Lightning Talk

Use of CA Plex Model API

Sergio Mendes
Lusodata

PLEX2E.COM COMMUNITY
RACE TO THE FUTURE 2023

cm FIRST
Rethink Modernization

AMI SOLUTIONS

BROADCOM SOFTWARE

A·B·A·C·U·S SOLUTIONS
BUSINESS UNIT OF FRESCHE SOLUTIONS

REMAIN SOFTWARE

LUSODATA

HAWKBRIDGE

Customer Infrastructure

Client / Server:

- IBM i (AS400) Servers
- Windows PC Clients

Software Updates Distribution (DevOps)

Development:

- Portugal

Tests/QA/Production Environments:

- Portugal
- Greece
- Egypt
- Saudi Arabia
- Morocco
- Spain

Object Collection and Distribution

Server Objects:

- Collected and distributed manually
- Manually updated by SysAdmin

Client Objects:

- DLLs / PNLs / EXEs
- How to collect the correct objects in a controlled way ?
- Deployed objects would have to be fetched by a “check for updates” mechanism

Know what objects to collect

Plex Lists:

– Are the main repository for logical object collection at development time

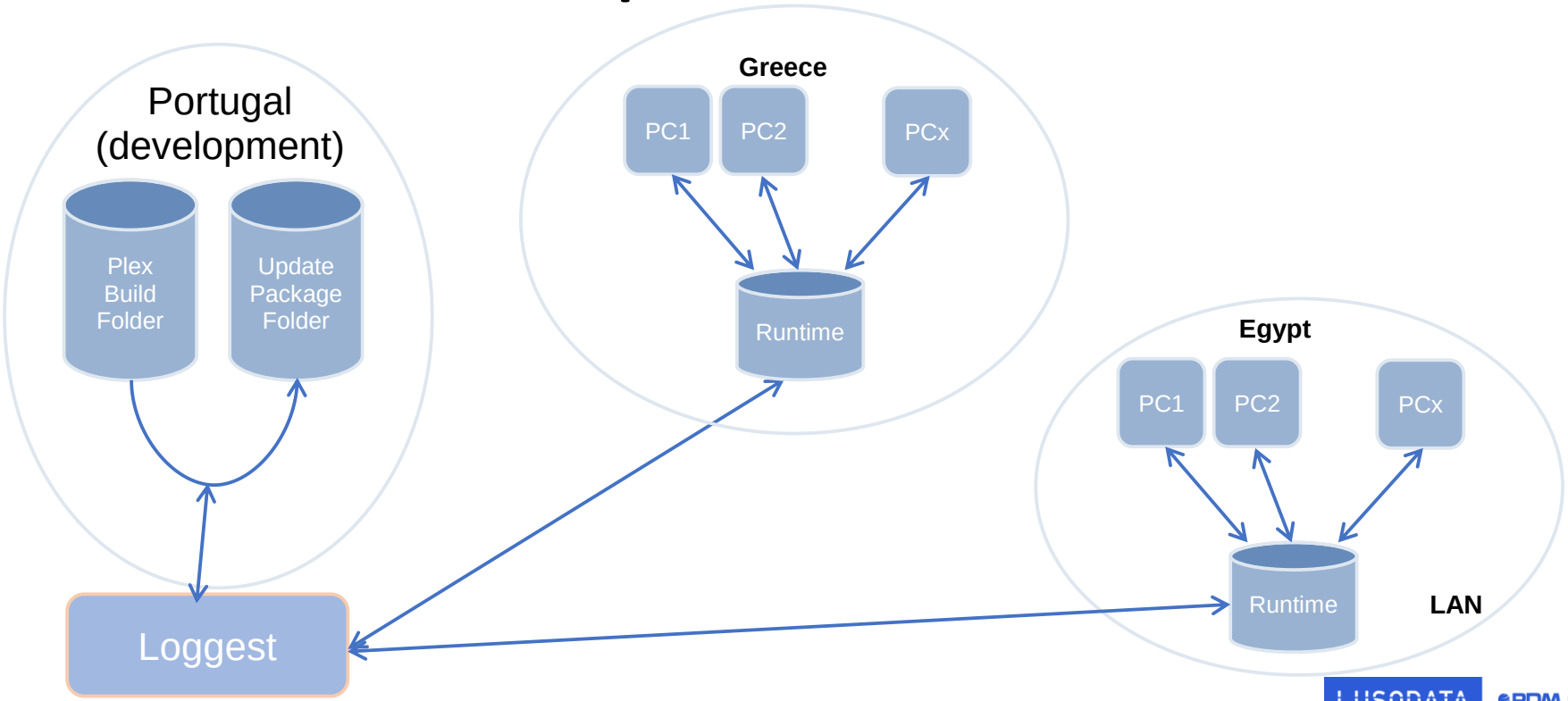
– Functions in the list are compiled, analyzed and their DLLs/PNLs stored inside “Update Packages”.

