process works will help us understand what the debug options are to successfully debug the DLL.

The custom exit DLL is basically in two parts; one, the loading and registration code, and two, the custom exit code. The loading and registration happens only once during the life of the tcserver process, while the custom exit code can be call many times. With a little coordination with the Teamcenter IIOP server it will be possible to debug the loading and registration code of the custom exit DLL. Generally, the need to debug this code often is not necessary because the code for the most part remains static; nevertheless, being able to debug it on occasion is beneficial.

In Teamcenter there are really two modes to start a tcserver process, “NORMAL” (default during installation) and “PER\_CLIENT”. NORMAL limits one tcserver process for n-number of RAC clients and the process is started immediately; PER\_CLIENT starts a new tcserver process for every RAC client and delays the start of the process until the user logs in.

To debug the load and registration code of the custom exit DLL it is best to use the “NORMAL” mode to start the tcserver process. This is because the tcserver process is started immediately and the initial loading of the custom exit DLL doesn’t actually happen until the user logs.

To manage the tcserver.exe start process, I recommend becoming familiar with two Teamcenter files, %TC\_ROOT%\iiopservers\start\_imr.bat and %TC\_ROOT%\iiopservers\tcserver.xml. These two files control the activation mode.

For example, in start\_imr.bat notice:

set TCSERVER\_ACTIVATION\_MODE=NORMAL

In tcserver.xml notice option:

activation\_mode="NORMAL"

By switching the values you change how the IIOP server starts a tcserver process.

The importance of managing the activation mode for debugging may become clearer once we actually get in to debugging of the custom exit DLL. I will cover this next. At this point configure the activation mode to NORMAL.

In the Visual Studio set break points at the following lines of code using F7 key:

extern DLLAPI int ItemCreate(METHOD\_message\_t\* msg, va\_list list)

● {

extern DLLAPI int register\_userservices(int \*decision, va\_list list)

● {

extern DLLAPI int itk\_custom\_exit\_register\_callbacks()

{

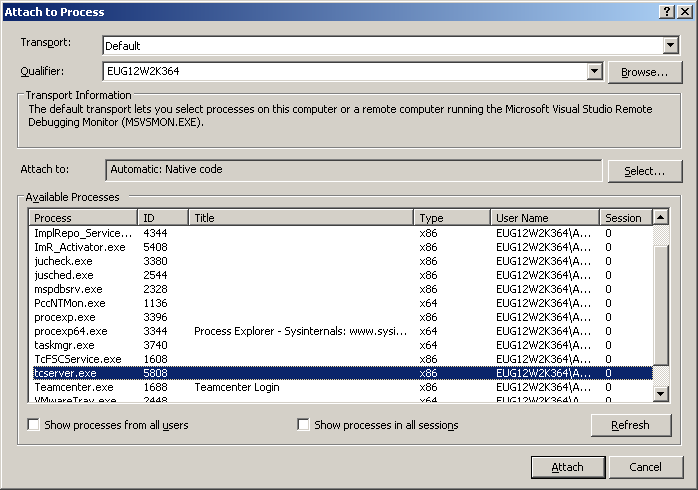
● printf("itk\_custom\_exit\_register\_callbacks\n");

Start the Teamcenter RAC client. This will automatically start the TAO window and bring up the Teamcenter login dialog.



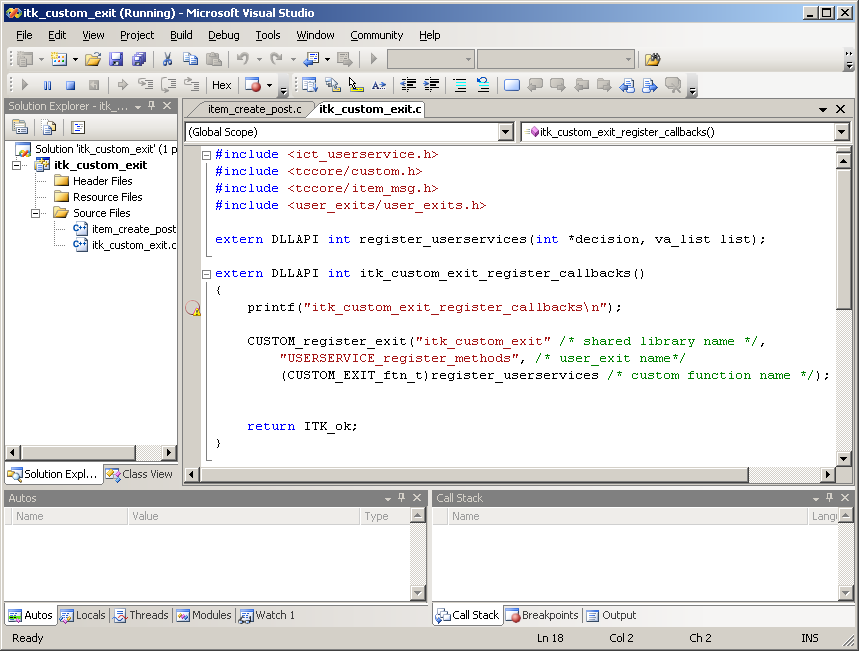
Do not login at this time and bring forward the Visual Studio IDE.

