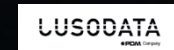


Lightning Talk

Use of CA Plex Model API



Sergio Mendes
Lusodata



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

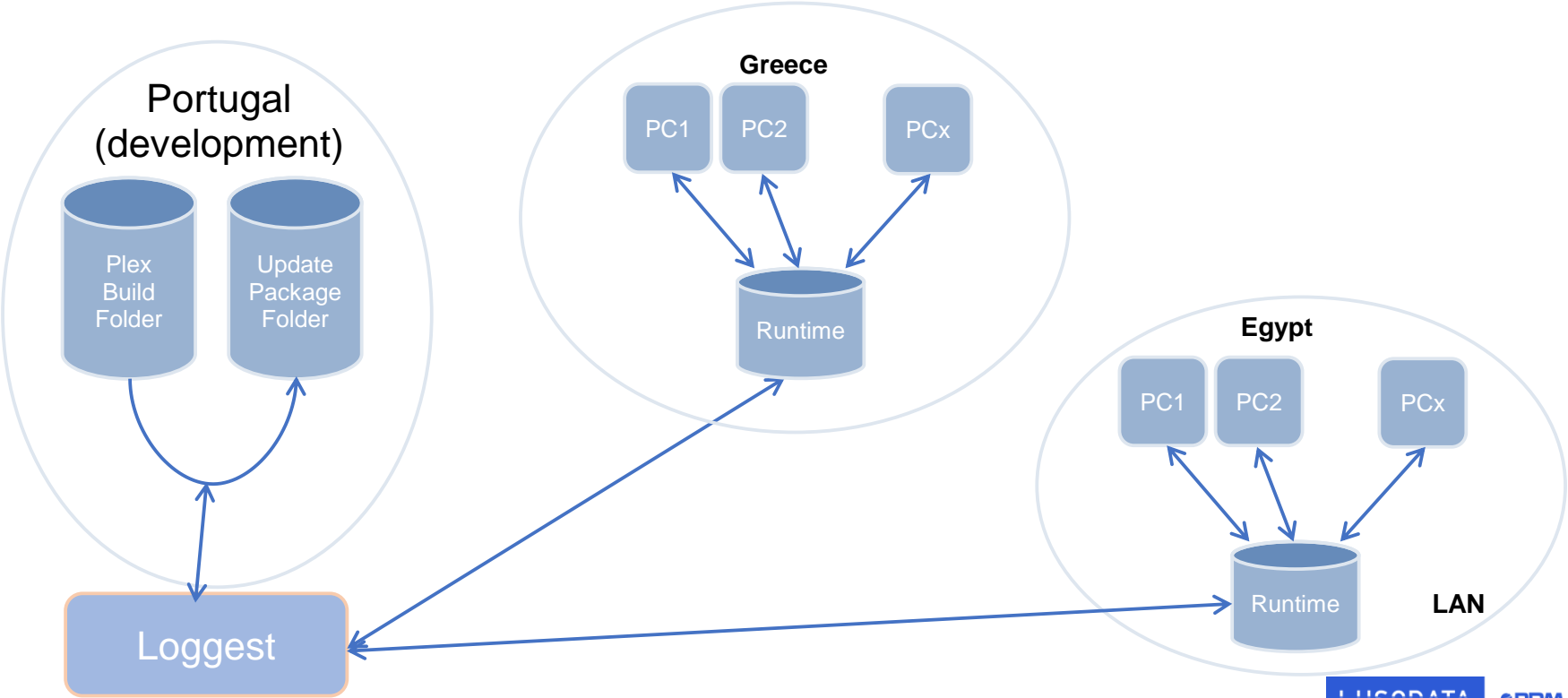
Plex Lists:

- 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



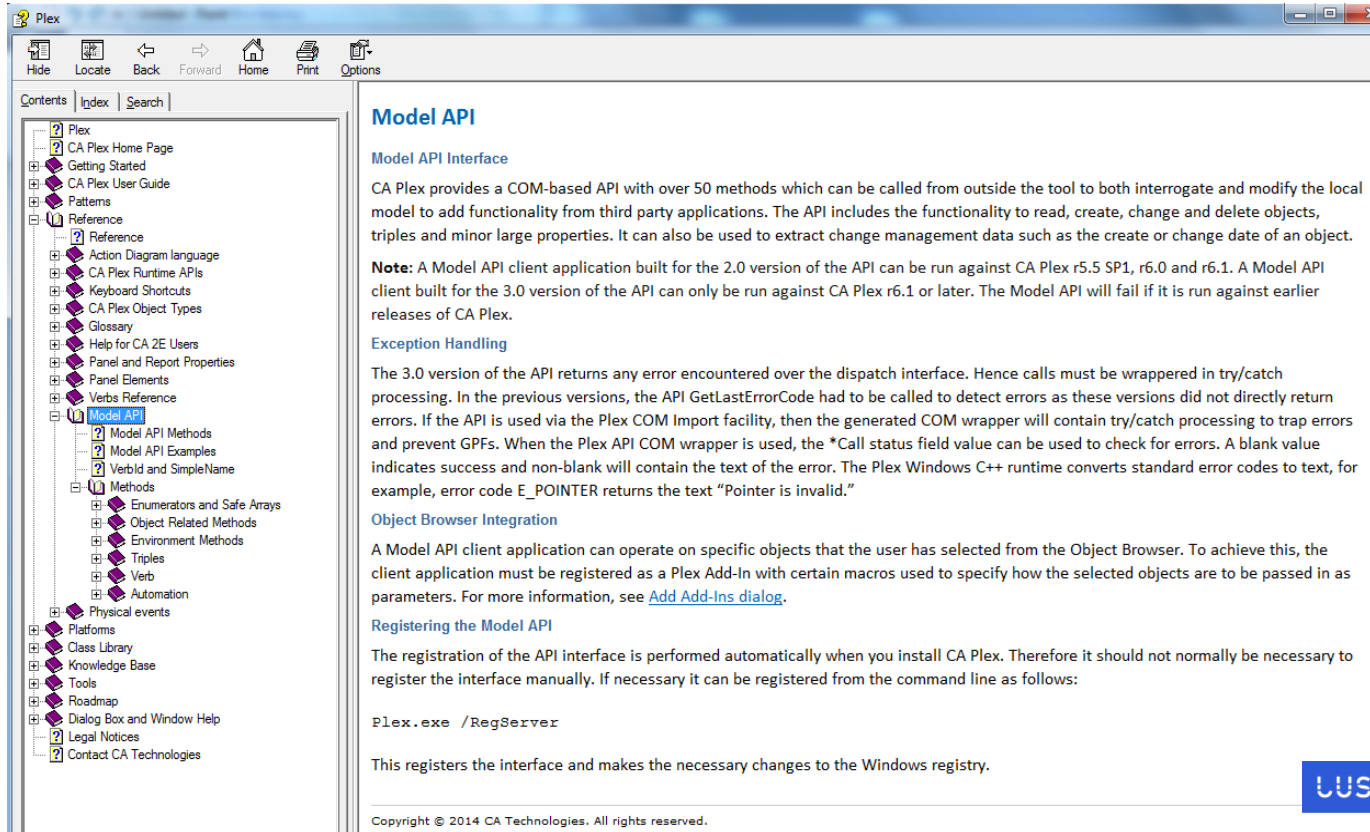
The Model API (using C++)

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



Model API

Model API Interface

CA Plex provides a COM-based API with over 50 methods which can be called from outside the tool to both interrogate and modify the local model to add functionality from third party applications. The API includes the functionality to read, create, change and delete objects, triples and minor large properties. It can also be used to extract change management data such as the create or change date of an object.

Note: A Model API client application built for the 2.0 version of the API can be run against CA Plex r5.5 SP1, r6.0 and r6.1. A Model API client built for the 3.0 version of the API can only be run against CA Plex r6.1 or later. The Model API will fail if it is run against earlier releases of CA Plex.