Solution:

- Make use of “CA Plex Model API” to process those lists

- Developed a Plex App, called: “LogGest”

Software Updates Distribution



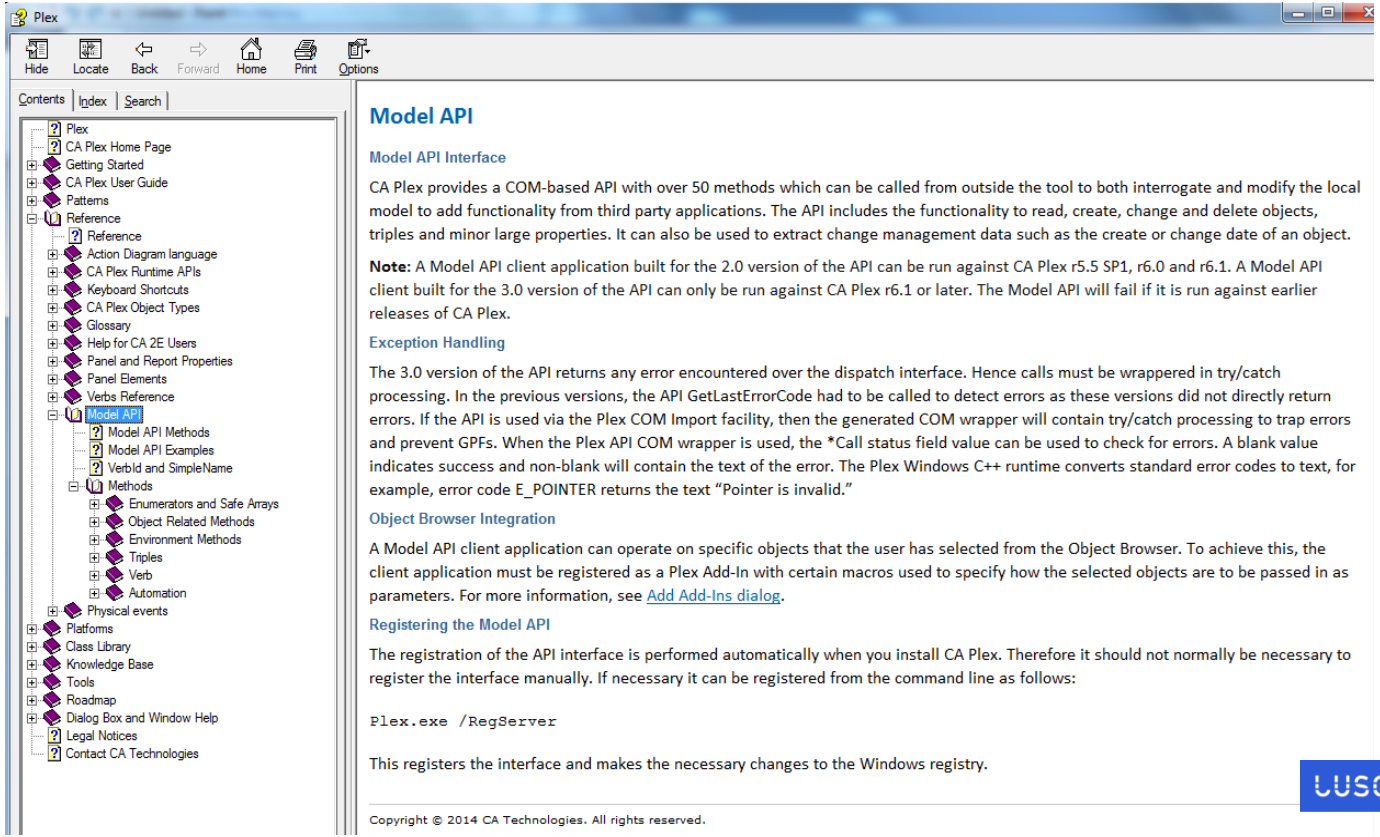
The Model API (using C++)

Plex Model API: COM-based API with over 50 methods

- Allows external applications to interrogate and even modify Local Models.
- Can be used to extract “Change Management” data
- Documented in Plex Help
- Used version 1.0 back in 2007

Current Version is 3.0, has better exception handling.