From the top menu, select Debug, Attach to Process…



The Attach Process dialog is displayed and scroll the Available Processes list to reveal the tcserver.exe process. Select it and click Attach.

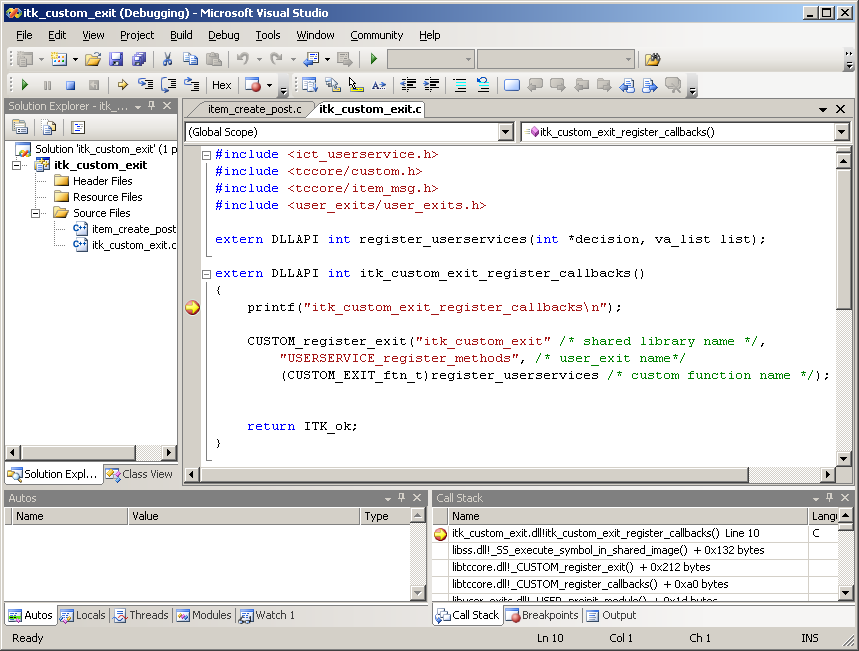
Note: If the activation mode was PER\_CLIENT the tcserver process is not listed at this time. In this mode, the server is not started until the user logs in.



Notice the break points have become tan color filled with a red edge; they are no longer solid red. This means the current process being debugged has yet to load the DLL resource.

Bring forward the Teamenter login screen and login.

Shortly, if all goes well, the break points should become solid red again, meaning the process has loaded the DLL into its address space and the debugger will break at the first break point in itk\_custom\_exit\_register\_callbacks() function. Also notice the Call Stack window has populated showing the current thread’s stack.



Pressing F10 key will step the debugger through the code line by line. Pressing F5 will run until the next break point. By pressing F5, eventually the debugger will break at the register\_userservices function. From here press F10 to step or F5 to run.

Once the load and registration code is done running the Teamcenter RAC interface is display.

Proceed to creating a new Item and once the item is created the ItemCreate function break point should be triggered.

At this point it should be clear on how to debug an ITK custom exit DLL.

Finally, I’d like to finish with additional comments in regards to NORMAL and PRE\_CLIENT activation modes. If the user were to start another RAC session and login the debugger will break at the registration code for the second time as the second RAC client comes up. This is expected because the second RAC client is using the same tcserver process as the first. However, in a PER\_CLIENT mode, starting a second RAC client session and logging in will start another tcserver process. Because this is a new tcserver process and the debugger is not attached to it the break points will not be triggered.

I invite the reader to work the different activation modes and learn the subtleties in the timing of attaching to a tcserver process.