Exception Handling

The 3.0 version of the API returns any error encountered over the dispatch interface. Hence calls must be wrapped in try/catch processing. In the previous versions, the API `GetLastErrorCode` had to be called to detect errors as these versions did not directly return errors. If the API is used via the Plex COM Import facility, then the generated COM wrapper will contain try/catch processing to trap errors and prevent GPFs. When the Plex API COM wrapper is used, the `*Call` status field value can be used to check for errors. A blank value indicates success and non-blank will contain the text of the error. The Plex Windows C++ runtime converts standard error codes to text, for example, error code `E_POINTER` returns the text "Pointer is invalid."

Object Browser Integration

A Model API client application can operate on specific objects that the user has selected from the Object Browser. To achieve this, the client application must be registered as a Plex Add-In with certain macros used to specify how the selected objects are to be passed in as parameters. For more information, see [Add Add-Ins dialog](#).

Registering the Model API

The registration of the API interface is performed automatically when you install CA Plex. Therefore it should not normally be necessary to register the interface manually. If necessary it can be registered from the command line as follows:

```
Plex.exe /RegServer
```

This registers the interface and makes the necessary changes to the Windows registry.

Copyright © 2014 CA Technologies. All rights reserved.

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:) dwColnit
- &(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 (fncls)

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 (fncls)

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;
}

Parameters
┌───┴───┐
│ &(1:) LG Object Ptr │
│ &(2:) LG Verb ID Ptr │
│ &(3:) LG Enumerator Ptr │
│ &(4:) *Message text │
└───┴───┘
```

List object pointer (obtained in the first place)

VerbID (ListContainsFnc)

Pointer to the enumerator object

Enumerate List Objects (fncls)

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;
}

Parameters
└─ &(1:) LG Enumerator Ptr
└─ &(2:) LG Nr of Objects
```

Pointer to the enumerator

Nr of objects in the enumerator

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:

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

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

    int ret_val;

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

        &(2:) = (LPCSTR)W2A(name);
        p = NULL;
    }
    catch (_com_error e) {
        AfxMessageBox(e.ErrorMessage());
    }
}
```

Function object pointer

Function's name

Parameters

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

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

    int ret_val;

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

        &(2:) = (LPCSTR)W2A(sObjType);
        p = NULL; Object type ("Function")
    }
    catch (_com_error e) {
        AfxMessageBox(e.ErrorMessage());
    }
}
}
```

Parameters

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

Process Target Object (FNC):

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

```
#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;
}

Parameters
&(1:) LG Simple Verb
&(2:) LG Verb Ptr
&(3:) LG Verb ID Ptr
&(4:) Message text
```

Simple Verb: "FncLanguageSys"

VerbID Pointer (will be used in the next API)

* 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;
}

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;
}

Parameters
┌───┴───┐
└───┬───┘
    &(1:) LG Triple Ptr
    &(2:) LG Object Ptr
```

Triple's pointer (obtained in previous step) ↓

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

Process Target Object (FNC):

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

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

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

    int ret_val;

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

        &(2:) = (LPCSTR)W2A(name);
        p = NULL;
    }
    catch (_com_error e) {
        AfxMessageBox(e.ErrorMessage());
    }
}
```

Parameters

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

Triple's target object pointer

Triple's target object name (fnc's language)

Process Target Object (FNC):

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

```
#include <wizardapidefs.h>
#include "import.h"      "FncLanguageSys" triple
                        enumerator pointer
{
    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());
    }
}
```

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;
}

Parameters
┌───┴───┐
└───┬───┘
    &(1:) LG Enumerator Ptr
    &(2:) LG Triple Ptr
```

Triple enumerator pointer (LstContainsFnc) ↓

↑ Pointer to the next "LstContainsFnc" triple

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

Parameters
&(1:) LG Enumerator Ptr

Clean up:

Last Step: Unitialize (free) COM

```
CoUninitialize();
```

LogGest DEMO

LUSO DATA

PDM Company

Contact



+1 207-691-2908

+351 21 417 35 67 (Portugal)



sergio.mendes@lusodata.pt



www.lusodata.pt