Plex Model API Documentation



Plex Model API – the source codes

Step 1: initialize the COM Framework.

```
#include <wizardapidefs.h>
#include "import.h"

&(3:) = (ObLongFld)CoInitializeEx(NULL,(DWORD)&(2:));
```

Parameters
- &(1:) MDDebugOn ?
- &(2:) dwCoInit
- &(3:) hrReturnCode

```
&(2:) -> Concurrency Level (usually, COINIT_MULTITHREADED)
&(3:) -> Return Code (S_OK is good)
```

Step 2: get a pointer for the List object, specified by its name.

```
#include <wizardapidefs.h>
#include "import.h"
{

    int ret_val;
    long listobject = 0;

    try {
        IPlexAPIPtr p(__uuidof(PlexAPI));
        ret_val = p->FindObj((LPCTSTR)&(1:), _TypeList, &listobject);

        p = NULL;
    }
    catch (_com_error e) {
       AfxMessageBox(e.ErrorMessage());
    }


    &(2:) = (ObLongFld)listobject;
}
```

List Name (parameter 1)

List object pointer (used by other APIs)

Parameters

- &(1:) LG List Name
- &(2:) LG List Ptr

Enumerate List Objects (fncs)

Step 3: get “Verb” pointer and “VerbID” (from simple Verb: “LstContainsFnc”)

```
#include <wizardapidefs.h>
#include "import.h"
{
&(4:) = " ";


    int ret_val;
    long enumerator = 0;
    long verb = 0L;
    long verbID = 0L;

    try {
        IPlexAPIPtr p(__uuidof(PlexAPI));

        // Get Verb object...
        ret_val = p->FindVerbByName((LPCSTR)&(1:), SimpleName, &verb, &verbID);

        p = NULL;
    }
    catch (_com_error e) {
        &(4:) = (LPCSTR)e.ErrorMessage();
       //AfxMessageBox(e.ErrorMessage());
    }


    &(2:) = (ObLongFld)verb;
    &(3:) = (ObLongFld)verbID;
}
```

Verb name (simple format)
In this case, we use "LstContainsFnc"

Verb pointer

VerbID (will be used by the List enumerator)

Parameters
- &[1:] LG Simple Verb
- &[2:] LG Verb Ptr
- &[3:] LG Verb ID Ptr
- &[4:] *Message text

Enumerate List Objects (fncs)

Step 4: use “VerbID” from step 3 and get enumerator pointer:

```
#include <wizardapidefs.h>
#include "import.h"
{

    int ret_val;
    long enumerator = 0;
    &(4:) = " ";

    try {
        IPlexAPIPtr p(__uuidof(PlexAPI));

        ret_val = p->EnumTriplesBySource((long)&(1:), (long)&(2:), &enumerator);

        p = NULL;
    }
    catch (_com_error e) {
       //AfxMessageBox(e.ErrorMessage());
        &(4:) = (LPCSTR)e.ErrorMessage();
    }


    &(3:) = (ObLongFld)enumerator;
}
```

List object pointer (obtained in the first place)

VerbID (ListContainsFnc)

Pointer to the enumerator object

Parameters

- &(1:) LG Object Ptr
- &(2:) LG Verb ID Ptr
- &(3:) LG Enumerator Ptr
- &(4:) *Message text

Enumerate List Objects (fncs)

Step 5: get the number of objects in the enumerator (using the enumerator pointer returned before):

```
#include <wizardapidefs.h>
#include "import.h"
{

    int ret_val;
    long count = 0L;

    try {
        IPlexAPIPtr p(__uuidof(PlexAPI));

        ret_val = p->GetCountOfItemsInEnumerator((long)&(1:), &count);
        p = NULL;
    }
    catch (_com_error e) {
       AfxMessageBox(e.ErrorMessage());
    }


    &(2:) = (ObLongFld)count;
}
```

Pointer to the enumerator

Nr of objects in the enumerator

Parameters
- &(1:) LG Enumerator Ptr
- &(2:) LG Nr of Objects

Process “LstContainsFNC” element:

Step 6: get first object’s pointer (should be a triple whose target is a Function):

```
#include <wizardapidefs.h>
#include "import.h"
{

    int ret_val;
    long triple = 0;
    &(3:) = " ";

    try {
        IPlexAPIPtr p(__uuidof(PlexAPI));
        ret_val = p->GetFirstTriple((long)&(1:), &triple);

        p = NULL;
    }
    catch (_com_error e) {
       //AfxMessageBox(e.ErrorMessage());
        &(3:) = (LPCSTR)e.ErrorMessage();
    }


    &(2:) = (ObLongFld)triple;
}
```

Enumerator pointer

Pointer to the first triple object (Lists contain triples and their targets are the Functions we need to collect and process)

Parameters

- &(1:) LG Enumerator Ptr
- &(2:) LG Triple Ptr
- &(3:) *Message text

Process “LstContainsFNC” element:

Step 7: inside a While, obtain the triple’s target, which is a pointer to a Function object:

```
#include <wizardapidefs.h>
#include "import.h"
{

    int ret_val;
    long object = 0;

    try {
        IPlexAPIPtr p(__uuidof(PlexAPI));
        long listriple = 0;
        ret_val = p->GetTripleTarget((long)&(1:), &object, &listriple);

        p = NULL;
    }
    catch (_com_error e) {
       AfxMessageBox(e.ErrorMessage());
    }


    &(2:) = (ObLongFld)object;
}
```

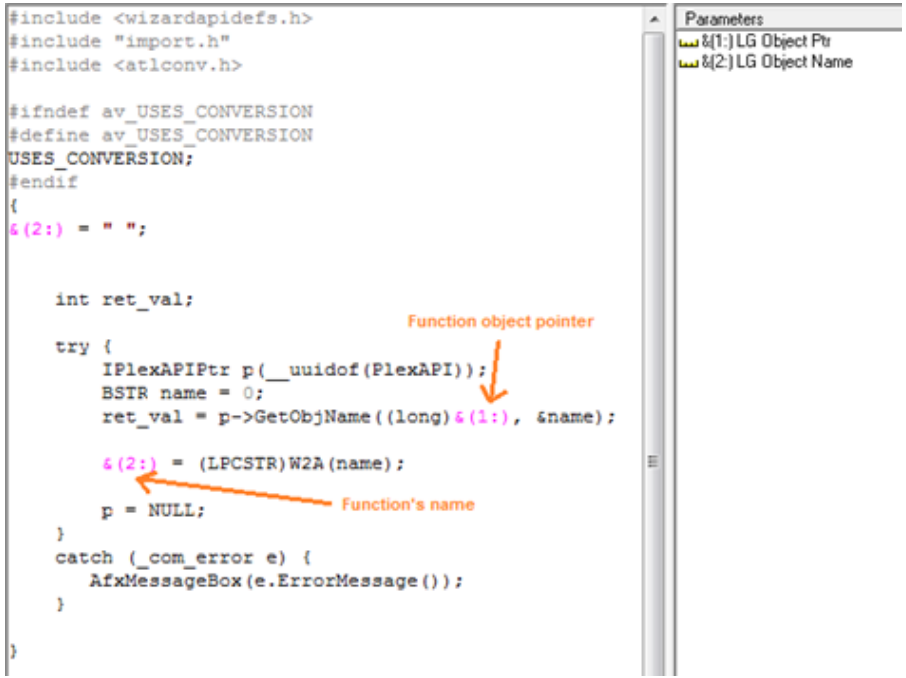
Triple pointer (obtained in the previous step)

Pointer to our Function object (which is a member of the Plex List we used to collect changed objs)

Parameters
- &(1:) LG Triple Ptr
- &(2:) LG Object Ptr

Process Target Object (FNC):

Step 8: get target object's (Function's) name:



W2A macro: is used to convert widechar (COM) into ansi (Windows 1252)

Process Target Object (FNC):

Step 9: get object's (Function's) type:

```
#include <wizardapidefs.h>
#include "import.h"
#include <atlconv.h>

#ifndef av_USES_CONVERSION
#define av_USES_CONVERSION
USES_CONVERSION;
#endif
{
&(2:) = " ";


    int ret_val;

    try {
        IPlexAPIPtr p(__uuidof(PlexAPI));
        BSTR sObjType = 0;
        ret_val = p->GetObjTypeAsText((long)&(1:), &sObjType);

        &(2:) = (LPCSTR)W2A(sObjType);

        p = NULL;
    }
    catch (_com_error e) {
       AfxMessageBox(e.ErrorMessage());
    }

}
```

Function obj pointer

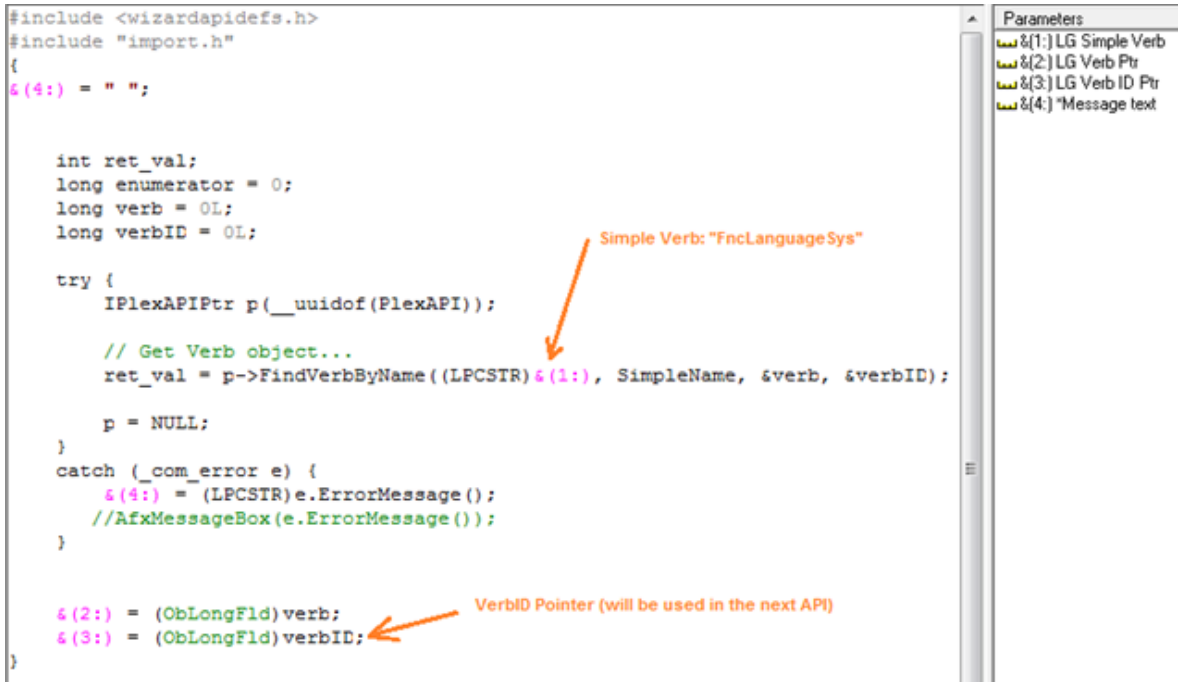
Object type ("Function")

Parameters

- &(1:) LG Object Ptr
- &(2:) LG Object Type

Process Target Object (FNC):

Step 10: get Function's language (1st, get VerbID: "FncLanguageSys" *):



* actually, function's language is another triple, needs a VerbID to start another enumeration. Obviously, it should occur only once.

Process Target Object (FNC):

Step 11: get Function's language (start another enumeration using VerbID obtained in previous step):

```
#include <wizardapidefs.h>
#include "import.h"
{

    int ret_val;
    long enumerator = 0;
    &(4:) = " ";

    try {
        IPlexAPIPtr p(__uuidof(PlexAPI));

        ret_val = p->EnumTriplesBySource((long)&(1:), (long)&(2:), &enumerator);

        p = NULL;
    }
    catch (_com_error e) {
       //AfxMessageBox(e.ErrorMessage());
        &(4:) = (LPCSTR)e.ErrorMessage();
    }


    &(3:) = (ObLongFld)enumerator;
}
```

Function object ptr

VerbID ptr ("FncLanguageSys") (from previous step)

Enumeration ptr

Parameters

- &(1:) LG Object Ptr
- &(2:) LG Verb ID Ptr
- &(3:) LG Enumerator Ptr
- &(4:) *Message text

NOTE: the enumeration pointer must be stored in another field, must not spoil the List enumerator pointer, which was obtained in the step number 4.

Process Target Object (FNC):

Step 12: get enumerator's first (and only) triple "FNC Language SYS":

```
#include <wizardapidefs.h>
#include "import.h"
{

    int ret_val;
    long triple = 0;
    &(3:) = " ";

    try {
        IPlexAPIPtr p(__uuidof(PlexAPI));
        ret_val = p->GetFirstTriple((long)&(1:), &triple);

        p = NULL;
    }
    catch (_com_error e) {
       //AfxMessageBox(e.ErrorMessage());
        &(3:) = (LPCSTR)e.ErrorMessage();
    }


    &(2:) = (ObLongFld)triple;
}
```

Enumerator ptr

Pointer to a triple object (will be used next to get its target)

Parameters

- &(1:) LG Enumerator Ptr
- &(2:) LG Triple Ptr
- &(3:) *Message text

Process Target Object (FNC):

Step 13: get triple's target object (language):

```
#include <wizardapidefs.h>
#include "import.h"
{

    int ret_val;
    long object = 0;

    try {
        IPlexAPIPtr p(__uuidof(PlexAPI));
        long listriple = 0;
        ret_val = p->GetTripleTarget((long)&(1:), &object, &listriple);

        p = NULL;
    }
    catch (_com_error e) {
       AfxMessageBox(e.ErrorMessage());
    }


    &(2:) = (ObLongFld)object;
}
```

Triple's pointer (obtained in previous step)

Pointer to the target object from which we can obtain its text (the fnc's language)

Parameters
- &(1:) LG Triple Ptr
- &(2:) LG Object Ptr

Process Target Object (FNC):

Step 14: get triple's target object name (fnc's language):

```
#include <wizardapidefs.h>
#include "import.h"
#include <atlconv.h>

#ifndef av_USES_CONVERSION
#define av_USES_CONVERSION
USES_CONVERSION;
#endif
{
&(2:) = " ";


    int ret_val;

    try {                                   Triple's target object pointer
        IPlexAPIPtr p(__uuidof(PlexAPI));
        BSTR name = 0;
        ret_val = p->GetObjName((long)&(1:), &name);

        &(2:) = (LPCSTR)W2A(name);
                Triple's target object name (fnc's language)
        p = NULL;
    }
    catch (_com_error e) {
       AfxMessageBox(e.ErrorMessage());
    }

}
```

Parameters
- &(1:) LG Object Ptr
- &(2:) LG Object Name

Process Target Object (FNC):

Step 15: get rid of the FncLanguageSys enumerator (free the ptr):

```
#include <wizardapidefs.h>
#include "import.h"
{

    int ret_val;
    long enumerator = (long)&(1:);

    try {
        IPlexAPIPtr p(__uuidof(PlexAPI));

        ret_val = p->EndEnum(&enumerator);

        p = NULL;
    }
    catch (_com_error e) {
       AfxMessageBox(e.ErrorMessage());
    }

}
```

"FncLanguageSys" triple enumerator pointer

Parameters

&(1:) LG Enumerator Ptr

Process Target Object (FNC):

Step 16: get next (LstContainsFnc) triple, then repeat all previous steps until there are no more triples:

```
#include <wizardapidefs.h>
#include "import.h"
{

    int ret_val;
    long triple = 0;

    try {
        IPlexAPIPtr p(__uuidof(PlexAPI));
        ret_val = p->GetNextTriple((long)&(1:), &triple);

        p = NULL;
    }
    catch (_com_error e) {
       AfxMessageBox(e.ErrorMessage());
    }


    &(2:) = (ObLongFld)triple;
}
```

Triple enumerator pointer (LstContainsFnc)

Pointer to the next "LstContainsFnc" triple

Parameters

- &(1:) LG Enumerator Ptr
- &(2:) LG Triple Ptr

Head back to step 7 (slide 15) and do it all again. Step 7 API will be fed with this triple pointer.

Clean up:

Step 17: deallocate triples enumerator:

```
#include <wizardapidefs.h>
#include "import.h"
{

    int ret_val;
    long enumerator = (long)&(1:);

    try {
        IPlexAPIPtr p(__uuidof(PlexAPI));

        ret_val = p->EndEnum(&enumerator);

        p = NULL;
    }
    catch (_com_error e) {
       AfxMessageBox(e.ErrorMessage());
    }

}
```

Pointer to the "LstContainsFnc" triple enumerator

Clean up:

Last Step: Unitialize (free) COM

```
CoUninitialize();
```


LogGest DEMO

Contact

+1 207-691-2908

+351 21 417 35 67 (Portugal)

sergio.mendes@lusodata.pt

www.lusodata.pt